The Linux Knowledge Base and Tutorial

16 January 2005

Table of Contents

Terms of Use and Disclamier of Warranty

YOU UNDERSTAND AND AGREE THAT YOUR USE OF THIS DOCUMENT AND ANY CONTENT PROVIDED IS MADE AVAILABLE AND PROVIDED TO YOU **AT YOUR OWN RISK**. IT IS PROVIDED TO YOU "AS IS" AND WE EXPRESSLY DISCLAIM ALL WARRANTIES OF ANY KIND, IMPLIED OR EXPRESSED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT.

WE MAKE NO WARRANTY, IMPLIED OR EXPRESS, THAT ANY PART OF THIS DOCUMENT WILL BE ERROR-FREE, TIMELY, ACCURATE, RELIABLE, OF ANY QUALITY, NOR THAT ANY CONTENT IS SAFE IN ANY MANNER FOR DOWNLOAD. YOU UNDERSTAND AND AGREE THAT NEITHER US NOR ANY PARTICIPANT IN THE SERVICE OF PROVIDING YOU THIS DOCUMENT OR THE DOCUMENT ITSELF PROVIDES PROFESSIONAL ADVICE OF ANY KIND AND THAT USE OF SUCH ADVICE OR ANY OTHER INFORMATION IS SOLELY **AT YOUR OWN RISK** AND WITHOUT OUR LIABILITY OF ANY KIND.

THE OPERATORS OF THE LINUX TUTORIAL AND ANY OTHER PERSONS OR ORGANIZATIONS THAT MAY PROVIDE THIS DOCUMENT ARE UNDER NO OBLIGATION TO CONTINUE DOING SO AND MAY, WITHOUT NOTICE OR LIABAILITY, DISCONTINUE PROVIDING THIS DOCUMENT.

FOR ADDITIONAL DETAILS SEE http://www.linux-tutorial.info/termsofuse.html.

Copyright

The material contained in this documented is released under various licenses. All of the material in this document is the copyright of the original creator (author, artist). Some of this material has been previously published in various places, but the creator stills own the copyright. This material is not only protected by the moral obligation to protect the creators' rights, but also by a number of different laws (which might not discourage some people). Much of the material was taken from the books Linux User's Resource and Supporting Windows NT and 2000 Server and Workstation, written by James Mohr and which were published Pearson Education Inc.

All material created by James Mohr is copyrighted 1994-2003 by James Mohr and is licensed under a modified GNU Free Documentation License. Reproduction and distribution of material copyrighted by James Mohr or derivative of such material in any standard (paper) book form is prohibited unless prior permission is obtained from the copyright holder. Distribution of substantively modified versions of material copyrighted by James Mohr is prohibited without the explicit permission of the copyright holder. Distribution of material copyrighted by James Mohr in a commercial product is prohibited without the explicit permission of the copyright holder. Use of material copyrighted by James Mohr in any commercial endeavour is prohibited without the explicit permission of the copyright holder. A "commercial endeavour" includes, but is not limited to training or other educational courses for which a fee is required. Public education institutions (such a state universities and colleges, community colleges and similar) are except from this requirement and need only inform the copyright holder of such use.

A number of pages in the tutorial do not fall within the above mentioned license and are licensed under various other license agreements. Each author or license holder has the right to determine the terms of the license under which the material is made available.

FOR ADDITIONAL DETAILS SEE http://www.linux-tutorial.info/copyright.html.

Preface

This is a **BETA** release of the "print friendly" version of the Linux Knowledge Base and Tutorial. Here are a few things to note:

- Due to technical problems, this version does **not** contain the chapter on Linux operating system internals.
- Graphics are unchanged from the live tutorial and thus many contain the "Play Animation" button.
- There is no index, glossary, list of images or list of tables. (I am not sure if I will ever do all of these).
- Some sections do not have any content, just like the live tutorial.

In that this is a **BETA** release, there are probably a lot of these wrong with it. Please let me know of anything you are not happy with by sending me feedback through the Linux Tutorial web site: http://www.linux-tutorial.info/feedback.html.

Please keep in mind that this is not intended to be a commercial product. It is **only** intended as a convenience for the users of the Linux Knowledge Base and Tutorial. Don't expect me to be spending any time getting the exact font that *you*. I just don't have the bandwidth. On the other hand, if more people starting volunteering to help, I would have more time to work on things like this document.

I am not going to make any promises of how often I will create a "print friendly" version. I am going to clean up several things before I create a first "official release". However, once that is done, I am going to work on a number of other things that I have been meaning to do so it will definitely be several months before I do another "print friendly" version.

Getting the material evenm this far was not a simple task. I had to write some addition code, as well as become familiar with a couple of new programs. This took time and effort, for which I am not asking any fee. However, donations to the Jessie Arbogast Medical Fund are always welcome.

At this point, I am only going to make this document available for download from SourceForge. My site just doesn't have the bandwidth.

Chapter 1 Introduction to Operating Systems

It is a common occurrence to find users who are not even aware of what operating system they are running. On occasion, you may also find an administrator who knows the name of the operating system, but nothing about the inner workings of it. In many cases, they have no time as they are often clerical workers or other personnel who were reluctantly appointed to be the system administrator.

Being able to run or work on a Linux system does not mean you must understand the intricate details of how it functions internally. However, there are some operating system concepts that will help you to interact better with the system. They will also serve as the foundation for many of the issues we're going to cover in this section.

In this chapter we are going to go through the basic composition of an operating system. First, we'll talk about what an operating system is and why it is important. We are also going to address how the different components work independently and together.

My goal is not to make you an expert on operating system concepts. Instead, I want to provide you with a starting point from which we can go on to other topics. If you want to go into more detail about operating systems, I would suggest *Modern Operating Systems* by Andrew Tanenbaum, published by Prentice Hall, and *Operating System Concepts* by Silberschatz, Peterson, and Galvin, published by Addison-Wesley. Another is *Inside Linux* by Randolph Bentson, which gives you a quick introduction to operating system concepts from the perspective of Linux.

1.1 What Is an Operating System

In simple terms, an operating system is a manager. It manages all the available resources on a computer. These resources can be the hard disk, a printer, or the monitor screen. Even memory is a resource that needs to be managed. Within an operating system are the management functions that determine who gets to read data from the hard disk, what file is going to be printed next, what characters appear on the screen, and how much memory a certain program gets.

Once upon a time, there was no such thing as an operating system. The computers of forty years ago ran one program at a time. The computer programmer would load the program he (they were almost universally male at that time) had written and run it. If there was a mistake that caused the program to stop sooner than expected, the programmer had to start over. Because there were many other people waiting for their turn to try their programs, it may have been several days before the first programmer got a chance to run his deck of cards through the machine again. Even if the program did run correctly, the programmer probably never got to work on the machine directly. The program (punched cards) was fed into the computer by an operator who then passed the printed output back to the programmer several hours later.

As technology advanced, many such programs, or jobs, were all loaded onto a single tape. This tape was then loaded and manipulated by another program, which was the ancestor of today's operating systems. This program would monitor the behavior of the running program

and if it misbehaved (crashed), the monitor could then immediately load and run another. Such programs were called (logically) monitors.

In the 1960's, technology and operating system theory advanced to the point that many different programs could be held in memory at once. This was the concept of "multiprogramming." If one program needed to wait for some external event such as the tape to rewind to the right spot, another program could have access to the CPU. This improved performance dramatically and allowed the CPU to be busy almost 100 percent of the time.

By the end of the 1960's, something wonderful happened: UNIX was born. It began as a one-man project designed by Ken Thompson of Bell Labs and has grown to become the most widely used operating system. In the time since UNIX was first developed, it has gone through many different generations and even mutations. Some differ substantially from the original version, like BSD (Berkeley Software Distribution) UNIX or Linux. Others, still contain major portions that are based on the original source code. (A friend of mine described UNIX as the only operating system where you can throw the manual onto the keyboard and get a real command.)

Linux is an operating system like many others, such as DOS, VMS, OS/360, or CP/M. It performs many of the same tasks in very similar manners. It is the manager and administrator of all the system resources and facilities. Without it, nothing works. Despite this, most users can go on indefinitely without knowing even which operating system they are using, let alone the basics of how the operating system works.

For example, if you own a car, you don't really need to know the details of the internal combustion engine to understand that this is what makes the car move forward. You don't need to know the principles of hydraulics to understand what isn't happening when pressing the brake pedal has no effect.

An operating system is like that. You can work productively for years without even knowing what operating system you're running on, let alone how it works. Sometimes things go wrong. In many companies, you are given a number to call when problems arise, you report what happened, and it is dealt with.

If the computer is not back up within a few minutes, you get upset and call back, demanding to know when "that darned thing will be up and running again." When the technician (or whoever has to deal with the problem) tries to explain what is happening and what is being done to correct the problem, the response is usually along the lines of, "Well, I need it back up now."

The problem is that many people hear the explanation, but don't understand it. It is common for people to be unwilling to acknowledge that they didn't understand the answer. Instead, they try to deflect the other person's attention away from that fact. Had they understood the explanation, they would be in a better position to understand what the technician is doing and that he/she is actually working on the problem.

By having a working knowledge of the principles of an operating system you are in a better position to understand not only the problems that can arise, but also what steps are necessary to find a solution. There is also the attitude that you have a better relationship with things you

understand. Like in a car, if you see steam pouring out from under the hood, you know that you need to add water. This also applies to the operating system.

In this section, I am going to discuss what goes into an operating system, what it does, how it does it, and how you, the user, are affected by all this.

Because of advances in both hardware design and performance, computers are able to process increasingly larger amounts of information. The speed at which computer transactions occur is often talked about in terms of *billionths* of a second. Because of this speed, today's computers can give the appearance of doing many things simultaneously by actually switching back and forth between each task extremely fast. This is the concept of multitasking. That is, the computer is working on multiple tasks "at the same time."

Another function of the operating system is to keep track of what each program is doing. That is, the operating system needs to keep track of whose program, or task, is currently writing its file to the printer or which program needs to read a certain spot on the hard disk, etc. This is the concept of multi-users, as multiple users have access to the same resources.

In subsequent sections, I will be referring to UNIX as an abstract entity. The concepts we will be discussing are the same for Linux and any other dialect. When necessary, I will specifically reference where Linux differs.

1.2 Processes

One basic concept of an operating system is the process. If we think of the program as the file stored on the hard disk or floppy and the process as that program in memory, we can better understand the difference between a program and a process. Although these two terms are often interchanged or even misused in "casual" conversation, the difference is very important for issues that we talk about later.

A process is more than just a program. Especially in a multi-user, multi-tasking operating system such as UNIX, there is much more to consider. Each program has a set of data that it uses to do what it needs. Often, this data is not part of the program. For example, if you are using a text editor, the file you are editing is not part of the program on disk, but is part of the process in memory. If someone else were to be using the same editor, both of you would be using the same program. However, each of you would have a different process in memory. See the figure below to see how this looks graphically.

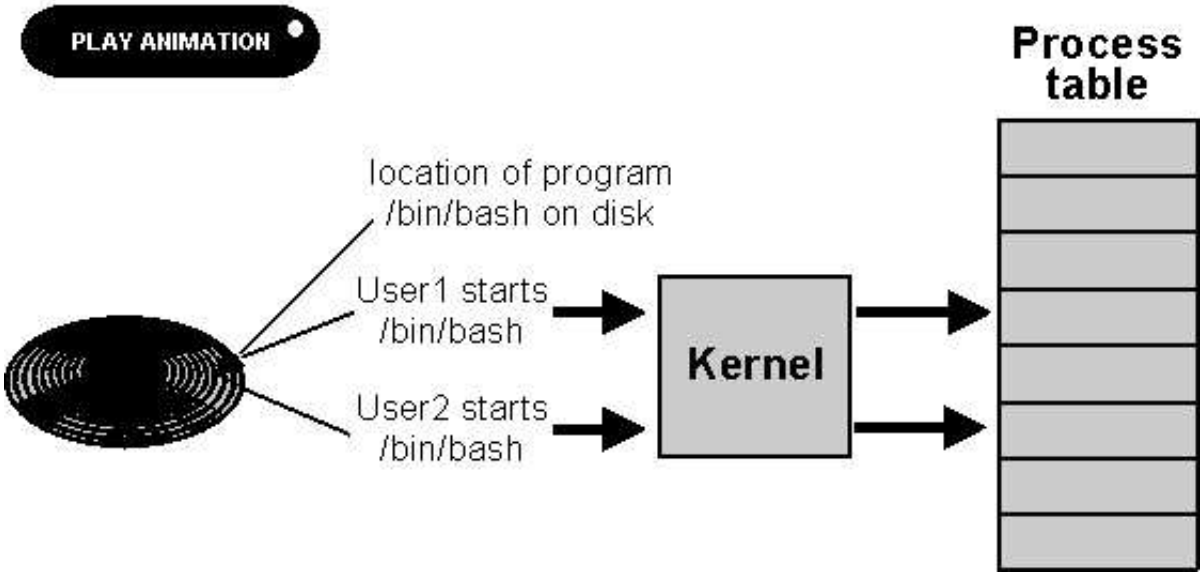


Image - Reading programs from the hard disk to create processes.

Under UNIX, many different users can be on the system at the same time. In other words, they have processes that are in memory all at the same time. The system needs to keep track of what user is running what process, which terminal the process is running on, and what other resources the process has such as open files. All of this is part of the process.

When you log onto a UNIX system, you usually get access to a command line interpreter, or shell. This takes your input and runs programs for you. If you are familiar with DOS, you already have used a command line interpreter: the COMMAND.COM program. Under DOS, your shell gives you the C:> prompt or something similar. Under UNIX, the prompt is usually something like $, #, or %. This shell is a process and it belongs to you. That is, the in-memory or in-core copy of the shell program belongs to you.

If you were to start up an editor, your file would be loaded and you could edit your file. The interesting thing is that the shell has not gone away. It is still in memory. Unlike what operating systems like DOS do with some programs, the shell remains in memory. The editor is simply another process that belongs to you. Because it was started by the shell, the editor is considered a "child" process of the shell. The shell is the *parent* process of the editor. A process has only one parent, but may have many children.

As you continue to edit, you delete words, insert new lines, sort your text and write it out occasionally to the disk. During this time, the backup is continuing. Someone else on the system may be adding figures to a spreadsheet, while a fourth person may be inputting orders into a database. No one seems to notice that there are other people on the system. For them, it appears as though the processor is working for them alone.

Another example we see in the next figure. When you login, you normally have a single process, which is your login shell bash. If you start the X Windowing System, your shell starts another process, xinit. At this point, both your shell and xinit are running, but the shell is waiting for xinit to complete. Once X starts, you may want a terminal in which you can enter commands, so you start **`xterm`**.

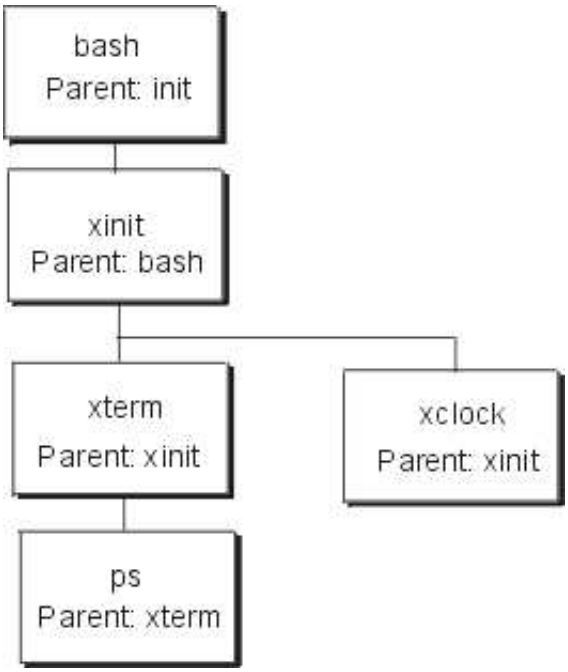


Image - Relationship between parent and child processes.

From the xterm, you might then start the **ps** command, to see what other processes are running. In addition, you might have something like I do, where a clock is automatically started when X starts. At this point, your process tree might look like the figure above.

The nice thing about UNIX is that while the administrator is backing up the system, you could be continuing to edit your file. This is because UNIX knows how to take advantage of the hardware to have more than one process in memory at a time. Note: It is not a good idea to do a backup with people on the system as data may become inconsistent. This was only used as an illustration.

As I write this sentence, the operating system needs to know whether the characters I press are part of the text or commands I want to pass to the editor. Each key that I press needs to be interpreted. Despite the fact that I can clip along at about thirty words per minute, the Central Processing UnitCPU is spending approximately 99 percent of its time doing nothing.

The reason for this is that for a computer, the time between successive keystrokes is an eternity. Let's take my Intel Pentium running at a clock speed of 1.7 GHz as an example. The clock speed of 1.7 GHz means that there are 1.7 billion! clock cycles per second. Because the Pentium gets close to one instruction per clock cycle, this means that within one second, the CPU can get close to executing 1.7 billion instructions! No wonder it is spending most of its time idle. Note: This is an oversimplification of what is going on.

A single computer instruction doesn't really do much. However, being able to do 1.7 billion little things in one second allows the CPU to give the user an impression of being the only one on the system. It is simply switching between the different processes so fast that no one is aware of it.

Each user, that is, each process, gets complete access to the CPU for an incredibly short period of time. On my Linux system, this period of time referred to as a *time slice* is 1/100th of a second. That means that at the end of that 1/100th of a second, it's someone else's turn and the current process is *forced* to give up the CPU. In reality, it is much more complicated than this. We'll get into more details later.

Compare this to an operating system like standard Windows not Windows NT/2000. The program will hang onto the CPU until it decides to give it up. An ill-behaved program can hold onto the CPU forever. This is the cause of a system hanging because nothing, not even the operating system itself, can gain control of the CPU. Linux uses the concept of *pre-emptive* multi-tasking. Here, the system can pre-empt one process or another, to let another have a turn. Older versions of Windows, use *co-operative* multi-tasking. This means the process must be "cooperative" and give up control of the CPU.

Depending on the load of the system how busy it is, a process may get several time slices per second. However, after it has run for its time slice, the operating system checks to see if some other process needs a turn. If so, that process gets to run for a time slice and then its someone else's turn: maybe the first process, maybe a new one.

As your process is running, it will be given full use of the CPU for the entire 1/100th of a second unless one of three things happens. Your process may need to wait for some event. For example, the editor I am using to write this in is waiting for me to type in characters. I said that I type about 30 words per minute, so if we assume an average of six letters per word, that's 180 characters per minute, or three characters per second. That means that on average, a character is pressed once every 1/3 of a second. Because a time slice is 1/100th of a second, more than 30 processes can have a turn on the CPU between each keystroke! Rather than tying everything up, the program waits until the next key is pressed. It puts itself to sleep until it is awoken by some external event, such as the press of a key. Compare this to a "busy loop" where the process keeps checking for a key being pressed.

When I want to write to the disk to save my file, it may appear that it happens instantaneously, but like the "complete-use-of-the-CPU myth," this is only appearance. The system will gather requests to write to or read from the disk and do it in chunks. This is much more efficient than satisfying everyone's request when they ask for it.

Gathering up requests and accessing the disk all at once has another advantage. Often, the data that was just written is needed again, for example, in a database application. If the system wrote everything to the disk immediately, you would have to perform another read to get back that same data. Instead, the system holds that data in a special buffer; in other words, it "caches" that data in the buffer. This is called the *buffer cache*.

If a file is being written to or read from, the system first checks the buffer cache. If on a read it finds what it's looking for in the buffer cache, it has just saved itself a trip to the disk. Because the buffer cache is in memory, it is substantially faster to read from memory than from the disk. Writes are normally written to the buffer cache, which is then written out in larger chunks. If the data being written already exists in the buffer cache, it is overwritten. The flow of things might look like this:

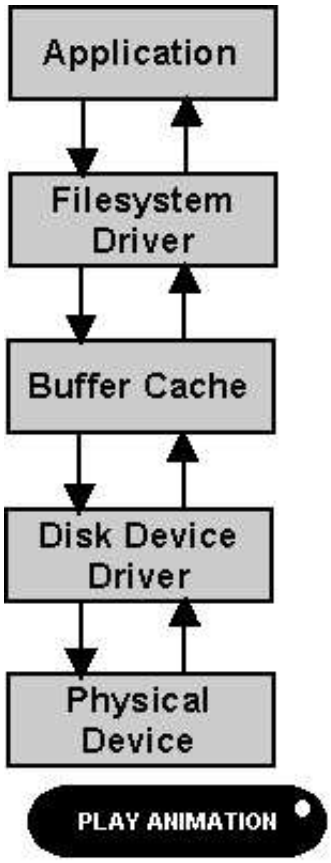


Image - Different layers of file access.

When your process is running and you make a request to read from the hard disk, you typically cannot do anything until you have completed the write to the disk. If you haven't completed your time slice yet, it would be a waste not to let someone else have a turn. That's exactly what the system does. If you decide you need access to some resource that the system cannot immediately give to you, you are "put to sleep" to wait. It is said that you are put to sleep waiting on an event, the event being the disk access. This is the second case in which you may not get your full time on the CPU.

The third way that you might not get your full time slice is also the result of an external event. If a device such as a keyboard, the clock, hard disk, etc. needs to communicate with the operating system, it signals this need through the use of an interrupt. When an interrupt is generated, the CPU itself will stop execution of the process and immediately start executing a routine in the operating system to handle interrupts. Once the operating system has satisfied this interrupt, it returns to its regularly scheduled process. Note: Things are much more complicated than that. The "priority" of both the interrupt and process are factors here. We will go into more detail in the section on the CPU.

As I mentioned earlier, there are certain things that the operating system keeps track of as a process is running. The information the operating system is keeping track of is referred to as the process *context*. This might be the terminal you are running on or what files you have open. The context even includes the internal state of the CPU, that is, what the content of each register is.

What happens when a process's time slice has run out or for some other reason another process gets to run? If things go right and they usually do, eventually that process gets a turn again. However, to do things right, the process must be allowed to return to the exact place where it left off. Any difference could result in disaster.

You may have heard of the classic banking problem concerning deducting from your account. If the process returned to a place *before* it made the deduction, you would deduct twice. If the process hadn't yet made the deduction but started up again at a point after which it would have made the deduction, it appears as though the deduction was made. Good for you, but not so good for the bank. Therefore, everything must be put back the way it was.

The processors used by Linux Intel 80386 and later, as well as the DEC Alpha, and SPARC have built-in capabilities to manage both multiple users and multiple tasks. We will get into the details of this in later chapters. For now, just be aware of the fact that the CPU *assists* the operating system in managing users and processes. This shows how multiple processes might look in memory:

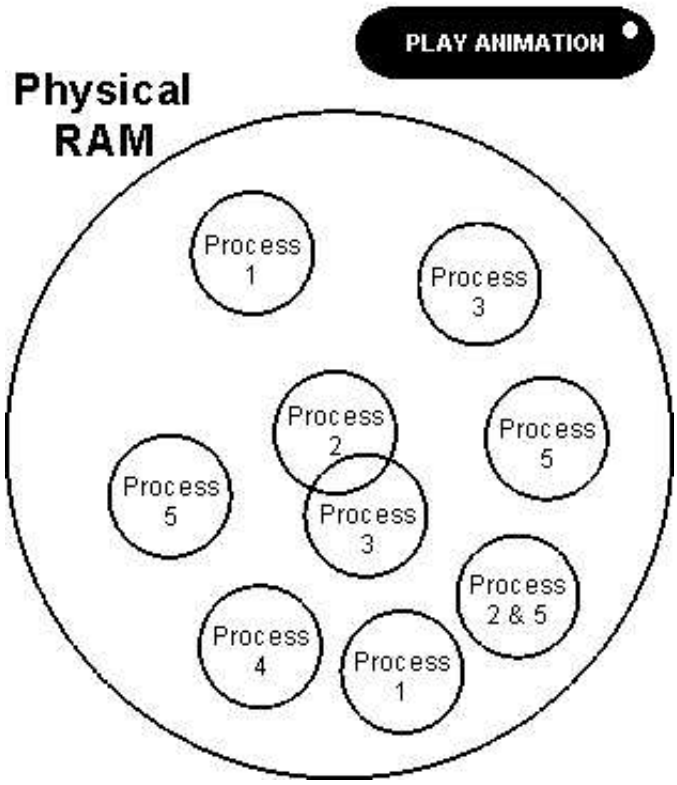


Image - Processes using differing areas of memory.

In addition to user processes, such as shells, text editors, and databases, there are system processes running. These are processes that were started by the system. Several of these deal with managing memory and scheduling turns on the CPU. Others deal with delivering mail, printing, and other tasks that we take for granted. In principle, both of these kinds of processes are identical. However, system processes can run at much higher priorities and therefore run more often than user processes.

Typically a system process of this kind is referred to as a daemon daemon process or background process because they run behind the scenes i.e. in the background without user intervention. It is also possible for a user to put one of his or her processes in the background. This is done by using the ampersand & metacharacter at the end of the command line. I'll talk more about metacharacters in the section on shells .

What normally happens when you enter a command is that the shell will wait for that command to finish before it accepts a new command. By putting a command in the background, the shell does not wait, but rather is ready immediately for the next command. If you wanted, you could put the next command in the background as well.

I have talked to customers who have complained about their systems grinding to a halt after they put dozens of processes in the background. The misconception is that because they didn't see the process running, it must not be taking up any resources. Out of sight, out of mind. The issue here is that even though the process is running in the background and you can't see it, it still behaves like any other process.

1.2.1 Virtual Memory Basics

One interesting aspect about modern operating systems is the fact that they can run programs that require more memory than the system actually has. Like the Tardis in Dr. Who, Linux memory is much bigger on the inside than on the outside.

At the extreme end, this means that if your CPU is 32-bit (meaning that it has registers that are 32-bits), you can access up to 2^{32} bytes (that 4,294,967,296 or 4 **billion**). That means you would need 4 Gb of main memory (RAM) in order to to completely take advantage of this. Although many systems are currently available (2003) with 256 MB or even 512 MB, more RAM than that is rare; and 4 Gb is extremely rare for a home PC.

The interesting thing is that when you sum the memory requirements of the programs you are running,you often reach far beyond the physical memory you have. Currently my system appears to need about 570 Mb. although my machine only has 384 Mb. Surprisingly enough I don't notice any performance problems. So, how is this possible?

Well, Linux, Unix and many other operating systems take advantage of the fact that most programs don't use all of the memory that they "require", as you typically do not use every part of the program at once. For example, you might be using a word processor, but not currently using the spell checking feature, or printing function, so there is no need to keep these in memory at the same time. Also, while you are using your word processor, your email program is probably sitting around doing nothing.

From the user's perspective the email program (or parts of the word processor) are loaded into memory. However, the system only loads what it needs. In some cases, they might all be in memory at once. However, if you load enough programs, you eventually reach a point where you have more programs than you have memory.

To solve this problem, Linux uses something called "virtual memory". It's virtual because it can use more than you actually have. In fact, with virtual memory you can use the whole 2^{32} bytes. Basically, what this means is that you can run more programs at once without the need

for buying more memory.

When a program starts, Linux does not load all of it, just the portion it takes to get started. One aspect of virtual memory is keeping parts of the program that are not needed on the hard disk. As the process runs, when it finds it needs other parts of the program, it goes and gets them. Those parts that are never needed are never loaded and the system does not use all of the memory it appears to "require".

If you have more **data** than physical memory, the system might store it temporarily on the hard disk should it not be needed at the moment. The process of moving data to and from the hard disk like this is called swapping, as the data is "swapped" in and out. Typically, when you install the system, you define a specific partition as the swap partition, or swap "space". However, Linux can also swap to a physical file, although with older Linux versions this is much slower than a special partition. An old rule of thumb is that you have at least as much swap space as you do physical RAM, this ensures that all of the data can be swapped out, if necessary. You will also find that some texts say that you should have at least *twice* as much swap as physical RAM. We go into details on swap in the section in installing and upgrading.

In order to do all this, the system needs to manage your memory. This function is logically called "memory management" and is one of the core aspects of any modern operating system. Although the details are different from one operating system to the next, the basic principles apply, even between different CPU types.

In other sections of the tutorial, we will talk about the details of memory management from both the perspective of the CPU and the operating system kernel.

1.3 Files and Directories

Another key aspect of any operating system is the concept of a file. A file is nothing more than a related set of bytes on disk or other media. These bytes are labeled with a name, which is then used as a means of referring to that set of bytes. In most cases, it is through the name that the operating system is able to track down the file’s exact location on the disk.

There are three kinds of files with which most people are familiar: programs, text files, and data files. However, on a UNIX system, there are other kinds of files. One of the most common is a device file. These are often referred to as *device files* or *device nodes*. Under UNIX, every device is treated as a file. Access is gained to the hardware by the operating system through the device files. These tell the system what specific device driver needs to be used to access the hardware.

Another kind of file is a pipe. Like a real pipe, stuff goes in one end and out the other. Some are named pipes. That is, they have a name and are located permanently on the hard disk. Others are temporary and are unnamed pipes. Although these do not exist once the process using them has ended, they do take up physical space on the hard disk. We’ll talk more about pipes later.

Unlike operating systems like DOS, there is no pattern for file names that is expected or followed. DOS will not even attempt to execute programs that do not end with .EXE, .COM, or .BAT. UNIX, on the other hand, is just as happy to execute a program called program as it is a program called program.txt. In fact, you can use any character in a file name except for "/" and NULL.

However, completely random things can happen if the operating system tries to execute a text file as if it were a binary program. To prevent this, UNIX has two mechanisms to ensure that text does not get randomly executed. The first is the file's permission bits. The permission bits determine who can read, write, and execute a particular file. You can see the permissions of a file by doing a long listing of that file. What the permissions are all about, we get into a little later. The second is that the system must recognize a *magic number* within the program indicating that it is a binary executable. To see what kinds of files the system recognizes, take a look in **`/etc/magic`**. This file contains a list of file types and information that the system uses to determine a file's type.

Even if a file was set to allow you to execute it, the beginning portion of the file must contain the right information to tell the operating system how to start this program. If that information is missing, it will attempt to start it as a shell script similar to a DOS batch file. If the lines in the file do not belong to a shell script and you try to execute the program, you end up with a screen full of errors.

What you name your file is up to you. You are not limited by the eight-letter name and three-letter extension as you are in DOS. You can still use periods as separators, but that's all they are. They do not have the same "special" meaning that they do under DOS. For example, you could have files called

letter.txt
letter.text
letter_txt
letter_to_jim
letter.to.jim

Only the first file example is valid under DOS, but all are valid under Linux. Note that even in older versions of UNIX where you were limited to 14 characters in a file name, all of these are still valid. With Linux, I have been able to create file names that are 255 characters long. However, such long file names are not easy to work with. Note that if you are running either Windows NT or Windows 95, you can create file names that are basically the same as with Linux.

Also keep in mind that although you can create file names with spaces in them, it can cause problems. Spaces are used to seperate the different components on the command line. You can tell your shell to treat a name with spaces as a single unit by including it in quotes. However, you need to be careful. Typically, I simply use an underline _ when the file name ought to have a space. It almost looks the same and I don't run into problems.

One naming convention does have special meaning in Linux: "dot" files. In these files, the first character is a "." dot. If you have such a file, it will by default be invisible to you. That is,

when you do a listing of a directory containing a "dot" file, you won't see it.

However, unlike the DOS/Windows concept of "hidden" files, "dot" files can be seen by simply using the -a all option to **ls**, as in ls -a. ls is a command used to list the contents of directories. With DOS/Windows the "dir" command can show you hidden files and directories, but has no option to show these along with the others.

The ability to group your files together into some kind of organizational structure is very helpful. Instead of having to wade through thousands of files on your hard disk to find the one you want, Linux, along with other operating systems, enables you to group the files into a *directory*. Under Linux, a directory is actually nothing more than a file itself with a special format. It contains the names of the files associated with it and some pointers or other information to tell the system where the data for the file actually reside on the hard disk.

Directories do not actually "contain" the files that are associated with them. Physically that is, how they exist on the disk, directories are just files in a certain format. The directory structure is imposed on them by the program you use, such as ls.

The directories have information that points to where the real files are. In comparison, you might consider a phone book. A phone book does not contain the people listed in it, just their names and telephone numbers. A directory has the same information: the names of files and their numbers. In this case, instead of a telephone number, there is an information node number, or *inode* number.

The logical structure in a telephone book is that names are grouped alphabetically. It is very common for two entries names that appear next to each other in the phone book to be in different parts of the city. Just like names in the phone book, names that are next to each other in a directory may be in distant parts of the hard disk.

As I mentioned, directories are logical groupings of files. In fact, directories are nothing more than files that have a particular structure imposed on them. It is common to say that the directory "contains" those files or the file is "in" a particular directory. In a sense, this is true. The file that is the directory "contains" the name of the file. However, this is the only connection between the directory and file, but we will continue to use this terminology.

One kind of file is a directory. What this kind of file can contain are files and more directories. These, in turn, can contain still more files and directories. The result is a hierarchical tree structure of directories, files, more directories, and more files. Directories that contain other directories are referred to as the *parent* directory of the *child* or subdirectory that they contain. Most references I have seen refer only to parent and subdirectories. Rarely have I seen references to child directories.

When referring to directories under UNIX, there is often either a leading or trailing slash "/", and sometimes both. The top of the directory tree is referred to with a single "/" and is called the "root" directory. Subdirectories are referred to by this slash followed by their name, such as /bin or /dev. As you proceed down the directory tree, each subsequent directory is separated by a slash. The concatenation of slashes and directory names is referred to as a path. Several levels down, you might end up with a path such as **/home/jimmo/letters/personal/chris.txt**, where chris.txt is the actual file and

/home/jimmo/letters/personal is all of the directories leading to that file. The directory **/home** contains the subdirectory jimmo, which contains the subdirectory letters, which contains the subdirectory personal. This directory contains the file chris.txt.

Movement up and down the tree is accomplished by the means of the cd change directory command, which is part of your shell. Although this is often difficult to grasp at first, you are not actually moving anywhere. One of the things that the operating system keeps track of within the context of each process is the process's *current directory*, also referred to as the *current working directory*. This is merely the name of a directory on the system. Your process has no physical contact with this directory; it is just keeping the directory name in memory.

When you change directories, this portion of the process memory is changed to reflect your new "location." You can "move" up and down the tree or make jumps to completely unrelated parts of the directory tree. However, all that really happens is that the current working directory portion of your process gets changed.

Although there can be many files with the same name, each *combination* of directories and file name must be unique. This is because the operating system refers to every file on the system by this unique combination of directories and file name. In the example above, I have a personal letter called chris.txt. I might also have a business letter by the same name. Its path or the combination of directory and file name would be **/home/jimmo/letters/business/chris.txt**. Someone else named John might also have a business letter to Chris. John's path or combination of path and file name might be **/home/john/letters/business/chris.txt**. This might look something like this:

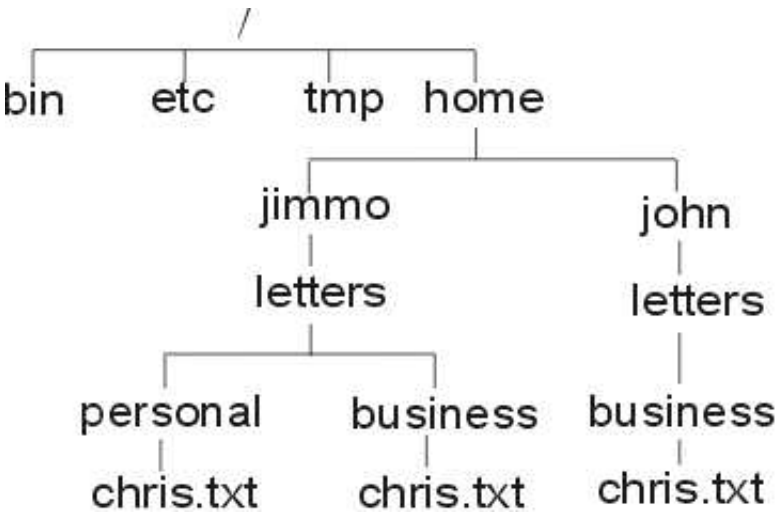


Image - Example of home directories.

One thing to note is that John's business letter to Chris may be the exact same file as Jim's. I am not talking about one being a copy of the other. Rather, I am talking about a situation where both names point to the same physical locations on the hard disk. Because both files are referencing the same bits on the disk, they must therefore be the same file.

This is accomplished through the concept of a *link*. Like a chain link, a file link connects two pieces together. I mentioned above the "telephone number" for a file was its inode. This number actually points to a special place on the disk called the *inode table*, with the inode number being the offset into this table. Each entry in this table not only contains the file's physical location on this disk, but the owner of the file, the access permissions, and the number of links, as well as many other things. In the case where the two files are referencing the same entry in the inode table, these are referred to as *hard links*. A *soft link* or *symbolic link* is where a file is created that contains the *path* of the other file. We will get into the details of this later.

An inode does *not* contain the name of a file. The name is *only* contained within the directory. Therefore, it is possible to have multiple directory entries that have the same inode. Just as there can be multiple entries in the phone book, all with the same phone number. We'll get into a lot more detail about inodes in the section on filesystems. A directory and where the inodes point to on the hard disk might look like this:

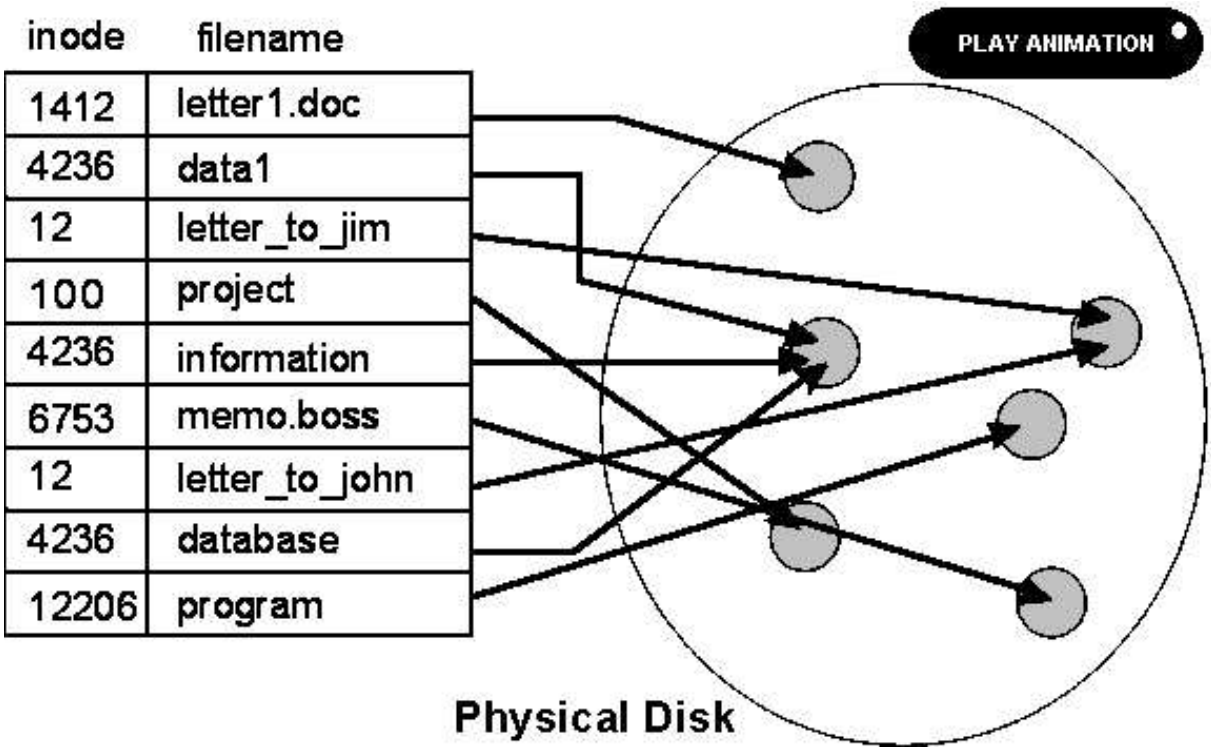


Image - The relationship between file names, inodes and physical data on your hard disk.

Lets think about the telephone book analogy once again. Although it is not common for an individual to have multiple listings, there might be two people with the same number. For example, if you were sharing a house with three of your friends, there might be only one telephone. However, each of you would have an entry in the phone book. I could get the same phone to ring by dialing the telephone number of four different people. I could also get to the same inode with four different file names.

Under Linux, files and directories are grouped into units called *filesystems*. A filesystem is a portion of your hard disk that is administered as a single unit. Filesystems exist within a section of the hard disk called a *partition*. Each hard disk can be broken down into multiple partitions and the filesystem is created within the partition. Each has specific starting and ending points that are managed by the system. Note: Some dialects of UNIX allow multiple filesystems within a partition.

When you create a filesystem under Linux, this is comparable to formatting the partition under DOS. The filesystem structure is laid out and a table is created to tell you where the actual data are located. This table, called the inode table in UNIX, is where almost all the information related to the file is kept.

In an operating system such as Linux, a file is more than just the basic unit of data. Instead, almost everything is either treated as a file or is only accessed through files. For example, to read the contents of a data file, the operating system must access the hard disk. Linux treats the hard disk as if it were a file. It opens it like a file, reads it like a file, and closes it like a file. The same applies to other hardware such as tape drives and printers. Even memory is treated as a file. The files used to access the physical hardware are the device files that I mentioned earlier.

When the operating system wants to access any hardware device, it first opens a file that "points" toward that device the device node. Based on information it finds in the inode, the operating system determines what kind of device it is and can therefore access it in the proper manner. This includes opening, reading, and closing, just like any other file.

If, for example, you are reading a file from the hard disk, not only do you have the file open that you are reading, but the operating system has opened the file that relates to the filesystem within the partition, the partition on the hard disk, and the hard disk itself more about these later. Three additional files are opened every time you log in or start a shell. These are the files that relate to input, output, and error messages.

Normally, when you login, you get to a shell prompt. When you type a command on the keyboard and press enter, a moment later something comes onto your screen. If you made a mistake or the program otherwise encountered an error, there will probably be some message on your screen to that effect. The keyboard where you are typing in your data is the input, referred to as standard input standard in or *stdin* and that is where input comes from by default. The program displays a message on your screen, which is the output, referred to as standard output standard out or *stdout*. Although it appears on that same screen, the error message appears on standard error *stderr*.

Although stdin and stdout appear to be separate physical devices keyboard and monitor, there is only one connection to the system. This is one of those device files I talked about a moment ago. When you log in, the file device is opened for both reading, so you can get data from the keyboard, and writing, so that output can go to the screen and you can see the error messages.

These three concepts standard in, standard out, and standard error may be somewhat difficult to understand at first. At this point, it suffices to understand that these represent input, output, and error messages. We'll get into the details a bit later.

1.4 Operating System Layers

Conceptually, the Linux operating system is similar to an onion. It consists of many layers, one on top of the other. At the very core is the interface with the hardware. The operating system must know how to communicate with the hardware or nothing can get done. This is the most privileged aspect of the operating system.

Because it needs to access the hardware directly, this part of the operating system is the most powerful as well as the most dangerous. What accesses the hardware is a set of functions within the operating system itself the kernel called *device drivers*. If it does not behave correctly, a device driver has the potential of wiping out data on your hard disk or "crashing" your system. Because a device driver needs to be sure that it has properly completed its task such as accurately writing or reading from the hard disk, it cannot quit until it has finished. For this reason, once a driver has started, very little can get it to stop. We'll talk about what can stop it in the section on the kernel.

Above the device driver level is what is commonly thought of when talking about the operating system, the management functions. This is where the decision is made about what gets run and when, what resources are given to what process, and so on.

In our previous discussion on processes, we talked about having several different processes all in memory at the same time. Each gets a turn to run and may or may not get to use up its time slice. It is at this level that the operating system determines who gets to run next when your time slice runs out, what should be done when an interrupt comes in, and where it keeps track of the events on which a sleeping process may be waiting. It's even the alarm clock to wake you up when you're sleeping.

The actual processes that the operating system is managing are at levels above the operating system itself. Generally, the first of these levels is for programs that interact directly with the operating system, such as the various shells. These interpret the commands and pass them along to the operating system for execution. It is from the shell that you usually start application programs such as word processors, databases, or compilers. Because these often rely on other programs that interact directly with the operating system, these are often considered a separate level. This is how the different levels or layers might look like graphically:

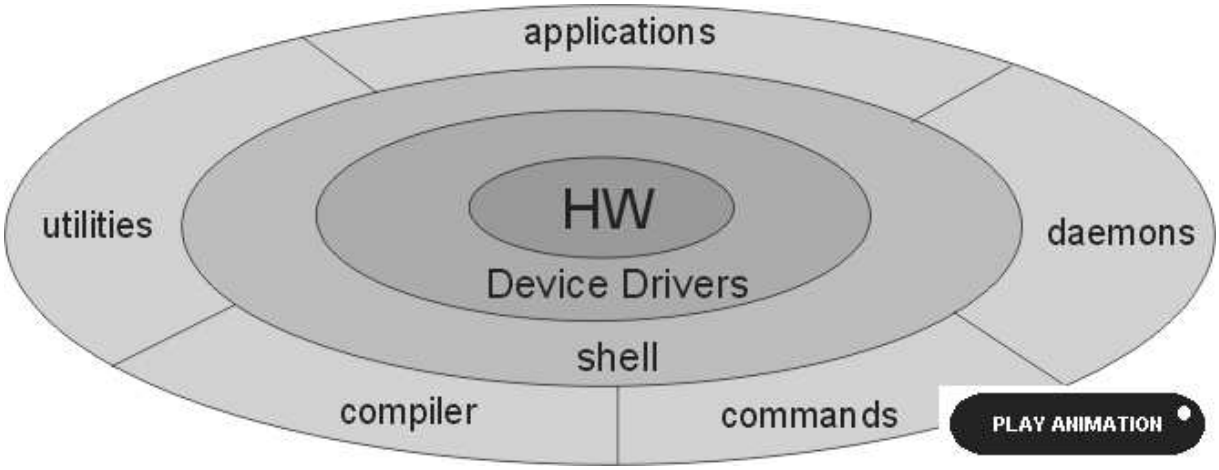


Image - Operating system layers.

If you are running Linux with a graphical interface e.g. the X Windowing System, you have an additional layer. Your shell might start the graphical interface, which then starts the other programs and applications as we discussed.

Under Linux, there are many sets of programs that serve common functions. This includes things like mail or printing. These groups of related programs are referred to as "System Services", whereas individual programs such as vi or fdisk are referred to as utilities. Programs that perform a single function such as ls or date are typically referred to as commands.

1.5 Moving On

So you now have an understanding of the basics of how Linux works. We talked about the different functions that the operating system is responsible for, what it manages, and a little about how everything fits together. As we move on through the book, well build on these ideas and concepts to give you a complete understanding of a Linux system.

I came from the DOS world before I started on UNIX. I had many preconceptions about the way an operating system "should" behave and react. The way DOS did things was the "right" way. As I learned UNIX, I began to see a completely different world. The hardest part was not that I had to learn a whole new set of commands, but rather that I was fighting myself because I was so used to DOS.

Therefore, I need to make one general comment about UNIX before I let you move on. Always remember that UNIX is not DOS. Nor is it any other operating system for that matter. UNIX is UNIX and Linux is Linux. There are probably as many "dialects" of Linux as there are dialects of UNIX. All have their own subtle differences. As you go through this book, keep that in mind.

For example, I believed that the way commands were given arguments or options was better in DOS. Every time I used a UNIX command, I grumbled about how wrong it was to do things like that. As I learned more about UNIX, I came to realize that many of the decisions on how things work or appear is completely arbitrary. There is no right way of doing many

things. There is a DOS way and a UNIX way. Neither is right. You might be used to the DOS way or whatever system you use. However, that does not make it right.

When I started working with Linux, I had several years experience with a half-dozen different dialects of UNIX. It was much easier for me to adjust and simply said to myself, "Oh, so this is the way Linux does it."

If you are new to Linux, keep in mind that there are going to be differences. There are even differences among the various distributions. If you keep this in mind, you will have a much more enjoyable time learning about the Linux way.

I have always found that the best way to learn something is by doing it. That applies to learning a new operating system as well. Therefore, I suggest that when you find something interesting in this book, go look at your Linux system and see what it looks like on your system. Play with it. Twist it. Tweak it. See if it behaves the way in which you expect and understand.

Chapter 2 Linux Basics

With many UNIX systems that are around, the user is unaware that the operating system is a UNIX system. Many companies have point-of-sales systems hooked up to a UNIX host. For example, the users at the cash register may never see what is being run. Therefore, there is really no need to go into details about the system other than for pure curiosity assuming that users find out that they are running on a UNIX system.

On the other hand, if you do have access to the command line or interact with the system by some other means, knowing how the system is put together is useful information. Knowing how things interact helps expand your knowledge. Knowing what's on your system is helpful in figuring out just what your system can do.

That's what this chapter is about: what's out there. We're going to talk about what makes up Linux. This brings up the question "What is Linux?" There are more than a dozen versions commercially available, in several different countries, all with their own unique characteristics. How can you call any one of them *the* Linux distribution?

The answer is you can't. What I will do instead is to synthesize all the different versions into a single pseudo-version that we can talk about. Although there are differences in the different versions, the majority of the components are the same. There has been a great deal of effort in the past few years to standardize Linux, with a great deal of success. I will therefore address this standard Linux and then mention those areas where specific versions diverge.

2.1 What Linux is All About

Linux is available from many companies and in many versions. Often, a company will produce its own version with specific enhancements or changes. These are then released commercially and called distributions. Although Linux is technically only the kernel, it is commonly considered to be all of the associated programs and utilities. Combined with the kernel, the utilities and often some applications comprise a commercial distribution.

2.1.1 Guided Tour

Unless you are on familiar ground, you usually need a map to get around any large area. To get from one place to another, the best map is a road map or street map. If you are staying in one general area and are looking for places of interest, you need a tourist map. Because we are staying within the context of Linux and were looking for things of interest, what I am going to give you now is a tourist map of Linux directories.

In later chapters, we'll go into detail about many of the directories that we are going to encounter here. For now, I am going to briefly describe where they are and what their functions are. As we get into different sections of the book, it will be a lot easier to move about and know how files relate if we already have an understanding of the basic directory structure.

One thing I would like to point out is that for the most part the directories of most UNIX systems are laid out according to the functionality of the files and programs within the directory. One enhancement that Linux makes is allowing things to be in more than one place. For example, files that the system uses may be in one place and those that normal users need may be in another place. Linux takes advantage of links to allow the necessary files to be in both places. We'll talk more about links as we move on.

One question people often ask is why it is necessary to know what *all* the directories are for. Well, it isn't. It isn't necessary to know them all, just the more important ones. While working in tech support, I have talked numerous times with administrators who were trying to clean up their systems a little. Because they had little experience with UNIX systems, they ended up removing things that they thought were unnecessary, but turned out to be vital for the operation of the system. If they knew more about where things were and what they were for, they wouldn't have made these mistakes.

As we go through these directories, keep in mind that your system may not be like this. I have tried to follow the structure of the Linux Filesystem Standard as well as to find some commonality among the different versions that I've installed. On your system, the files and directories may be in a different place, have different names, or may be gone altogether.

Note that depending on your distribution and the packages you have installed, these files and directories will look different. In addition, although my system has every conceivable package installed well, almost, I did not list all the files and directories I have. I included this list with the intention of giving you a representative overview. In addition, some of the directories are not mentioned in the text, as I cannot say too much more than in the popup in the image map in so little space.

With that said, let's have a look.

The top-most directory is the root directory. In verbal conversation, you say "root directory" or "slash," whereas it may be referred to in text as simply "/."

So when you hear someone talking about the /bin directory, you may hear them say "slash bin." This is also extended to other directories, so /usr/bin would be "slash user, slash bin." However, once you get the feeling and begin to talk "Linux-ese," you will start talking about the directories as "bin" or "user bin." Note that usr is read as "user."

Under the root, there are several subdirectories with a wide range of functions. The image below shows the key subdirectories of /. This representation does not depict every subdirectory of /, just the more significant ones that appear with most default installations. In subsequent diagrams, I will continue to limit myself to the most significant directories to keep from losing perspective.

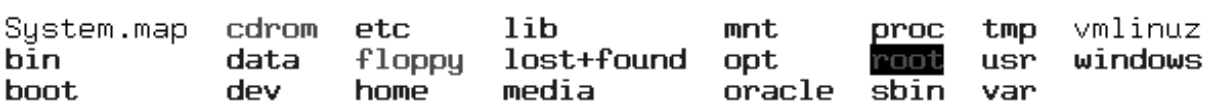


Image - Listing of a typical root directory.

One of these files, one could say, is the single *most* important file: vmlinuz. This file is the operating system proper. It contains all the functions that make everything go. When referring to the file on the hard disk, one refers to /vmlinuz, whereas the in-memory, executing version is referred to as the *kernel*.

The first directory we get to is **/bin**. Its name is derived from the word "**bin**ary." Often, the word "binary" is used to refer to executable programs or other files that contains non-readable characters. The **/bin** directory is where many of the system-related binaries are kept, hence the name. Although several of the files in this directory are used for administrative purposes and cannot be run by normal users, everyone has read permission on this directory, so you can at least see what the directory contains.

The **/boot** directory is used to boot the system. There are several files here that the system uses at different times during the boot process. For example, the files /boot/boot.???? are copies of the original boot sector from your hard disk. for example boot.0300 Files ending in .b are "chain loaders," secondary loaders that the system uses to boot the various operating systems that you specify.

The **/dev** directory contains the device nodes. As I mentioned in our previous discussion on operating system basics, device files are the way both the operating system and users gain access to the hardware. Every device has at least one device file associated with it. If it doesn't, you can't gain access to it. We'll get into more detail on individual device files later.

The **/etc** directory contains files and programs that are used for system configuration. Its name comes from the common abbreviation *etc.,* for et cetera, meaning "and so on." This seems to come from the fact that on many systems, /etc contains files that don't seem to fit elsewhere.

Under **/etc** are several subdirectories of varying importance to both administrators and users. The following image shows a number of important sub-directories. Depending on what software you have installed you may not have some of these or you may have many more not listed.

```
SuSEconfig  cron.daily    cron.weekly  httpd   pam.d        samba     ssh
X11         cron.hourly   cups         init.d  profile.d    security  vmware
cron.d      cron.monthly  default      opt     rc.config.d  skel
```

Image - Listing of a key directories under the /etc directory.

In some Linux distributions you will find the **/etc/lilo** directory, which is used for the Linux loader **lilo**. This directory contains a single file, install, which is a link to /sbin/lilo. This file is used among other things to install the boot configuration options. On some systems, the lilo configuration file lilo.conf is found directly in the **/etc** directory We'll get into this more in the section on starting and stopping your system.

There several directories named /etc/cron*. As you might quess these are used by the cron daemon. The /etc/cron.d contains configuration files used by cron. Typically what is here are various system related cron jobs, such as /etc/cron.d/seccheck, which does various security checks. The directories **/etc/cron.hourly**, **/etc/cron.daily**, **/etc/cron.weekly**, **/etc/cron.monthly** contain files with cron jobs which run

hourly, daily, weekly and monthly, respectively. There is a cron job listed in **/etc/crontab** that runs the program **/usr/lib/cron/run-crons**, which checks the other files.

The /etc/init.d directory contains scripts that the system uses when starting up or shutting down. Which files are read depends on whether the system is being started or shut down. We'll talk more about these directories and their associated files in the section on starting up and shutting down the system. You may also find that these files are located in **/etc/rc.d**. On SuSE, **/etc/rc.d** is a symbolic link to **/etc/init.d**.

The **/etc/skel** directory is used when you create a new user with the adduser command. This is the "skeleton" of files that is copied to the user's home directory when it's created hence the name "skel". If you want to ensure that each user gets other files at startup, place them in here. For example, you may want everyone to have a configuration file for **vi** .exrc or for mail .mailrc.

Depending on your Linux distribution, either the /etc/sysconfig or **/etc/rc.config.d** directory contains default system configuration information. For example, the keyboard file defines which keyboard table is to be used and the network file contains network parameters, such as the hostname.

The **/etc/pam.d** directory contains configuration files used by the Pluggable Authentication Modules PAM. PAM is a system of libraries that are responsible for authentication tasks of applications and services on your system. These libraries provide an Application Programming Interface API allowing for a standardization of authorization functions. Previously, where necessary each program did its own authorization/authentication. With PAM, a single set of configuration files allows for a more consistent security policy. In some cases, an **/etc/pam.conf** file is used instead of the **/etc/pam.d** directory.

The **/etc/profile.d** directory contains default configuration for many of the shells that Linux provides. As we talk about in the section on shells, each shell has an environment which contains a number of different characteristics. Many of the defaults are defined in the files under /etc/profile.d. The name of each file gives an indication of the appropriate shell.

The **/etc/security** directory contains security related configurations files. Whereas PAM concerns itself with the methods used to authenticate any given user, the files under /etc/security are concerned with just what a user can or cannot do. For example, the file **/etc/security/access.conf** is a list of what users are allowed to login and from what host for example, using telnet. The **/etc/security/limits.conf** contains various system limits, such as maximum number of processes. Yes, these are really related to security!

Moving back up to the root directory, we next find **/home**. As its name implies, this is the default location for user's home directories. However, as we'll talk about later, you can have the home directory anywhere.

The **/lost+found** directory is used to store files that are no longer associated with a directory. These are files that have no home and are, therefore, lost. Often, if your system crashes and the filesystem is cleaned when it reboots, the system can save much of the data

and the files will end up here. Note that a lost+found directory is created automatically for each filesystem you create. We'll get into more detail about this in the section on filesystems.

The **/lib** directory for library contains the libraries needed by the operating system as it is running. You will also find several sub directories.

The **/proc** directory takes a little while to get used to, especially if you come from a non-UNIX world or have used a version of UNIX without this directory. This is a "pseudo-filesystem" that is used to access information in the running system. Rather than having you access kernel memory directly i.e., through the special device /dev/kmem, you can access the files within this directory. There are directories for every running process as well. We will get into more detail about this when we talk about monitoring your system. If you are curious now, check out the **proc**8 man-page.

The **/root** directory is the home directory for the user root. This is different from many UNIX dialects that have the root's home directory in /. On SuSE, the /root directory is actually a symbolic link to /home/root.

The **/sbin** directory contains programs that are used more or less to administer the system. In other words, the **s**ystem **bin**aries. Many documentation sources say that this is *only* for system administrators. However, most of these files are executable by normal users, as well. Whether the support files or device nodes are accessible is another matter. If a normal user cannot access the device nodes or other files, the program won't run.

The **/usr** directory contains many user-related subdirectories. Note the 'e' is missing from "user". In general, one can say that the directories and files under /usr are used by and related to users. There are programs and utilities here that users use on a daily basis. Unless changed on some systems, /usr is where users have their home directory. The figure below shows what the subdirectories of /usr would look like graphically.

```
X11    bin   doc  i486-linux       include  local       man   share  src
X11R6  dict  etc  i486-suse-linux  lib      lost+found  sbin  spool  tmp
```

Image - Listing of a key directories under the /usr directory.

Where **/bin** contains programs that are used by both users and administrators, /usr/bin contains files that are almost exclusively used by users. However, like everything in UNIX, there are exceptions. Here again, the bin directory contains binary files.

The **/usr/adm** directory contains mostly administrative data. The name "adm" comes from "administration," which is no wonder considering this contains a lot of the administrative information that relates to users. This may be a symbolic link to the **/var** directory.

The **/usr/include** directory and its various subdirectories contain all the include files. These contain information that is needed both by the kernel when it is being recreated and by programs when they are being compiled. For normal users and even most system administrators, the information here is more a place to get one's curiosity satisfied. For those of you who know that this is dramatic over-simplification, all I can say is that you already know what this directory is for anyway.

The **/usr/src** directory contains the source code for both the Linux kernel and for any program that you specifically install.

Many system parameters and values are stored inside the files underneath **/usr/src/linux/include**. Because of the information provided in many of the files, I will be making reference to them through the book. Rather than spelling out the full path of the directory, I will make a reference to the files relative to the **/usr/src/linux/include** directory, the same way that it is done in C source code. For example, when I refer to something like <linux/user.h>, I mean the full path **/usr/src/linux/include/linux/user.h**. When you see something enclosed in angled brackets like this, you can make the expansion yourself.

The **/usr/lib** directory is difficult to explain. We could say that it contains the user-related library files based on its name. However, that still does not accurately describe the complete contents. One thing it contains is the library files that are less general than those you find in **/lib**. This directory contains many of the systemwide configuration files for user-level programs such as perl and emacs.

The **/usr/lib/kbd** directory contains files that are used to configure the system console keyboard. Through these files, you can configure your keyboard to accommodate one of several different languages. You can even configure it for dialects of the same language, such as the German keyboard as used in Switzerland or Germany. You can also change these files to create a totally new keyboard layout, such as the Dvorak.

If you have switched to the more secure npasswd program, the **/usr/lib/npasswd** directory is used to contain some configuration information.

The **/usr/lib/terminfo** directory contains both the source files and compiled versions of the terminfo database. Terminfo is the mechanism by which the system can work with so many different types of terminals and know which key is being pressed. For more information, see the terminfo5 man-page.

When configuring UUCP, all the necessary files are contained in the **/usr/lib/uucp** directory. Not only are the configuration files here, but this is also home for most of the UUCP programs. UUCP Unix-to-Unix Copy is a package that allows you to transfer files and communicate with remote systems using serial lines. We'll talk in more detail about this directory in the section on networking.

There are typically many more directories under **/usr/lib**. Most are related to user programs and operations. We'll get to some of them as we move along.

The directory **/usr/X11R6** contains all the X Windows System files. This makes upgrading to newer releases of X much easier as the files are not spread out over the entire system. If you have an older version of Linux, you might still have X11R5 or if a newer release comes out you might have X11R7. To simplify things even further, the directory **/usr/X11** is what many things look at instead. This is then linked to the appropriate directory i.e., **/usr/X11R6**, **/usr/X11R5**.

Underneath this directory are the subdirectories bin, lib, and man, which have the same functionality as those under /usr. In most cases, links in other directories point here. For example, you should have a directory **/usr/bin/X11**. This is a symbolic link to the directory **/usr/X11R6/bin**. The directory **/usr/lib/X11** is a symbolic link to **/usr/X11R6/lib**. The reason for this is to maintain the directory structure, but still make upgrading easy. When X11R7 comes out, all that you need to do is make the links point to the X11R7 directories and not copy the individual files.

Next, **/usr/sbin** contains more system binaries, including the daemon programs that run in the background. In some UNIX dialects, these files may be in **/etc**.

Moving back up to the **/usr** directory, we find the **/usr/local** sub-directory. This may or may not contain anything. In fact, there are no rules governing its contents. It is designed to contain programs, data files, and other information that is specific to your local system, hence the name. There is often a bin directory that contains local programs and a lib directory that contains data files or libraries used by the programs in **/usr/local/bin**.

Also in the **/usr** directory is **/usr/man**. This is where the man-pages and their respective indices are kept. This directory contains the index files, which you can search through to find a command you are looking for. You can also create and store your own manual pages here. The **/usr/info** and **/usr/doc** directories contain GNU Info documents and other documentation files.

The **/usr/spool** directory is the place where many different kinds of files are stored temporarily. The word "spool" is an acronym for **s**imultaneous **p**eripheral **o**peration **o**ff-**l**ine, the process whereby jobs destined for some peripheral printer, modem, etc. are queued to be processed later. This may be a link to /var/spool.

Several subdirectories are used as holding areas for the applicable programs. For example, the **/usr/spool/cron** directory contains the data files used by cron and at. The **/usr/spool/lp** directory not only contains print jobs as they are waiting to be printed, it also contains the configuration files for the printers.

```
X11R6  cache    games  lock  mail   opt  spool  tmp       yp
adm    deliver  lib    log   named  run  squid  ucd-snmp
```

The **/var** directory contains files that **va**r**y** as the system is running, such as log files. This was originally intended to be used when the /usr directory is shared across multiple systems. In such a case, you don't want things like the mail or print spoolers to be shared.

The **/var/man/cat** directory is a cache for man-pages when they are formatted. Some are stored in a pre-formatted form, and those that need to be formatted are cached here in case they are needed again soon.

Many system lock files are kept in **/var/lock**. These are used to indicate that one program or another is currently using a particular file or maybe even a device. If other programs are written to check in here first, you don't have collisions.

As you might guess, the **/var/log** directory contains log files. The **/var/run** contains information that is valid until the system is rebooted. For example, the process ID of the inetd daemon can be found here. It is often important to know this information when changes are made to the system and storing them here makes them quickly accessible.

The **/var/yp** directory contains the changing files that are used with the Network Information Service NIS, also know as Yellow Pages, or YP.

As I mentioned before, the **/usr/adm** directory is a link to **/var/adm**. There are several key log files stored here. Perhaps, the most important is the messages file that contains all the system service, kernel, and device driver messages. This is where the system logs messages from the syslogd daemon.

There were many directories that I skipped, as I said I would at the beginning of this section. Think about the comparison that I made to a tourist map. We visited all the museums, 200-year-old churches, and fancy restaurants, but I didn't show you where the office of city planning was. Granted, such offices are necessary for a large city, but you really don't care about them when you're touring the city; just as there are certain directories and files that are not necessary to appreciate and understand the Linux directory structure.

2.1.2 What Linux is Made of

There are many aspects of the Linux operating system that are difficult to define. We can refer to individual programs as either utilities or commands, depending on the extent of their functions. However, it is difficult to label collections of files. Often, the labels we try to place on these collections do not accurately describe the relationship of the files. However, I am going to try.

Linux comes with essentially all the basic UNIX commands and utilities that you have grown to know and love (plus some that you don't love so much). Basic commands like **ls** and **cat**, as well as text manipulation programs like **sed** and **awk** are available. If you don't come from a Unix background, then many of the commands may seem a little obscure and even intimidating. However, as you learn more about them you will see how useful and powerful they can be, even if it takes longer to learn them.

Linux also comes with a wide range of programming tools and environments, including the GNU **gcc** compiler, **make**, **rcs**, and even a debugger. Several languages are available, including Perl, Python, Fortran, Pascal, ADA, and even Modula-3.

Unless you have an extremely low-level distribution, you probably have X11R6 in the form of XFree86 3.x, which contains drivers for a wide range of video cards. There are a dozen text editors (**vi**, **emacs**, **jove**) and shells (**bash**, **zsh**, **ash**, **pdksh**), plus a wide range of text processing tools, like TeX and groff. If you are on a network, there is also a wide range of networking tools and programs.

Even if you have been working with a Linux or any UNIX dialect for a while, you may have heard of certain aspects of the operating system but not fully understood what they do. In this section, I'm going to talk about functions that the system performs as well as some of the programs and files that are associated with these functions. I'm also going to talk about how

many of the system files are grouped together into what are referred to as "packages," and discuss some of the more important packages.

To install, remove, and administer these packages on a *Slackware*-derived system, use the **pkgtool** tool, which is actually a link to the shell script **cpkgtool**. This tool can be called from the command line directly or by the /sbin/setup program. Each package comes on its own set of disks. These packages are:

- A Base Linux System
- AP various applications that do not need X
- D Program Development (C, C++, Lisp, Perl, etc.)
- E GNU emacs
- F FAQ lists, HOWTO documentation
- I Info files readable with info, JED, or emacs
- IV InterViews Development + Doc and Idraw apps for X
- N Networking (TCP/IP, UUCP, Mail, News)
- OOP Object-Oriented Programming (GNU Smalltalk 1.1.1)
- Q Extra Linux kernels with custom drivers
- T TeX ,text processing system
- TCL Tcl/Tk/TclX, Tcl language and Tk toolkit for X
- X XFree-86 3.1 X Window System
- XAP X applications
- XD XFree-86 3.1 X11 Server Development System
- XV XView 3.2 (OpenLook Window Manager, apps)
- Y games (that do not require X)

Why is it important to know the names of the different packages? Well, for the average user, it really isn't. However, the average user logs on, starts an application and has very little or no understanding of what lies under the application. The mere fact that you are reading this says to me that you want to know more about the operating system and how things work. Because these packages are the building blocks of the operating system (at least in terms of how it exists on the hard disk), knowing about them is an important part of understanding the whole system.

Plus one of the key advantages that Linux has over Windows is the ability to selectively install and remove packages with *much* finer granularity. For example, you can add and remove individual programs to a greater extent with Linux than you can with Windows. Further there are fewer groups of programs in Windows (such a group of programs is often called a "package" in Linux. This allows you to pick and chose what you want to install to a greater extent. Therefore, knowing where each package resides (or at least having a starting point) is a big a help.

To be able to do any work on a Linux system, you must first install software. Most people think of installing software as adding a word processing program or database application; but any program on the operating system needs to be installed at one time or another. Even the operating system itself was installed.

Earlier, I referred to the Linux operating system as all the files and programs on the hard disk. For the moment, I want to restrict the definition of "operating system" to just those files that are necessary for "normal" operation. Linux (at least Slackware) has defined that set of programs and files as the Base Linux System, or Base Package. Although there are many files in the Base Package that could be left out to have a running system, this is the base set that is usually installed.

Many versions of Linux are now using the Red Hat Package Manager (RPM) format. In fact, RPM is perhaps the format most commonly found on the Internet. Most sites will have new or updated programs as RPM files. You can identify this format by the rpm extension to the file name.

This has proven itself to be a much more robust mechanism for adding and removing packages, as it is much easier to add and manage single programs than with Slackware. We'll get into more detail about this when I talk about installing. You will also find that RPM packages are also grouped into larger sets like those in Slackware, so the concepts are the same.

Although most commercial distributions use the RPM format, there are often a number of differences in which package groups there are and which programs and applications appear in which group. For example, the SuSE 7.3 distribution has the following package group:

- a1 - Linux Base System (required)
- ap1 - Applications which do not need X
- ap4 - Applications which do not need X
- d1 - Development (C, C++, Lisp, etc.)
- doc1 - Documentation
- doc4 - Documentation
- e1 - Emacs
- fun1 - Games and more
- gra1 - All about graphics
- gra3 - All about graphics
- k2de1 - KDE2 - K Desktop Environment (Version 2)
- n1 - Network-Support (TCP/IP, UUCP, Mail, News)
- perl1 - Perl modules
- sec1 - Security related software
- snd1 - Sound related software
- spl1 - Spell checking utilities and databases
- tcl1 - Tcl/Tk/TclX, Tcl-Language and Tk-Toolkit for X
- tex1 - TeX/LaTeX and applications
- x1 - Base X Window System - XFree86\tm
- x3d1 - 3D software for X11 and console
- xap1 - X Applications
- xdev1 - Development under X11
- xsrv1 - Several X Servers (XFree86)
- xsrv2 - Several X Servers (XFree86)

- xsrv3 - Several X Servers (XFree86)
- xsrv4 - Several X Servers (XFree86)
- xv1 - XView (OpenLook, Applications)
- xwm1 - Window managers and desktop environments
- yast1 - YaST Components
- zq - source packages

Note that in the case of SuSE, when you are in the administration tool (YAST), the names of these groups will probably appear somewhat different. For example, there are two groups of applications: those that need X-Windows and those that do not. When you are in YAST, there are two dozen application groups, such as spreadsheets, math and databases. The groups listed above are how you might find them on the CD and date from a time when you did not have many applications and there were few distributions. Most people got Linux from the net and these package groups were pretty convenient. Today, SuSE 7.3 is on seven CDs and just to make things easier, you are also given a single DVD.

2.1.3 What Linux Does

On any operating system, a core set of tasks is performed. On multi-user or server systems such as Linux, these tasks include adding and configuring printers, adding and administering users, and adding new hardware to the system. Each of these tasks could take up an entire chapter in this book. In fact, I do cover all of these, and many others, in a fair bit of detail later on.

I think it's important to briefly cover all of the basic tasks that an administrator needs to perform in one place. There are a couple of reasons for this. First, many administrators of Linux systems are not only novice administrators, they are novice users. They get into the position as they are the only ones in the company or department with computer experience. (They've worked with DOS before.) Second, by introducing the varied aspects of system administration here, I hope to lay the foundation for later chapters. If you are not familar with this issue, you may have trouble later.

Keep in mind that depending on what packages are installed, any Linux distribution can do a lot more. Here we will be discussing just the basic administrative functions.

The average user may not want to get into the details that the later chapters provide. So here I give an overview of the more important components. Hopefully, this will give you a better understanding of what goes into an operating system as well as just how complex the job is that your system administrator does.

The first job of a system administrator is to add users to the system. Access is gained to the system only through user accounts. Although it may be all that a normal user is aware of, these accounts consist of substantially more than just a name and password. Each user must also be assigned one of the shells, a home directory, and a set of privileges to access system resources.

Although the system administrator could create a single user account for all users to use to log in, it ends up creating more problems than it solves. Each user has his/her own password and home directory. If there were a single user, everyone's files would be stored in the same place and everyone would have access to everyone else's data. This may be fine in certain circumstances, but not in most.

Users are normally added to the system through the adduser command. Here, when adding a user, you can input that user's default shell, his/her home directory as well as his/her access privileges.

Another very common function is the addition and configuration of system printers. This includes determining what physical connection the printer has to the system, what characteristics the printer has (to choose the appropriate model printer) as well as making the printer available for printing. Generically, all the files and programs that are used to access and manage printers are called the print spool, although not all of them are in the spool directory.

Adding a printer is accomplished like in many UNIX dialects: you do it manually with the primary configuration file, **`/etc/printcap`** file. The printcap man-page lists all the capabilities that your version of Linux supports. You must also add the appropriate directory and enable printing on the port. We'll get into more detail about it as we move on.

What happens when you want to remove a file and inadvertently end up removing the wrong one (or maybe more than one)? If you are like me with my first computer, you're in big trouble. The files are gone, never to show up again. I learned the hard way about the need to do backups. If you have a good system administrator, he/she has probably already learned the lesson and makes regular backups of your system.

There are several ways of making backups and several different utilities for doing them. Which program to use and how often to make backups completely depends on the circumstances. The system administrator needs to take into account things like how much data needs to be backed up, how often the data are changed, how much can be lost, and even how much will fit on the backup media.

There are tasks that an administrator may need to perform at regular intervals, such as backups, cleaning up temporary directories, or calling up remote sites to check for incoming mail. The system administrator could have a checklist of these things and a timer that goes off once a day or every hour to remind him/her of these chores, which he/she then executes manually.

Fortunately, performing regular tasks can be automated. One basic utility in every UNIX version is cron. Cron (the "o" is short) is a program that sits in the background and waits for specific times. When these times are reached, it starts pre-defined programs to accomplish various, arbitrarily defined tasks. These tasks can be set to run at intervals ranging from once a minute to once a year, depending on the needs of the system administrator.

Cron "jobs" (as they are called) are grouped together into files, called *cron tables*, or *crontabs* for short. There are several that are created by default on your system and many users and even system administrators can go quite a long time before they notice them. These monitor

certain aspects of system activity, clean up temporary files, and even check to see if you have UUCP jobs that need to be sent.

What about a program that you only want to run one time at a specific time and then never again? Linux provides a mechanism: at. Like cron, at will run a job at a specific time, but once it has completed, the job is never run again.

A third command that relates to cron and at, the batch command, differs from the other two in that batch runs the job you submit whenever it has time; that is, when the system load permits.

Linux supports the idea of virtual consoles (VCs), like SCO. With this, the system console (the keyboard and monitor attached to the computer itself) can work like multiple terminals. By default, the system is configured with at least four VCs that you switch between by pressing the ALT key and one of the function keys F1-F6.

Normally, you will only find the first six VCs active. Also, if you are using the X Windowing System, it normally starts up on VC 7. To switch from the X-Windows screen to one of the virtual consoles, you need to press CTRL-ALT plus the appropriate function key.

Keeping the data on your system safe is another important task for the system administrator. Linux provides a couple of useful tools for this: tar and cpio. Each has its own advantages and disadvantages. Check out the details on the respective man-page.

2.1.4 What goes with Linux

Throughout this site, we are going to be talking a great deal about what makes up the Linux operating system. In its earliest form, Linux consisted of the base operating system and many of the tools that were provided on a standard UNIX system. For many companies or businesses, that was enough. These companies may have only required a single computer with several serial terminals attached, running a word processor, database, or other application. However, when a single computer is not enough, the base Linux package does not provide you with everything that you need.

Suppose you want to be able to connect all the computers in your company into a computer network. The first thing that you could use is the networking capabilities of UUCP, which is included in Linux's network package. However, this is limited to exchanging files, remotely executing programs, and simple terminal emulation. Also, it is limited to serial lines and the speed at which data can be transferred is limited as well.

So it was in the dark recesses of ancient computer history. Today, products exist that allow simultaneous connection between multiple machines with substantially higher performance. One such product is TCP/IP (Transmission Control Protocol/Internet Protocol). If a company decides it needs an efficient network, it might decide to install TCP/IP, which has become the industry standard for connecting not only UNIX systems, but other systems as well.

There is a problem with TCP/IP that many companies run into. Suppose you want everyone in the company to be able to access a specific set of files. With TCP/IP you could devise a scheme that copies the files from a central machine to the others. However, if the files need to be changed, you need to ensure that the updated files are copied back to your source machine.

This is not only prone to errors, but it is also inefficient.

Why not have a single location where the source files themselves can be edited? That way, changes made to a file are immediately available to everyone. The problem is that TCP/IP by itself has nothing built in to allow you to share files. You need a way to make a directory (or set of directories) on a remote machine *appear* as though it were local to your machine.

Like many operating systems, Linux provides an answer: NFS (Network File System). With NFS, directories or even entire filesystems can appear as if they are local. One central computer can have the files physically on its hard disk and make them available via NFS to the rest of the network.

Two other products are worth mentioning. To incorporate the wonders of a graphical user interface (GUI), you have a solution in the form of X-Windows. And if you just switched to Linux and still have quite a few DOS applications that you can't live without, Linux provides a solution: dosemu or the DOS Emulator package.

2.2 Linux Documentation

Software documentation is a very hot subject. It continues to be debated in all sorts of forums from USENET newsgroups to user groups. Unless the product is very intuitive, improperly documented software can be almost worthless to use. Even if intuitive to use, many functions remain hidden unless you have decent documentation. Unfortunately for many, UNIX is not very intuitive. Therefore, good documentation is essential to be able to use Linux to its fullest extent.

Unlike a commercial UNIX implementation, Linux does not provide you with a bound set of manuals that you can refer to. The documentation that is available is found in a large number of documents usually provided with your Linux distribution. Because the documentation was developed by many different people at many different locations, there is no single entity that manages it all.

The Linux Documentation Project LDP was organized for this very reason. More and more documents are being produced as Linux develops. There are many HOWTOs available that give step-by-step instructions to perform various tasks. These are typically quite long, but go into the detail necessary to not only solve specific problems, but help you configure detailed aspects of your system. There are also a number of "mini" HOWTOs, which discuss less extensive topics.

In many cases, these were written by the program developers themselves, giving you insights into the software that you normally wouldn't get. You'll find ASCII versions on the CD under the doc/HOWTO directory and HTML versions under doc/HTML. The most current HOWTOs can be found on the LDP Web site.

Many HOWTOs will have a section of frequently asked questions FAQs. As their name implies, these are lists of questions that are most frequently asked about the particular topic. Sometimes these are questions about specific error messages, but are often questions about implementing certain features. These can also be found on the LDP Web site. The Brief Linux

FAQ BLFAQ provides answers to basic questions about working with Linux.

Unfortunately, in my experience in tech support, few administrators and even fewer users take the time to read the manuals. This is not good for two important reasons. The first is obviously the wasted time spent calling support or posting messages to the Internet for help on things in the manual. The second is that you miss many of the powerful features of the various programs. When you call support, you usually get a quick and simple answer. Tech support does not have the time to train you how to use a particular program. Two weeks later, when you try to do something else with the same program, you're on the phone again.

The biggest problem is that people see the long list of files containing the necessary information and are immediately intimidated. Although they would rather spend the money to have support explain things rather than spend time "wading" through documentation, it is not as easy with Linux. There is no tech support office. There is an increasing number of consulting firms specializing in Linux, but most companies cannot afford the thousands of dollars needed to get that kind of service.

The nice thing is that you don't have to. You neither have to wade through the manuals nor spend the money to have support hold your hand. Most of the necessary information is available on-line in the form of manual pages man-pages and other documentation.

Built into the system is a command to read these man-pages: **man**. By typing man <command>, you can find out many details about the command <command>. There are several different options to man that you can use. You can find out more about them by typing man man, which will bring up the man man-page or, the man-page for man.

When referring to a particular command in Linux documentation, you very often will see the name followed by a letter or number in parenthesis, such as ls1. This indicates that the **ls** command can be found in section 1 of the man-pages. This dates back to the time when man-pages came in books as they often still do. By including the section, you could more quickly find what you were looking for. Here I will be making references to files usually as examples. I will say only what section the files are in when I explicitly point you toward the man-page.

For a list of what sections are available, see the table below or the man man-page. If you are looking for the man-page of a particular command and know what section it is in, it is often better to specify the section. Sometimes there are multiple man-pages in different sections. For example, the passwd man-page in section 1 lists the details of the passwd command. The passwd man-page in section 5, lists the details of the **/etc/passwd** file. Therefore,if you wanted the man-page on the passwd file, you would use the -S option for "section" and then to specify section 4, you would call up the man-page like this:

```
man -S 5 passwd
```

Section	Description
1	Commands, Utilities and other executable programs, which are typically user-related
2	System calls
3	Library calls
4	Special files, typically device files in /dev
5	File formats and their respective conventions, layout
6	Games
7	Macro packages
8	System administration commands
9	Kernel routines

Table - Manual Page Sections

Man-pages usually have the same basic format, although not all of the different sections are there for every man-page. At the very top is the section NAME. This is simply the name of the command or file being discussed. Next is the SYNOPSIS section, which provides a brief overview of the command or file. If the man-page is talking about a command or utility, the SYNOPSIS section may list generalized examples of how the command syntax is put together. The tar man-page is a good example.

The DESCRIPTION section, is just that: a description of the command. Here you get a detailed overview about what the command does or what information a particular file contains. Under OPTIONS, you will find details of the various command line switches and parameters, if any. The SEE ALSO section lists other man-pages or other documentation, if any, that contain addition information. Often if there is an info page see below for this man-page it is listed here. BUGS is a list of known bugs, other problems and limitations the program might have. Sometimes, there is an AUTHOR section, which lists the authors of the program and possibly how to contact them.

Note that these sections are just a sampling and not all man-pages have these sections. Some man-pages have other sections that are not applicable to other man-pages. In general, the section headings are pretty straightforward. If all else fails, look at the man man-page.

In many cases, each section has its own man page. By running

```
man -k intro
```

you can see which sections have an introduction, which sometimes provides useful information about that section of man-pages.

Sometimes applications will provide their own man-pages and end up putting them in a directory that the normal man command doesn't use. If the installation routine for the application is well written, then you should not have a problem. Otherwise you need to tell the man command where to look. Some distributions use the **`/etc/manpath.config`** file

which has its own man-page, which contains among other things the directories that man should search. You might also have to define the MANPATH variable explicitly to tell the system where to look. Note that typically, if the MANPATH variable is set., the manpath.config file is ignored.

Often the manual pages are not stored in the original form, but in a pre-formatted form "cat pages". This is done to speed up the display, so that the man pages do not need to be processed each time they are called. I have worked on some systems where these pages are not created by default and every single man-page reports "No manual entry for whatever". To solve this problem simply run the command catman. It may take a while so be patient.

If you want to look at multiple man-pages, you can simply input them on the same line. For example, to look at the grep and find man-pages, you might have a command that looks like this:

man grep find

By pressing 'q' or waiting until the page is displayed, you will be prompted to go to the next file. If the same term is in multiple sections, you can use the -a option to display all of them. For example:

man –a passwd

Sometimes it will happen that you know there is a command that performs a certain function, but you are not sure what the name is. If you don't know the name of the command, it is hard to look for the man-page. Well, that is what the -k option is for -k for "keyword". The basic syntax is:

man –k keyword

where "keyword" is a keyword in the description of the command you are looking for. Note that "man -k" is the same thing as the apropos command. If you have a command and want to know what the command does, you can use the **whatis** command. For example, like this:

whatis diff

which would give you this:

```
diff 1              - find differences between two files>
```

Paired with **whatis** is the **whereis** command. The **whereis** command will tell you the path to the command that is being executed, just like the **which** that we discussed in the section on directory path. However, **whereis** will also show you other information like the location of the man-pages, source code, and so forth. This might give us something like this:

whereis find

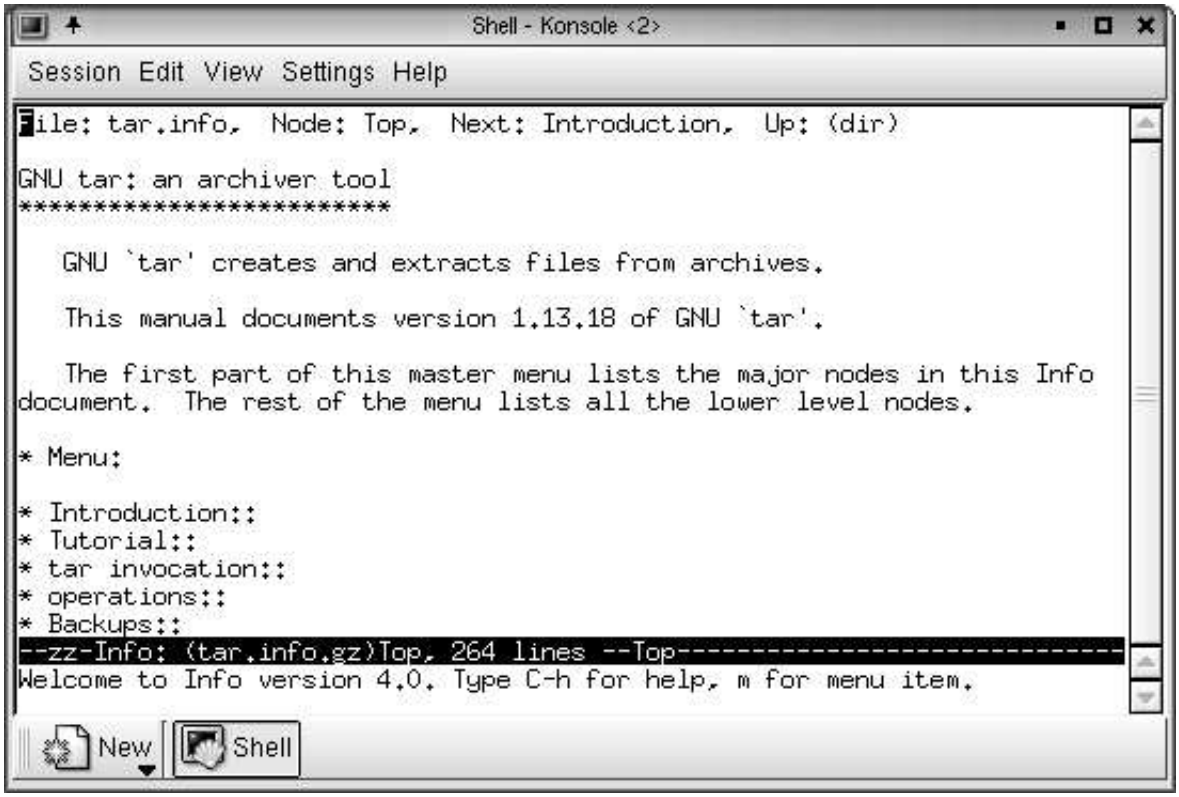
```
find: /usr/bin/find /usr/share/man/man1/find.1.gz /usr/share/man/mann/find.n.gz>
```

Should there be other, related files like /usr/bin/passwd and /etc/passwd, whereis will display these, as well.

For many commands, as well as general system information, there are additional info files that you access using the **info** command. Although there are not as many info files as there are commands, the info files contain information on more aspects of your system. In many cases, the information contained in the info files is identical with the man-pages. To get started, simply type "info" and the command line. To get the information page for particular command, like with man you give the command name as an option to the info command. So, to get information about the tar command, you would input:

```
info tar
```

which would bring up something like this:



An info page.

If you are familiar with the emacs editor, then navigation is fairly easy. However, for the most part, you can move around fairly well using the arrow keys and the enter key. As you can see in the image above, menu items are indicated with an asterisk *. Move with an arrow key or the tab key until the desired item is highlighted and then press enter to select that item. Depending how your keyboard is layed out, you can move up and down within each page using the page-up and page-down keys.

Rather than moving through the menu items, you can simply press ’m’ which will prompt you to input the text of the menu item you want to select. You don’t have to input the complete text, but just enough to differentiate it from other items.

Some commands including info itself have a tutorial section. This provides examples and step-by-step instructions how to use the specific command. To reach the info tutorial from the info page for any command, simply press ’h’ for "help".

SuSE takes this even further by providing you an online copy of their online support knowledge base. This can also be accessed on the internet here.

Before installing any Linux system it is best to know if there is anything to watch out for. For commercial software, this is usually the release notes. Often there is a file in the root directory of the CD if that’s what you are installing from called README or README.1ST which mentions the things to look out for. Typically when you download software or even source code from the Internet, there is a README file. If this file does not give you specific information about installing the software, it will tell you where to find it.

2.3 Other Resources

The Internet is full of resources to find out more about your Linux system. The most obvious places are the home page of the particular distribution you have, but there are many, many more sites that provide information, such as Linux.Org and the Linux Documentation Project, as we discussed in the section on Linux documentation. For a collection of links that I have found useful, check out the main "More Info" page.

One extremely useful place to find information is netnews. Actually, it would be more appropriate to say "places" as there are thousands of newsgroups, hundreds which apply to computers and dozens which apply specifically to Linux. Most are archived on www.deja.com, which, as of this writing, is being redirected to Google Groups. They have a **20 year** archive of the various news groups, not just those related to computers. Here you can also post, but you need to register first.

If you have a Internet Services Provider (ISP) that also provides its own news server, then you might want to consider a local newsreader such as knode, which comes with the KDE. Using a local reader has the advantage of being able to subscribe to newsgroups from various topics, such as both Linux and music, allowing you to easily bounce between the groups you like.

Newsgroups are broken into "hierarchies", or general groupings of particular topics. For example, the "comp" hierarchy is about computers, the "rec" hierarchy is for recreation. For the comp.os.linux newsgroup, click here.

Other good sources of information are mailing lists. The difference between a mailing list and newsgroup is that a copy of each message sent to a mailing list is also sent to every single member. This means that depending on the mailing list and how many you get, you could have hundreds of email messages each day. With newsgroups, you download them as you need them. Depending on your newsreader, you might download all of the messages (which could take quite a long time) or you can download just the headers and then the contents of the messages as you need to.

Mailing lists also have the advantage of being able to filter messages into sub-directories based on their content, sender and so forth. Also, most mailing lists allow only members to submit messages, whereas typically anyone can post to a newsgroup. This means there is often a lot of junk in the newsgroups, such as advertisements, Linux opponents who just want to start arguments and so forth. Since you are required to provide your email address for a mailing list, you cannot be so anonymous and things are *usually* a lot more pleasant.

To get a list of some of the currently available mailing lists send a message to majordomo@vger.kernel.org, which contains just the word "lists". To get detailed help information send a message with the word "help".

2.3.1 Linux Certification

This article was made available through the courtesy of Linux For You (www.linuxforu.com) -- Asia's First Linux Magazine.

Linux Certification The Right Choice?

If you have Linux experience,it is good.In addition,if you have a Linux certification it is even better.For an insight into the world of certifications and their advantages,read this article.

For a long time now, news is on IT job cuts and low demand of IT professionals in the country and abroad as well. But if you have decided on making IT your career, chances are that you will never be without a job for long. The IT industry is undoubtedly the fastest growing sector of employment.

For the one wishing to get into IT, Linux provides an area of opportunities. Linux is no longer of interest only to hackers and the open-source community. Today, there are more Web servers running on Linux than on any other operating system. According to a study done recently, more than 75 per cent of the Fortune 500 companies have already deployed Linux or are considering to do so in near future.

With this booming Linux deployment, comes increased demand for professionals who are conversant with Linux technology. This includes users, administrators, programmers, as well as buyers of Linux systems.

LINUX CERTIFICATION

A recognised Linux certification provides a tangible way to show prowess in the operating system. Many job postings have started quoting Linux Certified Professional in recruitment advertisements.

If Linux is to be widely accepted into the computing mainstream, I believe a certification programme is essential. But I would like to state clearly that I do not believe a certification programme could ever replace experience in the hiring process. However, a certification program is ultimately a tool, primarily for marketing and recruitment.

While Certification by itself does not make you a recognised Linux Guru, it provides a great start towards that goal.

Certifications provide an organisational path for students: People who want to learn about Linux, may read a book on Linux, while others find greater benefit in instructor-led classes. But what books? What classes? The certification programme gives the answer to the question where do I begin?

Certification creates industry recognition: Many software and application vendors have spent millions of dollars convincing the IT industry of the value of certification. People, especially managers, perceive that there is value and importance in certification. A Linux certification programme will allow those who value certification to see that Linux has emerged as a viable option.

Counter the no-support argument: Linux opponents are quick to slam Linux for a perceived lack of support. Linux community understands the truth... about the support from newsgroups and mailing lists. But corporate managers are looking for ways to support IT products. New programs from various Linux distributors allow corporations to purchase support contracts, certainly one great step in this direction. But the existence of certified individuals is another. A pool of Linux-certified professionals would counter this argument. Organisations would like to employ certified professionals to ensure optimum utilisation of resources.

Certification turns students into advocates: If students learn all about Linux: how to install, configure and use the operating system, they will then become advocates for Linux as they move within the IT industry. This is both due to the knowledge learned during the course of preparing for certification... and also due to the fact that since a certification candidate has invested a serious amount of time, energy and money into the product, they want to use that actual product. People recommend what they know. We need them to know Linux!

Certification provides an organisational mechanism for training centers: Training centers offer various certification programmes and play a key role in developing skilled manpower. If training centers want to teach Linux, how do they begin offering classes? A certification program allows a training center to provide a path for training that can generate good business and a healthy competition. Further, a Linux certification program will help promote the overall growth of the Linux operating system through the training programme.

If you’re an IT professional working with Linux, you have a handful of options if you want a certification, and the vendors are jockeying for the position as the dominant product.

There is a cottage industry of Linux training companies, which have grown around the certifications. Users can choose among a distribution-specific hands-on training and certification program (Red Hat), a test-and-training combo (Sair Linux and GNU Certification), or a community-driven test where you choose your own training (Linux Professional Institute).

Let s look at some of the options:

CompTIA Linux+: This is an entry-level, vendor-neutral certification intended to demonstrate foundation level proficiency with the Linux operating system. Earning it, requires passing a single exam covering seven domains: planning & implementation; installation; configuration; administration; system maintenance; troubleshooting; and identifying, installing, and maintaining system hardware. The exam is in multiple-choice

format and consists of 95 questions. It s available through Prometric and VUE testing centers.

Linux Professional Institute Certified (LPIC): The Linux Professional Institute (LPI) is a non-profit organisation formed specifically for the purpose of creating a vendor-neutral certification programme for Linux. The group began organising in late 1998 and officially incorporated on Oct. 25, 1999. The first exams became available in October 2000.

The LPIC programme is designed to offer three certifications signifying increasing skill level, with each requiring two exams. The first two tiers are fully operational; level 3 is yet to be developed. The two exams required for Level 1 certification are titled General Linux I and General Linux II. They cover topics such as: GNU and UNIX commands; devices; Linux file systems; boot, initialisation, shutdown, and run levels; documentation; installation and package management; hardware and architecture; and additional subjects. At Level 2 (again, two exams) candidates will be queried on advanced administration and networking topics, including how to track and solve problems, kernel administration, mail and news services, among other subjects.

Sair Linux/GNU Certified (LCP/LCA/LCE/ MLCE): Sair (pronounced zair) is an acronym for Software Architecture Implementation and Realisation. Sair Inc. started out in 1992 as a software development firm, only turning its attention to building a Linux certification programme in 1999. As in a familiar dot-com story, Sair was acquired by Wave Technologies, which was in turn acquired by Thomson Learning, which is the current owner/operator of the Sair certification programme.

Sair certification was originally created with three levels: Linux/GNU Certified Administrator (LCA), Linux/GNU Certified Engineer (LCE), and Master Linux/GNU Certified Engineer (MLCE), each requiring passage of four exams. The original design was cleanly organised around four system usage areas (thus four exams at each level): Linux installation; network connectivity; system administration; and security, ethics, and privacy.

Red Hat Certified Engineer (RHCE): Red Hat Inc. has been a favourite in the Linux marketplace virtually since its inception. It s also a leader in the world of Linux certification. The first RHCE exam was administered in February 1999, when the vendor-neutral Linux certification vendors were just getting organised.

To date more than 5,000 people have earned the RHCE title. While not an astounding number, but to appear for the exams, candidates must travel to a Red Hat testing center.

Unlike the other certification vendors, Red Hat doesn’t offer an entry-level option. There is only one exam, aimed at intermediate to advanced users of the Red Hat distribution of Linux. The exam is a three-part affair that includes a written test (1 hour); a server install and network services configuration lab (2.5 hours); and a diagnostics and troubleshooting lab (2.5 hours).

The exam covers installing and configuring Red Hat Linux; understanding limitations of hardware; configuring basic networking and file systems; configuring the X Windowing System; basic security, common network (IP) services, basic diagnostics and troubleshooting, and Red Hat Linux system administration.

CHOOSING A CERTIFICATION

As you decide which Linux certification to pursue, consider the skill level you ultimately wish to have, as well as your current abilities. It may be necessary to hop from one certification programme to another to meet your long-term goals. Neither of the multi-level programmes has their advanced certifications up nor running yet. This doesn t reflect lack of progress, but rather a shortage of people who have yet to acquire lower-level certifications.

Linux, as a technology, has matured much faster in its development than any other technology or operating system. It has changed the information technology landscape for good. In such a scenario, certification will always play a very vital role in the selection criteria of IT professionals for organisations. Given the opportunity in today s industry, Experience with Certification adds weight to being selected. certification

This article was written by Shankar Iyer, Head Training, Red Hat India.

Chapter 3 Working with the System

Whether you login using the GUI or a character console, the way you interact with a Linux system is essentially the same. You must first be able to identify yourself to the system by providing the appropriate information. This information is your user name or login name and a password. As we discuss in other sections, by default Linux will prompt you for this information when the system is started. Once you have correctly identified yourself, you are given access to the system. What happens next will depend on whether you are using the GUI or a character console.

For simplicities sake, we will first talk about interacting with a system from a character console, which can also be referred to as a character terminal. One important reason for this is that even if you are using a GUI, you still have access to a character terminal window and the way you interact is the same. Also, when you connect to a remote system (i.e. using something like telnet) the way you interact is the same as well.

3.1 Backing-up and Restoring Files

If you're using Linux in your company, the system administrator probably does regular backups (assuming he wants to keep his job). However, if you are administering your own Linux system (i.e. it's your home workstation), then it is up to you to ensure that your data and important system files are safe.

The computer boom of the 1990's put a PC in everyone's house, but it did not provide them with the same awareness and knowledge that computer users of the 1970's and 80's had. With point-n-click and plug-n-play computers became a "black box" where the insides are an unknown. You turn on your computer and it just works. When you turn on your computer and it doesn't work, people don't know what to do. It's possible that the computer can be repaired, but if the hard disk is damaged, the data may be unrecoverable.

If all you use your computer for it to surf the internet, then there may not be any valuable data on your system. However, if you write letters, manage your bank accounts or many other things on your computer, you may have files you want to keep. Although you may think it is safe, it is extremely important how quickly even a small defect can make the data inaccessible. Therefore, you need to be able to store that data on an external medium to keep it safe.

The data stored on an external medium like a floppy or CD ROM is called a backup. The process of storing the data (or making the copy) is called "making a backup". Sometimes, I will copy files onto a different hard disk. If the first one crashes, I still have access. Even if you don't have a different drive, you can still protect your data to a limited extent by copying it onto a different partition or even a different directory. If the drive develops a problem at the exact spot where your data is, it might be safe some place else. However, if the whole drive dies, your data is gone.

One advantage of storing it on an external device, is that if the computer completely crashes the data is completely safe. In many cases, companies will actually store the data at a different

location in case the building burns down or there is some other disaster (no kidding!).

Linux provides a number of different useful tools to help you backup your system. Perhaps the most commonly used tool is tar, probably because of its simplicity. For example let's say you wanted to make a backup copy of the entire directory **/data**, the command might look like this:

```
tar cvf data.backup /data
```

Where data.backup is the name of the file in which you want to store the backups of your files. When tar completes, you have a single file which contains a copy of everything in the **/data** directory. One thing that we discussed in another section, is that Linux treats hardware just like regular files. Therefore instead of using a filename you could use the name of a device file, like this:

```
tar cvf /dev/tape /data
```

Assuming you had a tape drive on your system, and you had named it **/dev/tape**, this command would backup your data to your tape drive.

Note that there are tools available for Linux which allow you to recover files which you have removed from your system. This goes into too much depth for now, but there is a how-to.

There are other options which you can use with tar that are very useful:

-z compresses - This compresses the file using gzip after it has made the archive. This should not be done telling tar to use different compression programs. See the tar man-page for details.

-T, --files-from=FILENAME - Here you can specify a file which contains a list of files you want to archive. This is useful for system configuration files spread out across your system. Although you could copy all of your system files into one directory prior to making a backup, this method is much more efficient.

Typically when files are removed on Linux they're gone for good. You can create your own "trash can" by creating a shell function that actually moves the file into a different directory for example:

```
function rm() {
mv $1 /home/jimmo/trashcan
}
```

Then when you want to clear out the trash, you would use the full path to the **rm** command: **/bin/rm**.

Keep in mind that simply being able to backup files is not enough. Often you do not have enough space on your tapes to do a complete backup of your system every day. Sometimes, doing a complete backup takes so long that even if you start right as people go home, there is not enough time to finish before they come back to work. Further, when trying to restore a complete backup of your system, it will take longer to find the files you need and thus will

takes longer to get people back to work. Therefore, you need a backup strategy, which we discuss in the section on problem solving.

3.2 Interacting with the System

It is common to have people working on UNIX systems that have *never* worked on a computer before or have only worked in pure windowing environments, like on a Macintosh. When they get to the command line, they are lost. On more than one occasion, I have talked to customers and I have asked them to type in cd /. There is a pause and I hear: click-click-click-click-click-click-click-click-click-click-click-click. "Hmmm," I think to myself, "that's too many characters." So I ask them what they typed, and they respond, "cd-space-slash."

We need to adhere to some conventions throughout this site to make things easier. One is that commands that I talk about will be in your path unless I say otherwise. Therefore, to access them, all you need to do is input the name of the command without the full path.

The second convention is the translation of the phrases "input the command," "enter the command," and "type in the command." These are translated to mean "input/enter/type in the command *and press Enter*." I don't know how many times I have talked with customers and have said "type in the command" and then asked them for what happens and their response is, "Oh, you want me to press Enter?" Yes! Unless I say otherwise, always press Enter after inputting, entering, or typing in a command.

Simply having shell is probably not enough for most users. Although you could probably come up with an interesting and possibly useful shell script, more than likely you're going to need some commands to run. There are literally hundreds of different commands that come with your system by default and there are many more different variations of these commands, which you can download from the Internet.

Sometimes the commands you issue are not separate files on the hard disk, but rather are built-in to your shell. For example, the **`cd`** command, which is used to change directories, is part of the shell, whereas the **`ls`** command, which is used to display the contents of directories is a separate program. In some cases one shell has a particular command built-in, but it is not available in another shell.

If you ever run into trouble and are confused about the behavior of your shell one important thing to know is what shell you have. If you weren't told what shell you had when your account was created or you are installing Linux for the first time and really don't know, there are a couple of ways of finding out. The first is to simply ask the shell. This is done by accessing the $SHELL environment variable. (We discuss environment variables in detail in the section on shell variables .) This is done using the echo command like this:

```
echo $SHELL
```

As you might guess, the echo command simply displays on the screen exactly what you told it, in this case we told it to display the $SHELL variable. (We know it is a variable because of the leading $, which we also will discuss in section on shell variables .) What should probably

happen is you get something like this:

```
/bin/bash>
```

In this case, the shell is **/bin/bash**. We can also find out what shell we are using by seeing which programs we are currently running. With Linux, as with other Unix dialects, a running program is called a " process", and you check your processes using the ps command (for process status). You can start it with no arguments simply by inputting **ps** and pressing the enter key. This will probably get you something like this:

```
   PID TTY          TIME CMD
21797 pts/1    00:00:00 bash
 6060 pts/1    00:00:00 ps
>
```

In this case we see under the heading CMD (for command) only "bash" and not the full pathname as in the previous example. However, there are options to the ps command which will show us the path.

The shell you are using is just one piece of information the system maintains in regard to your current session. Much of this information is stored in the form of variables, like your shell. These variables are set for you when you login to the system. You can also set variables yourself using the set command. This might look like this:

```
set VAR=value
```

Where VAR is the variable name and "value" is the value which you assigned to that variable. Note that it is not until you want to access the value of the variable that you preceded with the $. To find out the contents of all variables, you would use the set command by itself with no arguments. This gives you a long list of variables.

When you login to the system you start in your "home" directory, which can be stored in the $HOME variable. As we discussed earlier, to change your current directory (also called your working directory) you use the cd command. If you wanted to return to your home directory, you can issue the command cd $HOME and your shell will pass the value of the $HOME variable to the cd, which would then change directories for you. (Note that typically if you use the cd command with no arguments at all, you change to your home directory by default.)

One part of your environment which is extremely useful to know is the directory you are currently in. To do this you might want to tell the system to simply print your current working directory. This is done with the pwd command, which simply displays the full path to your current directory.

It is also useful to see what files and directories reside in your current directory. This is done with the ls command (short for "list"). Without the options the ls command provides you a simple list of what is in your current directory, without any additional information. The output might look like this:

```
prompt# ls
letter.txt      memo.txt     picture.jpg
>
```

you can use the -l option to get a "long" listing of the files and directories. This might show you something like this:

```
prompt# ls -l
-rw-r--r--    1 jimmo    users        2457 Feb 13 22:00 letter.txt
-rw-r--r--    1 jimmo    users        7426 Feb 15 21:33 memo.txt
-rw-r--r--    1 jimmo    users      34104 Feb 14 21:31 picture.jpg
>
```

This information includes the permissions on the file, who owns the file, the size, and so forth. Details of this can be found in the section on file permissions.

For a more detailed discussion on how various shells behave see the section on shells.

There are many ways to do the things you want to do. Some use a hammer approach and force the answer out of the system. In many cases, there are other commands that do the exact same thing without all the gyrations. So, what I am going to try to do here is step through some of the logic (and illogic) that I went through when first learning Linux. That way, we can all laugh together at how silly I was, and maybe you won't make the same mistakes I did.

Every dialect of UNIX that I have seen has the ls command. This gives a directory listing of either the current directory if no argument is given, or a listing of a particular file or directory if arguments are specified. The default behavior under Linux for the ls command is to list the names of the files in a single column. Try it and see.

It is a frequent (maybe not common) misconception for new users to think that they have to be in a particular directory to get a listing of it. They will spend a great deal of time moving up and down the directory tree looking for a particular file. Fortunately, they don't have to do it that way. The issue with this misunderstanding is that every command is capable of working with paths, as is the operating system that does the work. Remember our discussion of Linux basics. Paths can be relative to our current directory, such as ./directory, or absolute, such as /home/jimmo/directory.

For example, assume that you have a subdirectory of your current working directory called letters. In it are several subdirectories for types of letters, such as business, school, family, friends, and taxes. To get a listing of each of these directories, you could write

```
ls ./letters/business
ls ./letters/school
ls ./letters/family
ls ./letters/friends
ls ./letters/taxes
```

Because the ls command lets you have multiple commands on the same line, you also could have issued the command like this:

```
ls ./letters/business ./letters/school ./letters/family
./letters/friends ./letters/taxes
```

Both will give you a listing of each of the five directories. Even for five directories, typing all of that is a pain. You might think you could save some typing if you simply entered

```
ls ./letters
```

However, this gives you a listing of all the files and directories in ./letters, not the subdirectories. Instead, if you entered

```
ls ./letters/*
```

the shell would expand the wildcard (*) and give you a listing of both the **./letters** directory as well as the directories immediately below **./letters**, like the second example above. If each of the subdirectories is small, then this might fit onto one screen. If, on the other hand, you have 50 letters in each subdirectory, they are not all going to fit on the screen at once. Remember our discussion on shell basics? You can use the pipe (|) to send the command through something like more so that you could read it a page at a time.

3.3 Logging In

Like many contexts, the name you use as a "real person" is not necessarily the way the system identifies you. With Linux you see yourself as a particular user, such as jimmo, whereas the system might see you as the number 12709. For most of the time this difference is pretty much irrelevant, as the system makes the conversion between user name and this number the user ID or UID itself. There are a few cases where the difference is important, which we will get to in other sections. In addition, you could make the conversion yourself to see what your user ID is, which we will also get to elsewhere.

The place where this system makes this conversion is the file **/etc/passwd**. You can take a look at it by typing

```
cat /etc/passwd
```

from the command line. Here you find one user per line. The details of this file can be found in the section on administering user accounts. Note that there are a number of predefined, system users in the **/etc/passwd** file and they do not relate to real users. For security reasons, some system administrators will delete many of these users. However, you should leave them alone unless you know what you are doing.

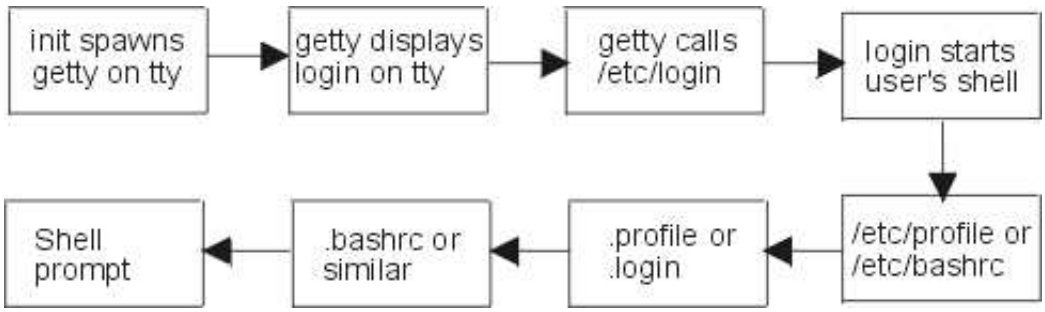
If you are installing Linux on your system at home, more than likely you are prompted to select a user name and password during installation. This is the account you should use for normal day-to-day work. You should not use the system administration account: root. The root account is "all powerful" and can essentially do anything to the system you can imagine. If you make a mistake, the system is unforgiving and if you are working as root, the results can be catastrophic. If you are used to Windows, this is a major difference. If you administer a Windows NT/2000 system, you are typically in the Administrators group. This means you automatically have administrator privileges, which means you can accidentally cause damage.

With Linux, you generally have to make a conscious effort to switch to the root user to carry out your administrative task which is fairly easy and safer.

if you are using a Linux system that someone else installed perhaps at work, then an account will need to be created for you. The name of the account will more than likely be unique to your company, so there's no real need to discuss the different possibilities here. Ask your system administrator for details.

Keep in mind that both the user name and password are case sensitive. That means it will make a difference if you spell either with upper or lowercase letters. Using a lowercase letter when creating the account, then using an uppercase letter when attempting to login, will prevent you from gaining access to the system.

The process of identifying yourself to the system, whereby you provide your user name and password, is referred to as " logging in". When the system is ready for you to login you are presented with a login prompt. That is, you are prompted to login. How the login prompt looks differs among the various Linux distributions, but generally has the word "login:". Once you input your username and press the enter key, you are then prompted to input your password. This is done simply by displaying the word " password:". Typically, you end up seeing something like this:



This example shows a login across a network using telnet.

It is possible that even after installing Linux and rebooting you do not see a login prompt. One possible reason is that the login prompt is there, but you just don't see it. Perhaps, some one turned off a monitor, the machine has gone into power saving mode, and so forth. It is also possible that your Linux system has been configured to automatically start into the GUI. If the video card was not correctly configured, you may not be able to see anything on the screen at all. In this case all is not lost because Linux provides you with something called "virtual consoles". We go into these in detail in the section on what Linux does..

Bear in mind that the system keeps track of who is currently logged into the system as well as who has logged in in the past. Since you might be held accountable for things which were done with your account, it is important to keep your password secret in order to prevent someone from gaining improper access to the system. This is the reason that when you login you see your username displayed on the screen as you type it, but not your password. It is possible that someone could be looking over your shoulder and sees your password as you type.

Once you login to system it starts your "login shell". In essence, a shell is a command line interpreter, meaning that the shell interprets and executes the commands you enter. If you are familiar with either DOS or Windows, the command prompt is basically the same thing as a Linux shell, in that it is also a command line interpreter. The shell indicates that it is ready to accept a new command by displaying a shell prompt or command prompt. Typical prompts are the #,%, or $. It is also possible that your system administrator has defined a different prompt. It is common to include your username, your current directory, the system you are working on or other pieces of information. You can find more details about how the system perceives the shell in the section on processes in the operating system introduction.

One of the first things that you should do when you login to a system where someone else created the user account for you is to change your password. Obviously, the person creating your account will have to know your password in order to be able to tell you what it is. Your password should be something which is easy for you to remember so you do not need to write it down, but extremely difficult for someone else to guess. What constitutes good and bad passwords is something we get into in the section on security.

The Linux command which is used to change your password is called **`passwd`**, which you can start from any shell prompt. As a security feature, the passwd program will first prompt you for your old password before allowing you to change to a new one. This is to ensure that you are the person the system thinks you are. Perhaps you have left your desk for a moment and someone wants to play a trick on you and changes your password.

The exception is the root account. The system administrator must have the ability to change any password and could not do this in every case if the old password was always required. For example, you may have forgotten your password and need the administrator to change it for you. It would do no good to ask the administrator to change your password if he had to know it first. This is one reason why you need to be careful when you are working with the root user account.

3.4 Logging Out

If you are running Linux at home, then there is probably no need to stop your sessions when you’re finished working. However, if your session is accessible by others, then it is not a bad idea to "log out" when you’re done. Where "logging in" connects you to a session, "logging out" disconnects you from that session. In most cases it is sufficient simply to type in the word exit to end your session (exit is actually a command built-in to many shells). It is also possible to exit a shell session by pressing CTRL-D (holding down the control key and pressing the letter "d").

After you log out, the system typically sends a new login prompt to your terminal.

The details of this process can be found in the section on logging in .

3.5 When Things Go Wrong

Until you become very accustomed to using Linux you're likely to make mistakes (which also happens to people who have been working with Linux for a long time). In this section, we'll be talking about some common mistakes and problems that occur when you first start using Linux.

Usually when you make mistakes the system will let you know in some way. When using using the command line, the system will tell you in the form of error messages. For example, if you try to execute a command and the command does not exist, the system may report something like this:

```
bash: some_command: command not found>
```

Such an error might occur if the command exists, but it does not reside in a directory in your search path. You can find more about this in the section on directory paths.

The system may still report an error, even if it can execute the command. For example, if the command acts on a file that does not exist. For example, the **more** displays the contents of a file. If the file you want to look at does not exist, you might get the error:

```
some_file: No such file or directory>
```

In the first example, the error came from your shell as it tried to execute the command. In the second case, the error came from the **more** command as it encountered the error when trying to access the file.

In both these cases, the problem is pretty obvious. In some cases, you are not always sure. Often you include such commands within shell scripts and want to change the flow of the script based on errors or success of the program. When a command ends, it provides its "exit code" or "return code" in the special variable $?. So after a command fails, running this command will show you the exit code:

echo $?

Note that it is up to the program to both provide the text message and the return code. Sometimes you end up with a text message that does not make sense (or there is no text at all), so all you get is the return code, which is probably even less understandable. To make a translation between the return code and a text message, check the file **/usr/include/asm/errno.h**.

It has happened before that I have done a directory listing and saw a particular file. When I tried to remove it, the system told me the file did not exist. The most likely explanation is that I misspelled the filename, but that wasn't it. What can happen sometimes is that a control character ends up becoming part of the filename. This typically happens with the backspace as it is not always defined as the same character on every system. Often the backspace is CTRL-H, but it could happen that you create a file on a system with a different backspace key and end up creating a filename with CTRL-H. When you display the file it prints out the name and when it reaches the backspace backs up one character before continuing. For example

your ls output might show you this file:

```
jimmo>
```

However trying to erase it you get an error message. To see any " non printable" characters you would use the -q option to ls. This might show you:

```
jimmoo?>
```

Which says the file name actually contains two o's and a trailing backspace. Since the backspace erased the last 'o' in the display, you do not see it when the file name is displayed normally.

Sometimes you lose control of programs and they seem to "runaway". In other cases, a program may seem to hang and freeze your terminal. Although it is possible because of a bug in the software or a flaky piece of hardware, oftentimes the user makes a mistake he was not even aware of. This can be extremely frustrating for the beginner, since you do not even know how you got yourself into the situation, let alone how to get out.

When I first started learning Unix (even before Linux was born) I would start programs and quickly see that I needed to stop them. I knew I could stop the program with some combination of the control key and some other letter. In my rush to stop the program, I would press the control key and many different letters in sequence. On some occassions, the program simply stop and goes no further. On other occasions, the program would appear to stop, but I would later discover that it was still running. What happened was that I hit a combination that did not stop the program but did something else.

In the first example, where the program would stop and go no further, I had "suspended" the program. In essence, I'd put it to sleep and it would wait for me to tell it to start up again. This is typically done by pressing CTRL-S. This feature can obviously be useful in the proper circumstance, but when it is unexpected and you don't know what you did, it can be very unnerving. To put things right, you resume the command with CTRL-Q.

In the second example, where the program seemed to have disappeared, I had also suspended the program but at the same time had put in the "background". This special feature of Unix shells dates from the time before graphical interfaces were common. It was a great waste of time to start a program and then have to wait for it to complete, when all you were interested in was the output which you could simply write to file. Instead you put a program in the background and the shell returned to the prompt, ready for the next command. It's sometimes necessary to do this once a command is started, which you do by pressing CTRL-Z, which suspends the program, but returns to the prompt. You then issue the bg command, which starts the previous command in the background. (This is all part of "job control" which is discussed in another section.)

To stop the program, what I actually wanted to do was to "interrupt" it. This is typical done with CTRL-C.

What this actually does is to send a signal to the program, in this case an interrupt signal. You can define which signal is sent when you press any given combination of keys. We talk about

this in thesection on terminal settings.

When you put a command in the background which send output to the screen, you need to be careful about running other programs in the meantime. What could happen is that your output gets mixed up, making it difficult to see which output belongs to which command.

There have been occasions where I have issued a command and the shell jumps to the next line, then simply displays a greater than symbol (>). What this often means is that the shell does not think you are done with the command. This typically happens when you are enclosing something on the command line quotes in you forget to close the quotes. For example if I wanted to search for my name in a file I would use the grep command. If I were to do it like this:

```
grep James Mohr filename.txt
```

I would get an error message saying that the file "Mohr" did not exist.

To issue this command correctly I would have to include my name inside quotes, like this:

```
grep "James Mohr" filename.txt
```

However, if I forgot the final quote, for example, the shell would not think the command was done yet and would perceive the enter key that I pressed as part of the command. What I would need to do here is to interrupt the command, as we discussed previously. Note this can also happen if you use single quotes. Since the shell does not see any difference between a single quote and an apostrophe, you need to be careful with what you type. For example if I wanted to print the phrase "I'm Jim", I might be tempted to do it like this:/P>

```
echo I'm Jim
```

However, the system does not understand contractions and thinks I have not finished the command.

As we will discuss in another section, you can send the output of a command to a file. This is done with the greater than symbol (>). The generic syntax looks like this:

```
command > filename
```

This can cause problems if the command you issue expects more arguments than you gave it. For example, if I were searching the contents of a file for occurrences of a particular phrase

```
grep phrase > filename
```

What would happen is the shell would drop down to the next line and simply wait forever or until you interrupted the command. The reason is that the grep command can also take input from the command line. It is waiting for you to type in text, before it will begin searching. Then if it finds the phrase you are looking for it will write it into the file. If that's not what you want the solution here is also to interrupt the command. You can also enter the end of file character (CTRL-D), which would tell grep to stop reading input.

One thing to keep in mind, is that you can put a program in the background even if the shell does not understand job control. In this case, it is impossible to bring the command back to the foreground in order to interrupt. You need to do something else. As we discussed earlier, Linux provides you a tool to display the processes which you are currently running (the ps command). Simply typing ps on the command line might give you something like this:

```
  PID TTY          TIME CMD
29518 pts/3    00:00:00 bash
30962 pts/3    00:00:00 ps
>
```

The PID column in the ps output is the process identifier (PID). You can send any process a signal to stop execution if you know its PID. This is the kill command and syntax is quite simple:

kill <PID>

By default, the kill command sends a termination signal to that process. Unfortunately, there are some cases where a process can ignore that termination signal. However, you can send a much more urgent "kill" signal like this:

kill -9 <PID>

Where "9" is the number of the kill signal. For details on what other signals can be sent and the behavior in different circumstances look at the kill man-page or simply try kill -l.

In some circumstances, it is not easy to kill processes by their PID. For example, if something starts dozens of other processes, it is ineffective to try to input all of their PIDs. To solve this problem Linux has the **killall** command and takes the command name instead of the PID.

There have been cases where I have frantically tried to stop a runaway program and repeatedly pressed Ctrl-C. The result is that the terminal gets into an undefined state whereby it does not react properly to any input, that is when you press the various keys. For example, pressing the enter key may not bring you to a new line (which it normally should do). If you try executing a command, it’s possible to command is not executed properly, because the system has not identified the enter key correctly. You can return your terminal to a "sane" condition by inputting:

stty sane Ctrl-J

The Ctrl-J character is the line feed character and is necessary as the system does not recognize the enter key.

It has happned to me a number of times, that the screen saver was activated and it was if the system had simply frozen. There were no error messages, no keys work and the machine did not even respond across the network (telnet, ping, etc.) Unfortunately, the only thing to do in this case is to turn the computer off and then on again.

On the other hand, you can prevent these problems in advance. THe most likely cause it that the Advanced Power Management (APM) is having problems. In this case, you should disable the APM within the system BIOS. Some machines also have something called "hardware monitoring". This can cause problems, as well, and should be disabled.

Problems can also be caused by the Advanced Programmable Interrup controller. This can be deactivated by changing the boot string used by either LILO or grub. In addtion, you can disable it by adding "disableapic" to your boot line.

3.6 Accessing Disks

For the most part, you need to tell Linux what to do. This gives you a lot of freedom, because it does what you tell it, but people new to Linux have a number of pre-conceptions from Windows. One thing you need to do is to tell the system to mount devices like hard disks and CD-ROMs.

Typically Linux sees the CD-ROMs the same way it does hard disks, since they are usually all on the IDE controllers. The device /dev/hda is the master device on the first controller, /dev/hdb is the slave device on the first controller, /dev/hdc is the master on the second controller and /dev/hdd is the slave device on the second controller.

To mount a filesystem/disk you use the **mount** command. Assuming that your CD-ROM is the master device on the second controller you might mount it like this:

```
mount DEVICE DIRECTORY
```

```
mount /dev/hdc /media/cdrom
```

Sometimes **/media/cdrom** does not exist, so you might want to try this.

```
mount /dev/hdc /mnt
```

Sometimes the system already know about the CD-ROM device, so you can leave off either component:

```
mount /media/cdrom
mount /dev/hdc
```

You can then cd into **/media/cdrom** and you are on the CD.

Details on this can be found in the section on hard disks and file systems.

Chapter 4 Shells and Utilities

Most UNIX users are familiar with "the shell"; it is where you input commands and get output on your screen. Often, the only contact users have with the shell is logging in and immediately starting some application. Some administrators, however, have modified the system to the point where users never even see the shell, or in extreme cases, have eliminated the shell completely for the users.

Because the Linux GUI has become so easy to use, it is possible that you can go for quite a long time without having to input commands at a shell prompt. If your only interaction with the operating system is logging into the GUI and starting applications, most of this entire site can only serve to satisfy your curiosity. Obviously, if all you ever do is start a graphical application, then understanding about shell is not all that important. However, if you are like most Linux users, understanding the basic workings of the shell will do wonders to improve your ability to use the system to its fullest extent.

Up to this point, we have referred to the shell as an abstract entity. In fact, in most texts, it is usually referred to as simply "the shell", although there are many different shells that you can use, and there is always a program that must be started before you can interact with "the shell". Each has its own characteristics (or even quirks), but all behave in the same general fashion. Because the basic concepts are the same, I will avoid talking about specific shells until later.

In this chapter, we are going to cover the basic aspects of the shell. We'll talk about how to issue commands and how the system responds. Along with that, we'll cover how commands can be made to interact with each other to provide you with the ability to make your own commands. We'll also talk about the different kinds of shells, what each has to offer, and some details of how particular shells behave.

4.1 The Shell

As I mentioned in the section on introduction to operating systems, the shell is essentially a user's interface to the operating system. The shell is a command line interpreter, just like other operating systems. In Windows you open up a "command window" or "DOS box" to input commands, which is nothing other than a command line interpreter. Through it, you issue commands that are interpreted by the system to carry out certain actions. Often, the state where the system is sitting at a prompt, waiting for you to type input, is referred to (among other things) as being at the shell prompt or at the command line.

For many years before the invention of graphical user interfaces, such as X-Windows (the X Windowing System, for purists), the only way to input commands to the operating system was through a command line interpreter, or shell. In fact, shells themselves were thought of as wondrous things during the early days of computers because prior to them, users had no direct way to interact with the operating system.

Most shells, be they under DOS, UNIX, VMS, or other operating systems, have the same input characteristics. To get the operating system to do anything, you must give it a command.

Some commands, such as the date command under UNIX, do not require anything else to get them to work. If you type in date and press Enter, that's what appears on your screen: the date.

Some commands need something else to get them to work: an *argument*. Some commands, like **mkdir** (used to create directories), work with only one argument, as in mkdir directory_name. Others, like **cp** (to copy files), require multiple arguments, as in

```
cp file1 file2
```

In many cases, you can pass flags to commands to change their behavior. These flags are generally referred to as *options*. For example, if you wanted to create a series of sub-directories without creating every one individually, you could run **mkdir** with the -p option, like this:

```
mkdir –p one/two/three/four
```

In principle, anything added to the command line after the command itself is an argument to that command. The convention is that an option changes the behavior, whereas an argument is acted upon by the command. Let's take the **mkdir** command as an example:

```
mkdir dir_name
```

Here we have a single argument which is the name of the directory to be created. Next, we add an option:

```
mkdir –p sub_dir/dir_name
```

The -p is an option. Using the terminology discussed, some arguments are optional and some options are required. That is, with some commands you must always have an option, such as the **tar** command. Some commands don't always need to have an argument, like the **date** command.

Generally, options are preceded by a dash (-), whereas arguments are not. I've said it before and I will say it again, nothing is certain when it comes to Linux or UNIX, in general. By realizing that these two terms are often interchanged, you won't get confused when you come across one or the other. I will continue to use *option* to reflect something that changes the command's behavior and *argument* to indicate something that is acted upon. In some places, you will also see arguments referred to as "operands". An operand is simply something on which the shell "operates", such as a file, directory or maybe even simple text.

Each program or utility has its own set of arguments and options, so you will have to look at the man-pages for the individual commands. You can call these up from the command line by typing in

```
man <command_name>
```

where <command_name> is the name of the command you want information about. Also, if you are not sure what the command is, many Linux versions have the whatis command that will give you a brief description. There is also the **apropos** command, which searches

through the man-pages for words you give as arguments. Therefore, if you don't know the name of the command, you can still find it.

Arguments (whether they are options or operands) which are enclosed in square brackets ([ ]) are optional. In some cases, there are optional components to the optional arguments, so you may end up having brackets within brackets.

An ellipsis (...) Indicates that the preceding arguments can be repeated. For example, the ls command can take multiple file or directory names as arguments as well as multiple options. Therefore, you might have a usage message that looks like this:

```
ls [OPTION] ... [FILE] ...
```

This tells us that no options are required, but if you wanted you could use multiple options. It also tells us that no file name is required, but if you wanted you could use multiple ones.

Words that appeared in angle brackets (< >) or possibly in italics in the printed form, indicate that the word is a place holder. Like in the example below:

```
man <filename>
```

Many commands require that an option appear immediately after the command and before any arguments. Others have options and arguments interspersed. Again, look at the man-page for the specifics of a particular command.

Often, you just need a quick reminder as to what the available options are and what their syntax is. Rather than going through the hassle of calling up the man-page, a quick way is to get the command to give you a *usage message.* As its name implies, a usage message reports the usage of a particular command. I normally use -? as the option to force the usage message, as I cannot think of a command where -? is a valid option. Your system may also support the --help (two dashes) option. More recent versions of the various commands will typically give you a usage message if you use the wrong option. Note that fewer and fewer commands support the -?.

To make things easier, the letter used for a particular option is often related to the function it serves. For example, the -a option to ls says to list "all" files, even those that are "hidden". On older versions of both Linux and Unix, options typically consisted of a single letter, often both upper and lowercase letters. Although this meant you could have 52 different options it made remembering them difficult, if they were multiple functions that all began with the same letter. Multiple options can either be placed separately, each preceded by a dash, or combined. For example, both of these commands are valid and have the exact same effect:

```
ls -a -l
```

```
ls -al
```

In both cases you get a long listing which also included all of the hidden files.

Newer versions of commands typically allow for **both** single letter options and "long options" which use full words. For example, the long equivalent of -a would be --all. Note that the long options are preceded with *two* dashes because it would otherwise be indistinguishable from the -a followed by two -l options.

Although it doesn't happen too often, you might end up with a situation where one of the arguments to your command starts with a dash (-), for example a file name. Since options typically start with a dash, the shell cannot figure out that it is an argument and not a long line of options. Let's assume that some application I had created a file called "**-jim**". If I wanted to do a simple listing of the file, I might try this:

```
ls -jim
```

However, since the shell first tries to figure out what options are being used before it shows you the listing, it thinks that these are all options and gives you the error message:

```
ls: invalid option -- j
Try 'ls --help' for more information.
>
```

You can solve this problem with *some* commands by using two dashes to tell the command that what follows is actually an argument. So to get the listing in the previous example, the command might look like this:

```
ls -- -jim
```

4.2 The Search Path

It may happen that you know there is a program by a particular name on the system, but when you try to start it from the command line, you are told that the file is not found. Because you just ran it yesterday, you assume it has gotten removed or you don't remember the spelling.

The most common reason for this is that the program you want to start is not in your search path. Your search path is a predefined set of directories in which the system looks for the program you type in from the command line (or is started by some other command). This saves time because the system does not have to look through every directory trying to find the program. Unfortunately, if the program is not in one of the directories specified in your path, the system cannot start the program unless you explicitly tell it where to look. To do this, you must specify either the full path of the command or a path relative to where you are currently located.

Lets look at this issue for a minute. Think back to our discussion of files and directories. I mentioned that every file on the system can be referred to by a unique combination of path and file name. This applies to executable programs as well. By inputting the complete path, you can run any program, whether it is in your path or not.

Lets take a program that is in everyones path, like date (at least it should be). The date program resides in the /bin directory, so its full path is /bin/date. If you wanted to run it, you could type in /bin/date, press Enter, and you might get something that looks like this:

```
Sat Jan 28 16:51:36 PST 1995>
```

However, because date is in your search path, you need to input only its name, without the path, to get it to run.

One problem that regularly crops up for users coming from a DOS environment is that the *only* place UNIX looks for commands is in your path. However, even if not specified in your path, the first place DOS looks is in your current directory. This is not so for UNIX. UNIX only looks in your path.

For most users, this is not a problem as the current directory is included in your path by default. Therefore, the shell will still be able to execute something in your current directory. Root does not have the current directory in its path. In fact, this is the way it should be. If you want to include the current directory in roots path, make sure it is the last entry in the path so that all "real" commands are executed before any other command that a user might try to "force" on you.

Assume a malicious user created a "bad" program in his/her directory called more. If root were to run more in that users directory, it could have potentially disastrous results. (Note that the current directory normally always appears at the end of the search path. So, even if there was a program called more in the current directory, the one in /bin would probably get executed first. However, you can see how this could cause problems for root.) To figure out exactly which program is actually being run, you can use the (what else?) which command.

Newer versions of the bash-Shell can be configured to not only complete commands automatically by inputting just part of the command, but also arguments to the command, as well as directory names. See the bash man-page for more details.

Commands can also be starting by including a directory path, whether or not they are in you search path. You can use relative or absolute paths, usually with the same result. Details on this can be found in the section on directory paths.

It is common to have people working on UNIX systems that have *never* worked on a computer before or have only worked in pure windowing environments, like on a Macintosh. When they get to the command line, they are lost. On more than one occasion, I have talked to customers and I have asked them to type in cd /. There is a pause and I hear: click-click-click-click-click-click-click-click-click-click-click-click-click. "Hmmm," I think to myself, "that's too many characters." So I ask them what they typed, and they respond, "cd-space-slash."

We need to adhere to some conventions throughout these pages to make things easier. One is that commands that I talk about will be in your path unless I say otherwise. Therefore, to access them, all you need to do is input the name of the command without the full path.

The second convention is the translation of the phrases "input the command," "enter the command," and "type in the command." These are translated to mean "input/enter/type in the command *and press Enter*." I don't know how many times I have talked with customers and have said "type in the command" and then asked them for what happens and their response is, "Oh, you want me to press Enter?" Yes! Unless I say otherwise, always press Enter after

inputting, entering, or typing in a command.

4.3 Directory Paths

As we discussed in the section on the seach path, you can often start programs simply by inputting their name, provided they lie in your search path. You could also start a program by referencing it through a *relative path*, the path in relation to your current working directory. To understand the syntax of relative paths, we need to backtrack a moment. As I mentioned, you can refer to any file or directory by specifying the path to that directory. Because they have special significance, there is a way of referring to either your *current directory* or its *parent directory*. The current directory is referenced by "." and its parent by ".." (often referred to in conversation as "dot" and "dot-dot").

Because directories are separated from files and other directories by a /, a file in the current directory could be referenced as **`./file_name`** and a file in the parent directory would be referenced as **`../file_name`**. You can reference the parent of the parent by just tacking on another ../, and then continue on to the root directory if you want. So the file **`../../file_name`** is in a directory two levels up from your current directory. This slash (/) is referred to as a *forward slash,* as compared to a *back-slash* (\), which is used in DOS to separate path components.

When interpreting your command line, the shell interprets everything up to the last / as a directory name. If we were in the root (upper-most) directory, we could access date in one of several ways. The first two, date and /bin/date, we already know about. Knowing that **`./`** refers to the current directory means that we could also get to it like this: **`./bin/date`**. This is saying relative to our current directory (./), look in the bin subdirectory for the command date. If we were in the /bin directory, we could start the command like this: **`./date`**. This is useful when the command you want to execute is in your current directory, but the directory is not in your path. (More on this in a moment.)

We can also get the same results from the root directory by starting the command like this: bin/date. If there is a ./ at the beginning, it knows that everything is relative to the current directory. If the command contains only a /, the system knows that everything is relative to the root directory. If no slash is at the beginning, the system searches until it gets to the end of the command or encounters a slash whichever comes *first*. If there is a slash there (as in our example), it translates this to be a subdirectory of the current directory. So executing the command bin/date is translated the same as **`./bin/date`**.

Let's now assume that we are in our home directory, **`/home/jimmo`** (for example). We can obviously access the date command simply as date because it's in our path. However, to access it by a relative path, we could say **`../../bin/date`**. The first **`../`** moves up one level into /home. The second ../ moves up another level to **`/.`** From there, we look in the subdirectory bin for the command date. Keep in mind that throughout this whole process, our current directory does not change. We are still in **`/home/jimmo`**.

Searching your path is only done for commands. If we were to enter vi file_name (vi is a text editor) and there was no file called file_name in our current directory, vi would start editing a new file. If we had a subdirectory called text where file_name was, we would have to access it

either as vi ./text/file_name or vi text/file_name. Of course, we could access it with the absolute path of vi /home/jimmo/text/file_name.

When you input the path yourself (either a command or a file) The shell interprets each component of a pathname before passing it to the appropriate command. This allows you to come up with some pretty convoluted pathnames if you so choose. For example:

cd /home/jimmo/data/../bin/../../chuck/letters

This example would be interpreted as first changing into the directory **/home/jimmo/data/**, moving back up to the parent directory (..), then into the subdirectory bin, back into the parent and its parent (../../) and then into the subdirectory **chuck/letters**. Although this is a pretty contrived example, I know many software packages that rely on relative paths and end up with directory references similar to this example.

4.4 Shell Variables

The shell's environment is all the information that the shell will use as it runs. This includes such things as your command search path, your *logname* (the name you logged in under), and the terminal type you are using. Collectively, they are referred to as your *environment variables* and individually, as the "so-and-so" environment variable, such as the TERM environment variable, which contains the type of terminal you are using.

When you log in, most of these are set for you in one way or another. (The mechanism that sets all environment variables is shell-dependent, so we will talk about it when we get to the individual shells.) Each environment variable can be viewed by simply typing echo $VARIABLE. For example, if I type

echo $LOGNAME

I get:

```
jimmo>
```

Typing

echo $TERM

I get:

```
ansi>
```

One very important environment variable is the PATH variable. Remember that the PATH tells the shell where it needs to look when determining what command it should run. One of the things the shell does to make sense of your command is to find out exactly what program you mean. This is done by looking for the program in the places specified by your PATH variable.

Although it is more accurate to say that the shell looks in the directories specified by your PATH environment variable, it is commonly said that the shell "searches your path." Because this is easier to type, I am going to use that convention here.

If you were to specify a path in the command name, the shell does not use your PATH variable to do any searching. That is, if you issued the command bin/date, the shell would interpret that to mean that you wanted to execute the command date that was in the bin subdirectory of your current directory. If you were in / (the root directory), all would be well and it would effectively execute /bin/date. If you were somewhere else, the shell might not be able to find a match.

If you do not specify any path (that is, the command does not contain any slashes), the system will search through your path. If it finds the command, great. If not, you get a message saying the command was not found.

Let's take a closer look at how this works by looking at my path variable. From the command line, if I type

`echo $PATH`,

I get

```
/usr/local/bin:/bin:/usr/bin:/usr/X11/bin:/home/jimmo/bin:/:.>
```

WATCH THE DOT!

If I type in date, the first place in which the shell looks is the /bin directory. Because that's where date resides, it is executed as /bin/date. If I type in vi, the shell looks in /bin, doesn't find it, then looks in /usr/bin, where it does find vi. Now I type in getdev. (This is a program I wrote to translate major device numbers into the driver name. Don't worry if you don't know what a major number is. You will shortly.) The shell looks in /usr/local/bin and doesn't find it. It then looks in /bin. Still not there. It then tries /usr/bin and /usr/X11/bin and still can't find it. When it finally gets to /home/jimmo/bin, it finds the getdev command and executes it. (Note that because I wrote this program, you probably won't have it on your system.)

What would happen if I had not yet copied the program into my personal bin directory? Well, if the getdev program is in my current directory, the shell finds a match with the last "." in my path. (Remember that the "." is translated into the current directory, so the program is executed as ./getdev.) If that final "." was missing or the getdev program was somewhere else, the shell could not find it and would tell me so with something like

`getdev: not found`

Note that shell variables are only accessible from the current shell. In order for them to be accessible to child processes (i.e. sub-processes) they must be made available using the **`export`** command. In the system-wide shell configuration file or "profile" (etc/profile) many variables, such as PATH are exported. More information on processes can be found in the section on processes in the chapter "Introduction to Operating Systems".

4.5 Permissions

All this time we have been talking about finding and executing commands, but there is one issue that I haven't mentioned. That is the concept of permissions. To access a file, you need to have permission to do so. If you want to read a file, you need to have read permission. If you want to write to a file, you need to have write permission. If you want to execute a file, you must have execute permission.

Permissions are set on a file using the chmod command or when the file is created the details of which I will save for later. You can read the permissions on a file by using either the l command or ls -l. At the beginning of each line will be ten characters, which can either be dashes or letters. The first position is the type of the file, whether it is a regular file -, a directory d, a block device file b, and so on. Below are some examples of the various file types.

```
-rw-rw--r-   1  jimmo   support   1988   Sep   15   10:05   letter.txt
crw-rw----   1  root    tty       4,1    Jul   2    10:05   /dev/tty1
brw-rw----   1  root    disk      1,1    Mar   8    07:34   /dev/hda1
drwxr-xr-x   2  root    bin       2048   May   26   14:15   /bin
pr--r--r--   1  root    root      0      Jul   2    09:48   /proc/1/maps
lrwxrwxrwx   1  root    root      5      Mar   28   15:45   /usr/bin/vi -> elvis
```

Image - Various file types.

- - regular file
c - character device
b - block device
d - directory
p - named pipe
l - symbolic link

We'll get into the details of these files as we move along. If you are curious about the format of each entry, you can look at the ls man-page.

The next nine positions are broken into three groups. Each group consists of three characters indicating the permissions. They are, in order, readr, writew, and executex. The first set of characters indicates what permissions the owner of the file has. The second set of characters indicates the permissions for the group of that file. The last set of characters indicates the permissions for everyone else.

If a particular permission is not given, a dash - will appear here. For example, rwx means all three permissions have been given. In our example above, the symbolic link **`/usr/bin/vi`** has read, write, and execute permissions for everyone. The device nodes /dev/tty1 and /dev/hda1 have permissions rw- for the owner and group, meaning only read and write, but not execute permissions have been given. The directory **`/bin`** has read and execute permissions for everyone r-x, but only the owner can write to it rwx.

For directories, the situation is slightly different than for regular files. If you do not have read permission on a directory, you cannot read the contents of that directory. Also, if you do not have write permission on a directory, you cannot write to it. This means that you cannot create a new file in that directory. Execute permissions on a directory mean that you can search it or list its contents. That is, if the execution bit is not set on a directory but the read bit is, you can see what files are in the directory but cannot execute any of the files or even change into that directory. If you have execution permission but no read permission, you can execute the files, change directories, but not see what is in the files.

Write permission on a directory also has an interesting side effect. Because you need to have write permission on a directory to create a new file, you also need to have write permission to remove an existing file. Even if you do not have write permission on the file itself, if you can write to the directory, you can erase the file.

At first this sounds odd. However, remember that a directory is nothing more than a file in a special format. If you have write permission to a directory-file, you can remove the references to other files, thereby removing the files themselves.

If we were to set the permissions for all users so that they could read, write, and execute a file, the command would look this:

```
chmod 777 filename
```

You can also use symbolic permissions to accomplish the same thing. We use the letters u, g, and o to specify the userowner, group, and others for this file, respectively. The permissions are then r for read, w for write, and x for execute. So to set the permissions so that the owner can read and write a file, the command would look like this:

```
chmod u=rw filename
```

Note that in contrast to the absolute numbers, setting the permissions symbolically is additive. So, in this case, we would just change the user's permissions to read and write, but the others would remain unchanged. If we changed the command to this

```
chmod u+w filename
```

we would be adding write permission for the user of that file. Again, the permissions for the others would be unchanged.

To make the permissions for the group and others to be the same as for the user, we could set it like this

```
chmod go=u filename
```

which simply means "change the mode so that the permissions for the group and others equals the user." We also could have set them all explicitly in one command, like this

```
chmod u=rw,g=rw,o=rw filename
```

which has the effect of setting the permissions for everyone to read and write. However, we don't need to write that much.

Combining the commands, we could have something that looks like this:

```
chmod u=rw, go=u filename
```

This means "set the permissions for the user to read and write, then set the permissions for group and others to be equal to the user."

Note that each of these changes is done in sequence. So be careful what changes are made. For example, let's assume we have a file that is read-only for everyone. We want to give everyone write permission for it, so we try

```
chmod u+w,gu=o filename
```

This is a typo because we meant to say go=u. The effect is that we added read permissions for the user, but then set the permissions on the group and user to the same as others.

We might want to try adding the write permissions like this:

```
chmod +w filename
```

This works on some systems, but not on the Linux distributions that I have seen. According to the man-page, this will not change those permissions where the bits in the UMASK are set. More on this later. See the chmod man-page for details.

To get around this, we use a to specify all users. Therefore, the command would be

```
chmod a+w filename
```

There are a few other things that you can do with permissions. For example, you can set a program to change the UID of the process when the program is executed. For example, some programs need to run as root to access other files. Rather than giving the user the root password, you can set the program so that when it is executed, the process is run as root. This is a Set-UID, or SUID program. If you want to run a program with a particular group ID, you would use the SGID program with the s option to chmod, like this

```
chmod u+s program
```

or

```
chmod g+s program
```

There are a few other special cases, but I will leave it up to you to check out the chmod man-page if you are interested.

When you create a file, the access permissions are determined by their file creation mask. This is defined by the UMASK variable and can be set using the umask command. One thing to keep in mind is that this is a mask. That is, it masks out permissions rather than assigning them. If you remember, permissions on a file can be set using the chmod command and a

three-digit value. For example

```
chmod 600 letter.john
```

explicitly sets the permissions on the file letter.john to 600 read and write permission for the user and nothing for everyone else. If we create a new file, the permissions might be 660 read/write for user and group. This is determined by the UMASK. To understand how the UMASK works, you need to remember that the permissions are octal values, which are determined by the permissions bits. Looking at one set of permissions we have

bit:	*2*	*1*	*0*
value:	*4*	*2*	*1*
symbol:	*r*	*w*	*x*

which means that if the bit with value 4 is set bit 2, the file can be read; if the bit with value 2 is set bit 1, the file can be written to; and if the bit with value 1 is set bit 0, the file can be executed. If multiple bits are set, their values are added together. For example, if bits 2 and 1 are set read/write, the value is 4+2=6. Just as in the example above, if all three are set, we have 4+2+1=7. Because there are three sets of permissions owner, group, others, the permissions are usually used in triplets, just as in the chmod example above.

The UMASK value *masks* out the bits. The permissions that each position in the UMASK masks out are the same as the file permissions themselves. So, the left-most position masks out the owner permission, the middle position the group, and the right most masks out all others. If we have UMASK=007, the permissions for owner and group are not touched. However, for others, we have the value 7, which is obtained by setting all bits. Because this is a mask, all bits are unset. The way I remember this is that the bits are inverted. Where it is set in the UMASK, it will be unset in the permissions, and vice versa.

The problem many people have is that the umask command does *not* force permissions, but rather limits them. For example, if we had UMASK=007, we could assume that any file created has permissions of 770. However, this depends on the program that is creating the file. If the program is creating a file with permissions 777, the umask will mask out the last bits and the permissions will, in fact, be 770. However, if the program creates permissions of 666, the last bits are still masked out. However, the new file will have permissions of 660, *not* 770. Some programs, like the C compiler, do generate files with the execution bit bit 0 set. However, most do not. Therefore, setting the UMASK=007 does not force creation of executable programs, unless the program creating the file does itself.

Lets look at a more complicated example. Assume we have UMASK=047. If our program creates a file with permissions 777, then our UMASK does nothing to the first digit, but masks out the 4 from the second digit, giving us 3. Then, because the last digit of the UMASK is 7, this masks out everything, so the permissions here are 0. As a result, the permissions for the file are 730. However, if the program creates the file with permissions 666, the resulting permissions are 620. The easy way to figure out the effects of the UMASK are to subtract the UMASK from the default permissions that the program sets. Note that all negative values become 0.

As I mentioned, one way the UMASK is set is through the environment variable UMASK. You can change it anytime using the umask command. The syntax is simply

```
umask <new_umask>
```

Here the <new_umask> can either be the numeric value e.g., 007 or symbolic. For example, to set the umask to 047 using the symbolic notation, we have

```
umask u=,g=r,o=rwx
```

This has the effect of removing no permissions from the user, removing read permission from the group, and removing all permissions from others.

Being able to change the permissions on a file is often not enough. What if the only person that should be able to change a file is not the owner? Simple! You change the owner. This is accomplished with the chown command, which has the general syntax:

```
chown new_owner filename
```

Where "new_owner" is the name of the user account we want to sent the owner of the file to, and "filename" is the file we want to change. In addition, you can use chown to change not only the owner, but the group of the file as well. This has the general syntax:

```
chown new_owner.new:group filename
```

Another useful trick is the ability to set the owner and group to the same ones as another file. This is done with the --reference= option, which sets to the name of the file you are referencing. If you want to change just the group, you can use the chgrp command, which has the same basic syntax as chown. Not that both chgrp and chmod can also take the --reference= option. Further, all three of these commands take the -R option, which recursively changes the permissions, owner or group.

4.6 Regular Expressions and Metacharacters

Often, the arguments that you pass to commands are file names. For example, if you wanted to edit a file called letter, you could enter the command vi letter. In many cases, typing the entire name is not necessary. Built into the shell are special characters that it will use to expand the name. These are called *metacharacters*.

The most common metacharacter is *. The * is used to represent any number of characters, including zero. For example, if we have a file in our current directory called letter and we input

```
vi let*
```

the shell would expand this to

```
vi letter
```

Or, if we had a file simply called let, this would match as well.

Instead, what if we had several files called letter.chris, letter.daniel, and letter.david? The shell would expand them all out to give me the command

vi letter.chris letter.daniel letter.david

We could also type in vi letter.da*, which would be expanded to

vi letter.daniel letter.david

If we only wanted to edit the letter to chris, we could type it in as vi *chris. However, if there were two files, letter.chris and note.chris, the command vi *chris would have the same results as if we typed in:

vi letter.chris note.chris

In other words, no matter where the asterisk appears, the shell expands it to match *every* name it finds. If my current directory contained files with matching names, the shell would expand them properly. However, if there were no matching names, file name expansion couldn't take place and the file name would be taken literally.

For example, if there were no file name in our current directory that began with letter, the command

vi letter*

could not be expanded and we would end up editing a new file called (literally) letter*, including the asterisk. This would not be what we wanted.

What if we had a subdirectory called letters? If it contained the three files **letter.chris**, **letter.daniel**, and **letter.david**, we could get to them by typing

vi letters/letter*. This would expand to be:

vi letters/letter.chris letters/letter.daniel letters/letter.david

The same rules for path names with commands also apply to files names. The command

vi letters/letter.chris

is the same as

vi ./letters/letter.chris

which as the same as

vi /home/jimmo/letters/letter.chris

This is because the shell is doing the expansion before it is passed to the command. Therefore, even directories are expanded. And the command

vi le*/letter.*

could be expanded as both letters/letter.chris and lease/letter.joe., or any similar combination

The next wildcard is ?. This is expanded by the shell as one, and only one, character. For example, the command vi letter.chri? is the same as vi letter.chris. However, if we were to type in vi letter.chris? (note that the "?" comes after the "s" in chris), the result would be that we would begin editing a *new* file called (literally) letter.chris?. Again, not what we wanted. This wildcard could be used if, for example, there were two files named letter.chris1 and letter.chris2. The command vi letter.chris? would be the same as

vi letter.chris1 letter.chris2

Another commonly used metacharacter is actually a pair of characters: [ ]. The square brackets are used to represent a list of possible characters. For example, if we were not sure whether our file was called **letter.chris** or **letter.Chris,** we could type in the command as: vi letter.[Cc]hris. So, no matter if the file was called **letter.chris** or **letter.Chris**, we would find it. What happens if both files exist? Just as with the other metacharacters, both are expanded and passed to **vi**. Note that in this example, vi letter.[Cc]hris appears to be the same as vi letter.?hris, but it is not always so.

The list that appears inside the square brackets does not have to be an upper- and lowercase combination of the same letter. The list can be made up of any letter, number, or even punctuation. (Note that some punctuation marks have special meaning, such as *, ?, and [ ], which we will cover shortly.) For example, if we had five files, letter.chris1-letter.chris5, we could edit all of them with vi letter.chris[12435].

A nice thing about this list is that if it is consecutive, we don’t need to list all possibilities. Instead, we can use a dash (-) inside the brackets to indicate that we mean a range. So, the command

vi letter.chris[12345]

could be shortened to

vi letter.chris[1-5]

What if we only wanted the first three and the last one? No problem. We could specify it as

vi letter.chris[1-35]

This does not mean that we want files letter.chris1 through letter.chris35! Rather, we want letter.chris1, letter.chris2, letter.chris3, and letter.chris5. All entries in the list are seen as individual characters.

Inside the brackets, we are not limited to just numbers or just letters. we can use both. The command vi letter.chris[abc123] has the potential for editing six files: letter.chrisa, letter.chrisb, letter.chrisc, letter.chris1, letter.chris2, and letter.chris3.

If we are so inclined, we can mix and match any of these metacharacters any way we want. We can even use them multiple times in the same command. Let's take as an example the command

vi *.?hris[a-f1-5]

Should they exist in our current directory, this command would match *all* of the following:

letter.chrisa	note.chrisa	letter.chrisb	note.chrisb	letter.chrisc
note.chrisc	letter.chrisd	note.chrisd	letter.chrise	note.chrise
letter.chris1	note.chris1	letter.chris2	note.chris2	letter.chris3
note.chris3	letter.chris4	note.chris4	letter.chris5	note.chris5
letter.Chrisa	note.Chrisa	letter.Chrisb	note.Chrisb	letter.Chrisc
note.Chrisc	letter.Chrisd	note.Chrisd	letter.Chrise	note.Chrise
letter.Chris1	note.Chris1	letter.Chris2	note.Chris2	letter.Chris3
note.Chris3	letter.Chris4	note.Chris4	letter.Chris5	note.Chris5

Also, any of these names without the leading letter or note would match. Or, if we issued the command:

vi *.d*

these would match

letter.daniel note.daniel letter.david note.david

Remember, I said that the shell expands the metacharacters only with respect to the name specified. This obviously works for file names as I described above. However, it also works for command names as well.

If we were to type dat* and there was nothing in our current directory that started with dat, we would get a message like

```
dat*: not found >
```

However, if we were to type /bin/dat*, the shell could successfully expand this to be /bin/date, which it would then execute. The same applies to relative paths. If we were in / and entered ./bin/dat* or bin/dat*, both would be expanded properly and the right command would be executed. If we entered the command /bin/dat[abcdef], we would get the right response as well because the shell tries all six letters listed and finds a match with /bin/date.

An important thing to note is that the shell expands as long as it can before it attempts to interpret a command. I was reminded of this fact by accident when I input /bin/l*. If you do an

ls /bin/l* you should get the output:

```
-rwxr-xr-x    1 root     root        22340 Sep 20 06:24 /bin/ln
-r-xr-xr-x    1 root     root        25020 Sep 20 06:17 /bin/login
-rwxr-xr-x    1 root     root        47584 Sep 20 06:24 /bin/ls
>
```

At first, I expected each one of the files in /bin that began with an "l" (ell) to be executed. Then I remembered that expansion takes place *before* the command is interpreted. Therefore, the command that I input, /bin/l*, was expanded to be

/bin/ln /bin/login /bin/ls

Because /bin/ln was the first command in the list, the system expected that I wanted to link the two files together (what /bin/ln is used for). I ended up with error message:

```
/bin/ln: /bin/ls: File exists>
```

This is because the system thought I was trying to link the file /bin/login to **/bin/ls**, which already existed. Hence the message.

The same thing happens when I input /bin/l? because the /bin/ln is expanded first. If I issue the command /bin/l[abcd], I get the message that there is no such file. If I type in

/bin/l[a-n]

I get:

```
/bin/ln: missing file argument>
```

because the **/bin/ln** command expects two file names as arguments and the only thing that matched is /bin/ln.

I first learned about this aspect of shell expansion after a couple of hours of trying to extract a specific subdirectory from a tape that I had made with the cpio command. Because I made the tape using absolute paths, I attempted to restore the files as /home/jimmo/letters/*. Rather than restoring the entire directory as I expected, it did nothing. It worked its way through the tape until it got to the end and then rewound itself without extracting any files.

At first I assumed I made a typing error, so I started all over. The next time, I checked the command before I sent it on its way. After half an hour or so of whirring, the tape was back at the beginning. Still no files. Then it dawned on me that hadn't told the cpio to overwrite existing files unconditionally. So I started it all over again.

Now, those of you who know cpio realize that this wasn't the issue either. At least not entirely. When the tape got to the right spot, it started overwriting everything in the directory (as I told it to). However, the files that were missing (the ones that I really wanted to get back) were still not copied from the backup tape.

The next time, I decided to just get a listing of all the files on the tape. Maybe the files I wanted were not on this tape. After a while it reached the right directory and lo and behold, there were the files that I wanted. I could see them on the tape, I just couldn't extract them.

Well, the first idea that popped into my mind was to restore *everything*. That's sort of like fixing a flat tire by buying a new car. Then I thought about restoring the entire tape into a temporary directory where I could then get the files I wanted. Even if I had the space, this still seemed like the wrong way of doing things.

Then it hit me. I was going about it the wrong way. The solution was to go ask someone what I was doing wrong. I asked one of the more senior engineers (I had only been there less than a year at the time). When I mentioned that I was using wildcards, it was immediately obvious what I was doing wrong (obvious to him, not to me).

Lets think about it for a minute. It is the *shell* that does the expansion, not the command itself (like when I ran /bin/l*). The shell interprets the command as starting with /bin/l. Therefore, I get a listing of all the files in **/bin** that start with "l". With **cpio** , the situation is similar.

When I first ran it, the shell interpreted the files (/home/jimmo/data/*) before passing them to **cpio**. Because I hadn't told cpio to overwrite the files, it did nothing. When I told cpio to overwrite the files, it only did so for the files that it was told to. That is, only the files that the shell saw when it expanded /home/jimmo/data/*. In other words, cpio did what it was told. I just told it to do something that I hadn't expected.

The solution is to find a way to pass the wildcards to cpio. That is, the shell must ignore the special significance of the asterisk. Fortunately, there is a way to do this. By placing a back-slash (\) before the metacharacter, you remove its special significance. This is referred to as "escaping" that character.

So, in my situation with **cpio**, when I referred to the files I wanted as /home/jimmo/data/*, the shell passed the arguments to cpio as /home/jimmo/data/*. It was then cpio that expanded the * to mean all the files in that directory. Once I did that, I got the files I wanted.

You can also protect the metacharacters from being expanded by enclosing the entire expression in single quotes. This is because it is the shell that first expands wildcard before passing them to the program. Note also that if the wild card cannot be expanded, the entire expression (including the metacharacters) is passed as an argument to the program. Some programs are capable of expanding the metacharacters themselves.

Another symbol with special meaning is the dollar sign ($). This is used as a marker to indicate that something is a variable. I mentioned earlier in this section that you could get access to your login name environment variable by typing:

```
echo $LOGNAME
```

The system stores your login name in the environment variable LOGNAME (note no "$"). The system needs some way of knowing that when you input this on the command line, you are talking about the variable LOGNAME and not the literal string LOGNAME. This is done with the "$".Several variables are set by the system. You can also set variables yourself and use them later on. I'll get into more detail about shell variables later.

So far, we have been talking about metacharacters used for searching the names of files. However, metacharacters can often be used in the arguments to certain commands. One example is the grep command, which is used to search for strings within files. The name grep comes from Global Regular Expression Print (or Parser). As its name implies, it has something to do with regular expressions. Lets assume we have a text file called documents, and we wish to see if the string "letter" exists in that text. The command might be

```
grep letter documents
```

This will search for and print out every line containing the string "letter." This includes such things as "letterbox," "lettercarrier," and even "love-letter." However, it will not find "Letterman," because we did not tell grep to ignore upper- and lowercase (using the -i option). To do so using regular expressions, the command might look like this

```
grep [Ll]etter documents
```

Now, because we specified to look for either "L" or "l" followed by "etter," we get both "letter" and "Letterman." We can also specify that we want to look for this string only when it appears at the beginning of a line using the caret (^) symbol. For example

```
grep ^[Ll]etter documents
```

This searches for all strings that start with the "beginning-of-line," followed by either "L" or "l," followed by "etter." Or, if we want to search for the same string at the end of the line, we would use the dollar sign to indicate the end of the line. Note that at the beginning of a string, the dollar sign is treated as the beginning of the string, whereas at the end of a string, it indicates the end of the line. Confused? Lets look at an example. Lets define a string like this:

```
VAR=^[Ll]etter
```

If we echo that string, we simply get ^[Ll]etter. Note that this includes the caret at the beginning of the string. When we do a search like this

```
grep $VAR documents
```

it is equivalent to

```
grep ^[Ll]etter documents
```

Now, if write the same command like this

```
grep $VAR$ documents
```

This says to find the string defined by the VAR variable(^[Ll]etter) , but only if it is at the end of the line. Here we have an example, where the dollar sign has *both* meanings. If we then take it one step further:

```
grep ^$VAR$ documents
```

This says to find the string defined by the VAR variable, but only if it takes up the entry line. In other words, the line consists only of the beginning of the line (^), the string defined by VAR, and the end of the line ($).

Often you need to match a series of repeated characters, such as spaces, dashes and so forth. Although you could simply use the asterisk to specify any number of that particular character, you can run into problems on both ends. First, maybe you want to match a *minimum* number of that character. This could easily solved by first repeating that character a certain number of times before you use the wildcard. For example, the expression

```
====*
```

would match at least *three* equal signs. Why three? Well, we have explicitly put in three equal signs and the wildcard follows the fourth. Since the asterisk can be *zero* or more, it could mean *zero* and therefore the expression would only match three.

The next problem occurs when we want to limit the maximum number of characters that are matched. If you know exactly how many to match, you could simply use that many characters. What do you do if you have a minimum and a maximum? For this, you enclose the range with curly-braces: {min,max}. For example, to specify at least 5 and at most 10, it would look like this: {5,10}. Keep in mind that the curly braces have a special meaning for the shell, so we would need to escape them with a back-slash when using them on the command line. So, lets say we wanted to search a file for all number combinations between 5 and 10 number long. We might have something like this:

```
grep "[0-9]\{5,10\}" FILENAME
```

This might seem a little complicated, but it would be far more complicated to write an regular expression that searches for each combination individually.

As we mentioned above, to define a specific number of a particular character you could simply input that character the desired number of times. However, try counting 17 periods on a line or 17 lower-case letters ([a-z]). Imagine trying to type in this combination 17 times! You could specify a range with a maximum of 17 *and* a minimum of 17, like this: {17,17}. Although this would work, you could save yourself a little typing by simply including just the single value. Therefore, to match exactly 17 lower-case letters, you might have something like this:

```
grep "[a-z]\{17\}" FILENAME
```

If we want to specify a minimum number of times, without a maximum, we simply leave off the maximum, like this:

```
grep "[a-z]\{17,\}" FILENAME
```

This would match a pattern of at least 17 lower-case letters.

Another problem occurs when you are trying to parse data that is not in English. If you were looking for all letters in an English text, you could use something like this: [a-zA-Z]. However, this would not include German letters, like ä,Ö,ß and so forth. To do so, you would use the expressions [:lower:], [:upper:] or [:alpha:] for the lower-case letters, upper-case letters or all letters, respectively, *regardless* of the language. (Note this assumes that national language support (NLS) is configured on your system, which it normally is for newer Linux distributions.

Other expressions include:

- [:alnum:] - Alpha-numeric characters.
- [:cntrl:] - Control characters.
- [:digit:] - Digits.

- [:graph:] - Graphics characters.
- [:print:] - Printable characters.
- [:punct:] - Punctuation.
- [:space:] - White spaces.

One very important thing to note is that the brackets are part of the expression. Therefore, if you want to include more in a bracket expression you need to make sure you have the correction number of brackets. For example, if you wanted to match any number of alpha-numeric or punctuation, you might have an expression like this: [[:alnum:][:digit:]]*.

Another thing to note is that in most cases, regular expression are expanded as much as possible. For example, let's assume I was parsing an HTML file and wanted to match the *first* tag on the line. You might think to try an expression like this: "<.*>". This says to match any number of characters between the angle brackets. This works if there is only one tag on the line. However, if you have more than one tag, this expression would match *everything* from the first opening angle-bracket to the last closing angle bracket with *everything* inbetween.

4.7 Quotes

One last issue that causes its share of confusion is quotes. In Linux, there are three kinds of quotes: *double-quotes* ("), *single-quotes* (’), and *back-quotes*(‘‘) (also called back-ticks). On most US keyboards, the single-quotes and double-quotes are on the same key, with the double-quotes accessed by pressing Shift and the single-quote key. Usually this key is on the right-hand side of the keyboard, next to the Enter key. On a US-American keyboard the back-quote is usually in the upper left-hand corner of the keyboard, next to the 1.

To best understand the difference between the behavior of these quotes, I need to talk about them in reverse order. I will first describe the back-quotes, or back-ticks.

When enclosed inside back-ticks, the shell interprets something to mean "the output of the command inside the back-ticks." This is referred to as *command substitution*, as the output of the command inside the back-ticks is substituted for the command itself. This is often used to assign the output of a command to a variable. As an example, lets say we wanted to keep track of how many files are in a directory. From the command line, we could say

```
ls | wc
```

The wc command gives me a word count, along with the number of lines and number of characters. The | is a "pipe" symbol that is used to pass the output of one command through another. In this example, the output of the ls command is passed or piped through wc. Here, the command might come up as:

```
7 7 61>
```

However, once the command is finished and the value has been output, we can only get it back again by rerunning the command. Instead, If we said:

count=`ls |wc`

The entire line of output would be saved in the variable count. If we then say echo $count, we get

```
7 7 61>
```

showing me that count now contains the output of that line. If we wanted, we could even assign a multi-line output to this variable. We could use the ps command, like this

trash=`ps`

then we could type in

echo $trash

which gives us:

```
PID TTY TIME CMD 29519 pts/6 00:00:00 bash 12565 pts/6 00:00:00 ps>
```

This is different from the output that ps would give when not assigned to the variable trash:

```
  PID   TTY          TIME CMD
29519 pts/6    00:00:00 bash
12564 pts/6    00:00:00 ps
>
```

The next kind of quote, the single-quote (’), tells the system not to do *any* expansion at all. Lets take the example above, but this time, use single quotes:

count=’ls |wc’

If we were to now type

echo $count

we would get

ls |wc

And what we got was exactly what we expected. The shell did no expansion and simply assigned the literal string "ls | wc" to the variable count. This even applies to the variable operator "$." For example, if we simply say

echo ’$LOGNAME’

what comes out on the screen is

```
$LOGNAME>
```

No expansion is done at all and even the "$" is left unchanged.

The last set of quotes is the double-quote. This has partially the same effect as single-quotes, but to a limited extent. If we include something inside of double-quotes, everything loses its special meaning except for the variable operator ($), the back-slash (\), the back-tick (‘), and the double-quote itself. Everything else takes on its absolute meaning. For example, we could say

```
echo "`date`"
```

which gives us

```
Wed Feb 01 16:39:30 PST 1995>
```

This is a round-about way of getting the date, but it is good for demonstration purposes. Plus, I often use this in shell scripts when I want to log something. Remember that the back-tick first expands the command (by running it) and then the echo echoes it to the screen.

That pretty much wraps up the quote characters. For details on other characters that have special meaning to the shell check out the section on regular expressions. You can get more details from any number of references books on Linux or UNIX in general (if you need it). However, the best way to see what’s happening is to try a few combinations and see if they behave as you expect.

Previously, I mentioned that some punctuation marks have special meaning, such as *, ?, and [ ]. In fact, most of the other punctuation marks have special meaning, as well. We’ll get into more detail about them in the section on basic shell scripting.

4.8 Pipes and Redirection

Perhaps the most commonly used character is "|", which is referred to as the pipe symbol, or simply pipe. This enables you to pass the output of one command through the input of another. For example, say you would like to do a long directory listing of the **/bin** directory. If you type ls -l and then press Enter, the names flash by much too fast for you to read. When the display finally stops, all you see is the last twenty entries or so.

If instead we ran the command **ls -l | more** the output of the ls command will be "piped through more". In this way, we can scan through the list a screenful at a time.

In our discussion of standard input and standard output in Chapter 1, I talked about standard input as being just a file that usually points to your terminal. In this case, standard output is also a file that usually points to your terminal. The standard output of the ls command is changed to point to the pipe, and the standard input of the more command is changed to point to the pipe as well.

The way this works is that when the shell sees the pipe symbol, it creates a temporary file on the hard disk. Although it does not have a name or directory entry, it takes up physical space on the hard disk. Because both the terminal and the pipe are seen as files from the perspective of the operating system, all we are saying is that the system should use different files instead of standard input and standard output.

Under Linux (as well as other UNIX dialects), there exist the concepts of standard input, standard output, and standard error. When you log in and are working from the command line, standard input is taken from your terminal keyboard and both standard output and standard error are sent to your terminal screen. In other words, the shell expects to be getting its input from the keyboard and showing the output (and any error messages) on the terminal screen.

Actually, the three (standard input, standard output, and standard error) are references to files that the shell automatically opens. Remember that in UNIX, everything is treated as a file. When the shell starts, the three files it opens are usually the ones pointing to your terminal.

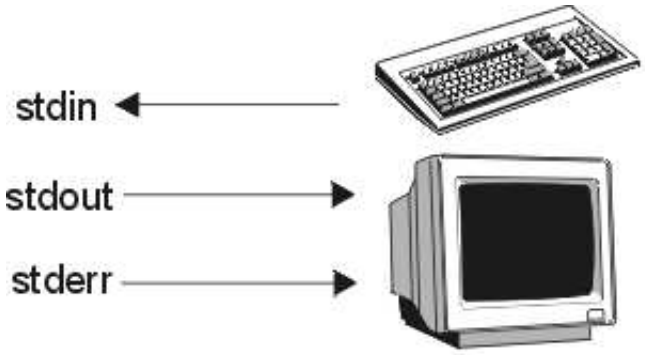
When we run a command like cat, it gets input from a file that it displays to the screen. Although it may appear that the standard input is coming from that file, the standard input (referred to as stdin) is still the keyboard. This is why when the file is large enough and you are using something like **`more`** to display the file one screen at a time and it stops after each page, you can continue by pressing either the Spacebar or Enter key. That's because *standard* input is still the keyboard.

As it is running, more is displaying the contents of the file to the screen. That is, it is going to standard output (stdout). If you try to do a more on a file that does not exist, the message

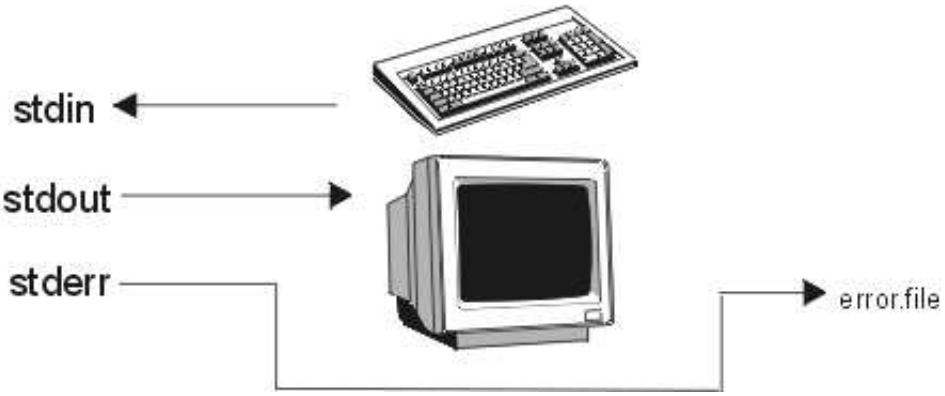
```
file_name: No such file or directory>
```

shows up on your terminal screen as well. However, although it appears to be in the same place, the error message was written to standard error (stderr). (I'll show how this differs shortly.)

One pair of characters that is used quite often, "<" and ">," also deal with stdin and stdout. The more common of the two, ">," redirects the output of a command into a file. That is, it changes standard output. An example of this would be ls /bin > myfile. If we were to run this command, we would have a file (in my current directory) named myfile that contained the output of the ls /bin command. This is because stdout is the file myfile and not the terminal. Once the command completes, stdout returns to being the terminal. What this looks like graphically, we see in the figure below.



redirection with: *command 2> error.file*



Now, we want to see the contents of the file. We could simply say more myfile, but that wouldn't explain about redirection. Instead, we input

more <myfile

This tells the more command to take its standard input from the file myfile instead of from the keyboard or some other file. (Remember, even when stdin is the keyboard, it is still seen as a file.)

What about errors? As I mentioned, stderr *appears* to be going to the same place as stdout. A quick way of showing that it doesn't is by using output redirection and forcing an error. If wanted to list two directories and have the output go to a file, we run this command:

ls /bin /jimmo > /tmp/junk

We then get this message:

```
/jimmo not found >
```

However, if we look in **/tmp**, there is indeed a file called junk that contains the output of the ls **/bin** portion of the command. What happened here was that we redirected stdout into the file **/tmp/junk**. It did this with the listing of **/bin**. However, because there was no directory **/jimmo** (at least not on my system), we got the error **/jimmo** not found. In other words, stdout went into the file, but stderr still went to the screen.

If we want to get the output and any error messages to go to the same place, we can do that. Using the same example with ls, the command would be:

```
ls /bin /jimmo > /tmp/junk 2>&1
```

The new part of the command is 2>&1, which says that file descriptor 2 (stderr) should go to the same place as file descriptor 1 (stdout). By changing the command slightly

```
ls /bin /jimmo > /tmp/junk 2>/tmp/errors
```

we can tell the shell to send any errors someplace else. You will find quite often in shell scripts throughout the system that the file that error messages are sent to is /dev/null. This has the effect of ignoring the messages completely. They are neither displayed on the screen nor sent to a file.

Note that this command does not work as you would think:

```
ls /bin /jimmo 2>&1 > /tmp/junk
```

The reason is that we redirect stderr to the same place as stdout *before* we redirect stdout. So, stderr goes to the screen, but stdout goes to the file specified.

Redirection can also be combined with pipes like this:

```
sort < names | head
```

or

```
ps | grep sh > ps.save
```

In the first example, the standard input of the sort command is redirected to point to the file names. Its output is then passed to the pipe. The standard input of the head command (which takes the first ten lines) also comes from the pipe. This would be the same as the command

```
sort names | head
```

which we see here:

In the second example, the ps command (process status) is piped through grep and all of the output is redirected to the file ps.save.

If we want to redirect stderr, we can. The syntax is similar:

```
command 2> file
```

It's possible to input multiple commands on the same command line. This can be accomplished by using a semi-colon (;) between commands. I have used this on occasion to create command lines like this:

```
man bash | col –b > man.tmp; vi man.tmp; rm man.tmp
```

This command redirects the output of the man-page for bash into the file man.tmp. (The pipe through col -b is necessary because of the way the man-pages are formatted.) Next, we are brought into the vi editor with the file **man.tmp**. After I exit **vi**, the command continues and removes my temporary file **man.tmp**. (After about the third time of doing this, it got pretty monotonous, so I created a shell script to do this for me. I'll talk more about shell scripts later.)

4.9 Interpreting the Command

One question I had was, "In what order does everything get done?" We have shell variables to expand, maybe an alias or function to process, "real" commands, pipes and input/output redirection. There are a lot of things that the shell must consider when figuring out what to do and when.

For the most part, this is not very important. Commands do not get so complex that knowing the evaluation order becomes an issue. However, on a few occasions I have run into situations in which things did not behave as I thought they should. By evaluating the command myself as the shell would, it became clear what was happening. Let's take a look.

The first thing that gets done is that the shell figures out how many commands there are on the line. Remember, you can separate multiple commands on a single line with a semicolon. This process determines how many *tokens* there are on the command line. In this context, a token could be an entire command or it could be a control word such as "if." Here, too, the shell must deal with input/output redirection and pipes.

Once the shell determines how many tokens there are, it checks the syntax of each token. Should there be a syntax error, the shell will not try to start *any* of the commands. If the syntax is correct, it begins interpreting the tokens.

First, any alias you might have is expanded. Aliases are a way for some shells to allow you to define your own commands. If any token on the command line is actually an alias that you have defined, it is expanded before the shell proceeds. If it happens that an alias contains another alias, they are both expanded before continuing with the next step.

The next thing the shell checks for is functions. Like the functions in programming languages such as C, a shell function can be thought of as a small subprogram. Check the other sections for details on aliases and functions.

Once aliases and functions have all been completely expanded, the shell evaluates variables. Finally, it uses any wildcards to expand them to file names. This is done according to the rules we talked about previously.

After the shell has evaluated everything, it is *still* not ready to run the command. It first checks to see if the first token represents a command built into the shell or an external one. If it's not internal, the shell needs to go through the search path.

At this point, it sets up the redirection, including the pipes. These obviously must be ready before the command starts because the command may be getting its input from someplace other than the keyboard and may be sending it somewhere other than the screen. The figure below shows how the evaluation looks graphically.

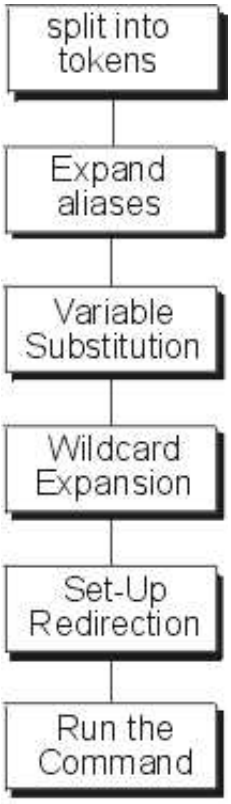


Image - Steps in interpreting command line input.

This is an oversimplification. Things happen in this order, though many more things occur in and around the steps than I have listed here. What I am attempting to describe is the general process that occurs when the shell is trying to interpret your command.

Once the shell has determined what each command is and each command is an executable *binary* program not a shell script, the shell makes a copy of itself using the fork system call. This copy is a child process of the shell. The *copy* then uses the exec system call to overwrite itself with the binary it wants to execute. Keep in mind that even though the child process is executing, the original shell is still in memory, waiting for the child to complete assuming the command was not started in the background with &.

If the program that needs to be executed is a shell script, the program that is created with fork and exec is another shell. This new shell starts reading the shell script and interprets it, one line at a time. This is why a syntax error in a shell script is not discovered when the script is started, but rather when the erroneous line is first encountered.

Understanding that a new process is created when you run a shell script helps to explain a very common misconception under UNIX. When you run a shell script and that script changes directories, your original shell knows nothing about the change. This confuses a lot of people

who are new to UNIX as they come from the DOS world, where changing the directory from within a batch file *does* change the original shell. This is because DOS does not have the same concept of a process as UNIX does.

Look at it this way: The sub-shell's environment has been changed because the current directory is different. However, this is *not* passed back to the parent. Like "real" parent-child relationships, only the children can inherit characteristics from their parent, not the other way around. Therefore, any changes to the environment, including directory changes, are not noticed by the parent. Again, this is *different* from the behavior of DOS .bat files.

You can get around this by either using aliases or shell functions assuming that your shell has them. Another way is to use the dot command in front of the shell script you want to execute. For example:

. myscript <--NOTICE THE DOT!

This script will be interpreted directly by the current shell, *without* forking a sub-shell. If the script makes changes to the environment, it is *this* shell's environment that is changed.

You can use this same functionality if you ever need to reset your environment. Normally, your environment is defined by the start-up files in your home directory. On occasion, things get a little confused maybe a variable is changed or removed and you need to reset things. You can you the dot command to do so. For example, with either sh or ksh, you can write it like this:

. $HOME/.profile <--NOTICE THE DOT!

Or, using a function of bash you can also write

. ~/.profile <--NOTICE THE DOT!

This uses the tilde ~, which I haven't mentioned yet. Under many shells, you can use the tilde as a shortcut to refer to a particular users home directory.

If you have csh, the command is issued like this:

source $HOME/.login <--NOTICE THE DOT!

Some shells keep track of your *last* directory in the OLDPWD environment variable. Whenever you change directories, the system saves your current directory in OLDPWD *before* it changes you to the new location.

You can use this by simply entering cd $OLDPWD. Because the variable $OLDPWD is expanded before the cd command is executed, you end up back in your previous directory. Although this has more characters than just popd, it's easier because the system keeps track of my position, current and previous, for you. Also, because it's a variable, I can access it in the same way that I can access other environment variables.

For example, if there were a file in your old directory that you wanted to move to your current one, you could do this by entering:

cp $OLDPWD/<file_name> ./

However, things are not as difficult as they seem. Typing in *cd $OLDPWD* is still a bit cumbersome. It is a lot less characters to type in popd -like in the csh. Why isn't there something like that in the ksh or bash? There is. In fact, it's much simpler. When I first found out about it, the adjective that first came to mind was "sweet." To change directories to your previous directory, simply type "cd -".

4.10 Different Kinds of Shells

The great-grandfather of all shells is /bin/sh, called simply sh or the Bourne Shell, named after its developer, Steven Bourne. When it was first introduced in the mid-1970s, this was almost a godsend as it allowed *interaction* with the operating system. This is the "standard" shell that you will find on every version in UNIX (at least all those I have seen). Although many changes have been made to UNIX, sh has remained basically unchanged.

All the capabilities of "the shell" I've talked about so far apply to **sh**. Anything I've talked about that sh can do, the others can do as well. So rather than going on about what sh can do (which I already did), I am going to talk about the characteristics of some other shells.

Later, I am going to talk about the C-Shell, which kind of throws a monkey wrench into this entire discussion. Although the concepts are much the same between the C-Shell and other shells, the constructs are often quite different. On the other hand, the other shells are extensions of the Bourne Shell, so the syntax and constructs are basically the same.

Be careful here. This is one case in which I have noticed that the various versions of Linux are different. Not every shell is in every version. Therefore, the shells I am going to talk about may not be in your distribution. Have no fear! If there is a feature that you really like, you can either take the source code from one of the other shells and add it or you can find the different shells all over the Internet, which is much easier.

Linux includes several different shells and we will get into the specific of many of them as we move along. In addition, many different shells are available as either public domain, shareware, or commercial products that you can install on Linux.

As I mentioned earlier, environment variables are set up for you as you are logging in or you can set them up later. Depending on the shell you use, the files used and where they are located is going to be different. Some variables are made available to everyone on the system and are accessed through a common file. Others reside in the user's home directory.

Normally, the files residing in a users home directory can be modified. However, a system administrator may wish to prevent users from doing so. Often, menus are set up in these files to either make things easier for the user or to prevent the user from getting to the command line. (Often users never need to get that far.) In other cases, environment variables that shouldn't be changed need to be set up for the user.

One convention I will be using here is how I refer to the different shells. Often, I will say "the bash" or just "bash" to refer to the Bourne-Again Shell as a concept and not the program **/bin/bash**. I will use "bash" to refer to the "Bourne Shell" as an abstract entity and not

specifically to the program **/bin/sh**.

Why the Bourne-Again Shell? Well, this shell is compatible with the Bourne Shell, but has many of the same features as both the Korn Shell (**ksh**) and C-Shell (**csh**). This is especially important to me as I flail violently when I don't have a Korn Shell.

Most of the issues I am going to address here are detailed in the appropriate man-pages and other documents. Why cover them here? Well, in keeping with one basic premise of this book, I want to show you the relationships involved. In addition, many of the things we are going to look at are not emphasized as much as they should be. Often, users will go for months or years without learning the magic that these shells can do.

Only one oddity really needs to be addressed: the behavior of the different shells when moving through symbolic links. As I mentioned before, symbolic links are simply pointers to files or directories elsewhere on the system. If you change directories into symbolic links, your "location" on the disk is different than what you might think. In some cases, the shell understands the distinction and hides from you the fact that you are somewhere else. This is where the problem lies.

Although the concept of a symbolic link exists in most versions of UNIX, it is a relatively new aspect. As a result, not all applications and programs behave in the same way. Let's take the directory /usr/spool as an example. Because it contains a lot of administrative information, it is a useful and commonly accessed directory. It is actually a symbolic link to **/var/spool**. If we are using ash as our shell, when we do a cd /usr/spool and then pwd, the system responds with: /var/spool. This is where we are "physically" located, despite the fact that we did a cd /usr/spool. If we do a cd .. (to move up to our parent directory), we are now located in /var. All this seems logical. This is also the behavior of csh and sh on some systems.

If we use bash, things are different. This time, when we do a cd /usr/spool and then pwd, the system responds with **/usr/spools**. This is where we are "logically". If we now do a cd .., we are located in /usr. Which of these is the "correct" behavior? Well, I would say *both*. There is nothing to define what the "correct" behavior is. Depending on your preference, either is correct. I tend to prefer the behavior of **ksh**. However, the behavior of ash is also valid.

4.11 Command Line Editing

When I first started working in tech support, I was given a csh and once I figured out all it could do, I enjoyed using it. I found the editing to be cumbersome from time to time, but it was better than retyping everything.

One of my co-workers, Kamal (of IguanaCam fame), was an avid proponent of the Korn Shell. Every time he wanted to show me something on my terminal, he would grumble when he forgot that I wasn't using ksh. Many times he tried to convert me, but learning a new shell wasn't high on my list of priorities.

I often complained to Kamal how cumbersome **vi** was (at least I thought so at the time). One day I asked him for some pointers on vi, because every time I saw him do something in vi, it looked like magic. He agreed with the one condition that I at least try the **ksh**. All he wanted to do was to show me one thing and if after that I still wanted to use the **csh**, that was my own decision. Not that he would stop grumbling, just that it was my own choice.

The one thing that Kamal showed me convinced me of the errors of my ways. Within a week, I had requested the system administrator to change my login shell to ksh.

What was that one thing? Kamal showed me how to configure the **ksh** to edit previous commands using the same syntax as the **vi** editor. I felt like the **csh** editing mechanism was like using a sledge-hammer to pound in a nail. It does what you want, but it is more work than you need.

Many different shells have a history mechanism. The history mechanism of both the ksh and bash has two major advantages over that of the csh. First, the information is actually saved to a file. This is either defined by the HISTFILE environment variable *before* the shell is invoked, or it defaults to .bash_history (for the bash) in your home directory. At any point you can edit this file and make changes to what the ksh perceives as your command history.

This could be useful if you knew you were going to be issuing the same commands every time you logged in and you didn't want to create aliases or functions. If you copied a saved version of this file (or any other text file) and named it **.sh_history**, you would immediately have access to this new history. (Rewriting history? I shudder at the ramifications.)

The second advantage is the ability to edit directly any of the lines in your **.bash_history** file from the command line. If your EDITOR environment variable is set to **vi** or you use the set -o vi command, you can edit previous commands using many of the standard vi editing commands.

To enter edit mode, press Esc. You can now scroll through the lines of your history file using the **vi** movement keys (h-j-k-l). Once you have found the line you are looking for, you can use other vi commands to delete, add, change, or whatever you need. If you press "v," you are brought into the full-screen version of vi (which I found out by accident). For more details, check out the vi or ksh man-page or the later section on vi.

Note that by default, the line editing commands are similar to the emacs editor. If vi-mode is activated, you can activate emacs-mode with set -o emacs". Turning either off can be done with +o emacs or +o vi.

One exciting thing that bash can do is extend the command line editing. There are a large number of key combinations to which you can get bash to react. You say that the key combinations are "bound" to certain actions. The command you use is bind. To see what keys are currently bound, use bind -v. This is useful for finding out all the different editing commands to which you can bind keys.

4.12 Functions

Most (all?) shells have the means of creating new "internal" commands. This is done by creating shell *functions*. Shell functions are just like those in a programming language. Sets of commands are grouped together and jointly called by a single name.

The format for functions is:

```
function_name() { first thing to do second thing to do third thing to do }
```

Functions can be defined anywhere, including from the command line. All you need to do is simply type in the lines one at a time, similar to the way shown above. The thing to bear in mind is that if you type a function from a command line, once you exit that shell, the function is gone.

Shell functions have the ability to accept arguments, just like commands. A simple example is a script that looks like this:

```
display() { echo $1 }

display Hello
```

```
The output would be>
```

Hello

Here we need to be careful. The variable $1 is the positional parameter from the call to the display function and not to the script. We can see this when we change the script to look like this:

```
display() { echo $1 } echo $1
```

display Hello

Lets call the script display.sh and start it like this:

```
display.sh Hi
```

The output would then look like this:

```
Hi>

Hello>
```

The first echo shows us the parameter from the command line and the second one shows us the parameter from the function.

4.13 Job Control

Job control is the ability to move processes between the foreground and background. This is very useful when you need to do several things at once, but only have one terminal. For example, let's say there are several files spread out across the system that we want to edit. Because we don't know where they are, we can't use full paths. Because they don't have anything common in their names, we can't use find. So we try ls -R > more.

After a minute or two, we find the first file we want to edit. We can then suspend this job by pressing Ctrl+Z. We then see something that looks like this:

```
[1]+ Stopped ls -R | more>
```

This means that the process has been stopped or suspended. One very important thing to note is that this process is not in the background as if we had put an "&" at the end. When a process is suspended, it stops doing anything, unlike a process in the background, which keeps on working.

Once the ls is in the background, we can run **vi**. When we are done with vi, we can bring the ls command back with the fg (foreground) command.

If we wanted to, we could have more than just one job suspended. I have never had the need to have more than two running like this, but I have gotten more than ten during tests. One thing that this showed me was the meaning of the plus sign (+). This is the "current" job, or the one we suspended last.

The number in brackets is the process entry in the job table, which is simply a table containing all of your jobs. Therefore, if we already had three jobs, the next time we suspended a job, the entry would look like this:

```
[4]+ Stopped ls -R >> output>
```

To look at the entire job table, we simply enter the command jobs, which might give us

```
[1] Stopped ls -R /usr >> output.usr[2] Stopped find / -print > output.find[3]- Stopped ls -R /var >> output.var[4]+ Stopped ls -R >> output.root>
```

The plus sign indicates the job that we suspended last. So this is the one that gets called if we run fg without a job number. In this case, it was Job 4. Note that there is a minus sign (-) right after Job 3. This was the second to last job that we suspended. Now, we bring Job 2 in the foreground with fg 2 and then immediately suspend it again with Ctrl+Z. The table now looks like this:

```
[1] Stopped ls -R /usr >> output[2]+ Stopped find / -print > output.find[3] Stopped ls -R /var >> output[4]- Stopped ls -R >> output
>
```

Note that Job 2 now has the plus sign following it and Job 4 has the minus sign.

In each of these cases, we suspended a job that was running in the foreground. If we had started a job and put it in the background from the command line, the table might have an entry that looked like this:

```
[3] Running ls -R /var >> output &>
```

This shows us that although we cannot see the process (because it is in the background), it is still running. We could call it to the foreground if we wanted by running fg 3. And, if we wanted, we could use the bg command to send one of the stopped jobs to the background. So

bg %1

would send Job 1 to the background just as if we had included & from the command line.

One nice thing is that we don't have to use just the job numbers when we are pulling something into the foreground. Because we know that we started a process with the find command, we can get it by using

fg %find

Actually, we could have used %f or anything else that was not ambiguous. In this case, we were looking for a process that started with the string we input. We could even look for strings anywhere within the command. To do this, the command might be

fg %?print

which would have given us the same command. Or, if we had tried

fg %?usr

we would have gotten Job 1 because it contains the string usr.

If we find that there is a job that we want to kill (stop completely), we can use the kill command. This works the same way, so kill %<nr> kills the job with number <nr>, kill %<string> kills the job starting with string, and so on.

Keep in mind that process takes up resources whether they are in the foreground or not. That is, background processes take up resources,too.

4.14 Aliases

What is an alias? It isn't the ability to call yourself Thaddeus Jones when your real name is Jedediah Curry. Instead, in a Linux-context it is the ability to use a different name for a command. In principle, personal aliases can be anything you want. They are special names that you define to accomplish tasks. They aren't shell scripts, as a shell script is external to your shell. To start up a shell script, type in its name. The system then starts a shell as a child process of your current shell to run the script.

Aliases, too, are started by typing them in. However, they are internal to the shell (provided your shell uses aliases). That is, they are internal to your shell process. Instead of starting a sub-shell, the shell executes the alias internally. This has the obvious advantage of being quicker, as there is no overhead of starting the new shell or searching the hard disk.

Another major advantage is the ability to create new commands. You can do this with shell scripts (which we will get into later), but the overhead of creating a new process does not make it worthwhile for simple tasks. Aliases can be created with multiple commands strung together. For example, I created an alias, t, that shows me the time. Although the date command does that, all I want to see is the time. So, I created an alias, t, like this:

```
alias t=`date | cut -c12-16`
```

When I type in t, I get the hours and minutes, just exactly the way I want.

Aliases can be defined in either the .profile, .login or the .cshrc, depending on your shell. However, as I described above, if you want them for all sub-shells, they need to go in **.cshrc**. If you are running a Bourne Shell, aliasing may be the first good reason to switch to another shell.

Be careful when creating aliases or functions so that you don't redefine existing commands. Either you end up forgetting the alias, or some other program uses the original program and fails because the alias gets called first. I once had a call from a customer with a system in which he could no longer install software. We tried replacing several programs on his system, but to no avail. Fortunately, he had another copy of the same product, but it, too, died with the same error. It didn't seem likely that it was bad media. At this point, I had been with him for almost an hour, so I decided to hand it off to someone else (often, a fresh perspective is all that is needed).

About an hour later, one of the other engineers came into my cubicle with the same problem. He couldn't come up with anything either, which relieved me, so he decided that he needed to research the issue. Well, he found the exact same message in the source code and it turned out that this message appeared when a command could not run the sort command. Ah, a corrupt sort binary. Nope! Not that easy. What else was there? As it turned out, the customer had created an alias called sort that he used to sort directories in a particular fashion. Because the Linux command couldn't work with this version of sort, it died.

Why use one over the other? Well, if there is something that can be done with a short shell script, then it can be done with a function. However, there are things that are difficult to do with an alias. One thing is making long, relatively complicated commands. Although you can do this with an alias, it is much simpler and easier to read if you do it with a function. I will go into some more detail about shell functions later in the section on shell scripting. You can also find more details in the bash man-page.

On some systems, you will find that they have already provide a number of aliases for you. To see what alias are currently configured, just run **alias** with no options and you might get something like this:

```
alias +='pushd .'
alias -='popd'
alias ..='cd ..'
alias ...='cd ../..'
alias beep='echo -en "\007"'
alias dir='ls -l'
```

```
alias l='ls -alF'
alias la='ls -la'
alias ll='ls -l'
alias ls='ls $LS_OPTIONS'
alias ls-l='ls -l'
alias md='mkdir -p'
alias o='less'
alias rd='rmdir'
alias rehash='hash -r'
alias unmount='echo "Error: Try the command: umount" 1>&2; false'
alias which='type -p'
alias you='yast2 online_update'
```

As you can see there are many different ways you can use aliases.

4.15 A Few More Constructs

There are a few more loop constructs that we ought to cover as you are likely to come across them in some of the system scripts. The first is for a for-loop and has the following syntax:

for var in word1 word2 ... do list of commands done

We might use this to list a set of pre-defined directories like this:

or dir in bin etc usr do ls -R $dir done

This script does a recursive listing three times. The first time through the loop, the variable dir is assigned the value bin, next etc, and finally usr.

You may also see that the do/done pair can be replaced by curly braces ({ }). So, the script above would look like this:

for dir in bin etc usr { ls -R $dir }

Next, we have while loops. This construct is used to repeat a loop while a given expression is true. Although you can use it by itself, as in

while ($VARIABLE=value)

I almost exclusively use it at the end of a pipe. For example:

cat filename | while read line do commands done

This sends the contents of the file *filename* through the pipe, which reads one line at a time. Each line is assigned to variable line. I can then process each line, one at a time. This is also the format that many of the system scripts use.

For those of you who have worked with UNIX shells before, you most certainly should have noticed that I have left out some constructs. Rather than turning this into a book on shell programming, I decided to show you the constructs that occur most often in the shell scripts on your system. I will get to others as we move along. The man-pages of each of the shells

provide more details.

4.16 The C-Shell

One of the first "new" shells to emerge was the csh or C-Shell. It is so named because much of the syntax it uses is very similar to the C programming language. This isn't to say that this shell is only for C programmers, or programmers in general. Rather, knowing C makes learning the syntax much easier. However, it isn't essential. (Note: The csh syntax is *similar to C*, so don't get your dander up if it's not *exactly* the same.)

The csh is normally the shell that users get on many UNIX systems. Every place I ever got a UNIX account, it was automatically assumed that I wanted csh as my shell. When I first started out with UNIX, that was true. In fact, this is true for most users. Because they don't know any other shells, the csh is a good place to start. You might actually have tcsh on your system, but the principles are the same as for csh.

As you login with csh as your shell, the system first looks in the global file /etc/cshrc. Here, the system administrator can define variables or actions that should be taken by every csh user. Next, the system reads two files in your home directory: .login and .cshrc. The .login file normally contains the variables you want to set and the actions you want to occur each time you log in.

In both of these files, setting variables have a syntax that is unique to the csh. This is one *major* difference between the csh and other shells. It is also a reason why it is not a good idea to give root csh as its default shell. The syntax for csh is

```
set variable_name=value
```

whereas for the other two, it is simply

```
variable=value
```

Because many of the system commands are Bourne scripts, executing them with csh ends up giving you a lot of syntax errors.

Once the system has processed your .login file, your .cshrc is processed. The .cshrc contains things that you want executed or configured every time you start a csh. At first, I wasn't clear with this concept. If you are logging in with the csh, don't you want to start a csh? Well, yes. However, the reverse is not true. Every time I start a csh, I don't want the system to behave as if I were logging in.

Let's take a look as this for a minute. One of the variables that gets set for you is the SHELL variable. This is the shell you use anytime you do a shell escape from a program. A shell escape is starting a shell as a subprocess of a program. An example of a program that allows a shell escape is vi.

When you do a shell escape, the system starts a shell as a new (child) process of whatever program you are running at the time. As we talked about earlier, once this shell exits, you are back to the original program. Because there is no default, the variable must be set to a shell. If

the variable is set to something else, you end up with an error message like the following from vi:

invalid SHELL value: <something_else>

where <something_else> is whatever your SHELL variable is defined as.

If you are running **csh** and your SHELL variable is set to **/bin/csh**, every time you do a shell escape, the shell you get is csh. If you have a .cshrc file in your home directory, not only is this started when you log in, but anytime you start a new csh. This can be useful if you want to access personal aliases from inside of subshells.

One advantage that the **csh** offered over the Bourne Shell is its ability to repeat, and even edit, previous commands. Newer shells also have this ability, but the mechanism is slightly different. Commands are stored in a shell "history list," which, by default, contains the last 20 commands. This is normally defined in your **.cshrc** file, or you can define them from the command line. The command set

history=100

would change the size of your history list to 100. However, keep in mind that everything you type at the command line is saved in the history file. Even if you mistype something, the shell tosses it into the history file.

What good is the history file? Well, the first thing is that by simply typing "history" with nothing else you get to see the contents of your history file. That way, if you can't remember the exact syntax of a command you typed five minutes ago, you can check your history file.

This is a nice trick, but it goes far beyond that. Each time you issue a command from the csh prompt, the system increments an internal counter that tells the shell how many commands have been input up to that point. By default, the csh often has the prompt set to be a number followed by a %. That number is the current command, which you can use to repeat those previous commands. This is done with an exclamation mark (!), followed by the command number as it appears in the shell history.

For example, if the last part of your shell history looked like this:

```
21 date
22 vi letter.john
23 ps
24 who
>
```

You could edit **letter.john** again by simply typing in !22. This repeats the command vi letter.john and adds this command to your history file. After you finish editing the file, this portion of the history file would look like

```
21 date
22 vi letter.john
23 ps
24 who
25 vi letter.john
>
```

Another neat trick that's built into this history mechanism is the ability to repeat commands without using the numbers. If you know that sometime within your history you edited a file using vi, you could edit it again by simply typing !vi. This searches backward though the history file until it finds the last time you used vi. If there were no other commands since the last time you used vi, you could also Enter !v.

To redo the last command you entered, you could do so simply by typing in !!.

This history mechanism can also be used to *edit* previously issued commands. Lets say that instead of typing vi letter.john, we had typed in vi letter.jonh. Maybe we know someone named jonh, but that's not who we meant to address this letter to. So, rather than typing in the whole command, we can edit it. The command we would issue would be !!:s/nh/hn/.

At first, this seems a little confusing. The first part, however, should be clear. The "!!" tells the system to repeat the previous command. The colon (:) tells the shell to expect some editing commands. The "s/nh/hn/" says to substitute for pattern nh the hn. (If you are familiar with vi or sed, you understand this. If not, we get into this syntax in the section on regular expressions and metacharacters.)

What would happen if we had edited a letter to john, done some other work and decided we wanted to edit a letter to chris instead. We could simply type !22:s/john/chris/. Granted, this is actually more keystrokes than if we had typed everything over again. However, you hopefully see the potential for this. Check out the csh man-page for many different tricks for editing previous commands.

In the default .cshrc are two aliases that I found quite useful. These are pushd and popd. These aliases are used to maintain a directory "stack". When you run pushd <dir_name>, your current directory is pushed onto (added to) the stack and you change the directory to <dir_name>. When you use popd, it pops (removes) the top of the directory stack and you change directories to it.

Like other kinds of stacks, this directory stack can be several layers deep. For example, lets say that we are currently in our home directory. A "pushd /bin" makes our current directory **/bin** with our home directory the top of the stack. A "pushd /etc" brings us to **/etc**. We do it one more time with **pushd /usr/bin**, and now we are in /usr/bin. The directory /usr/bin is now the top of the stack.

If we run **popd** (no argument), **/usr/bin** is popped from the stack and **/etc** is our new directory. Another **popd**, and **/bin** is popped, and we are now in /bin. One more pop brings me back to the home directory. (In all honesty, I have never used this to do anything more than to switch directories, then jump back to where I was. Even that is a neat trick.)

There is another useful trick built into the **csh** for changing directories. This is the concept of a directory path. Like the execution search path, the directory path is a set of values that are searched for matches. Rather than searching for commands to execute, the directory path is searched for directories to change into.

The way this works is by setting the cdpath variable. This is done like any other variable in csh. For example, if, as system administrator, we wanted to check up on the various spool directories, we could define cdpath like this:

```
set cdpath = /usr/spool
```

Then, we could enter

```
cd lp
```

If the shell can't find a subdirectory named lp, it looks in the cdpath variable. Because it is defined as /usr/spool and there is a /usr/spool/lp directory, we jump into /usr/spool/lp. From there, if we type

```
cd mail
```

we jump to /usr/spool/mail. We can also set this to be several directories, like this:

```
set cdpath = ( /usr/spool /usr/lib /etc )
```

In doing so, each of the three named directories will be searched.

The csh can also make guesses about where you might want to change directories. This is accomplished through the cdspell variable. This is a Boolean variable (true/false) that is set simply by typing

```
set cdspell
```

When set, the cdspell variable tells the csh that it should try to guess what is really meant when we misspell a directory name. For example, if we typed

```
cd /sur/bin (instead of /usr/bin)
```

the cdspell mechanism attempts to figure out what the correct spelling is. You are then prompted with the name that it guessed as being correct. By typing in anything other than "n" or "N," you are changing into this directory. There are limitations, however. Once it finds what it thinks is a match, it doesn't search any further.

For example, we have three directories, "a," "b," and "c." If we type "cd d," any of the three could be the one we want. The shell will make a guess and choose one, which may or may not be correct.

Note that you may not have the C-Shell on your system. Instead, you might have something called tcsh. The primary difference is that tcsh does command line completion and command line editing.

4.17 Commonly Used Utilities

There are hundreds of utilities and commands plus thousands of support files in a normal Linux installation. Very few people I have met know what they all do. As a matter of fact, I don't know anyone who knows what they all do. Some are obvious and we use them everyday, such as date. Others are not so obvious and I have never met anyone who has used them. Despite their overwhelming number and often cryptic names and even more cryptic options, many commands are very useful and powerful.

I have often encountered users, as well as system administrators, who combine many of these commands into something relatively complicated. The only real problem is that there is often a single command that would do all of this for them.

In this section, we are going to cover some of the more common commands. I am basing my choice on a couple of things. First, I am going to cover those commands that I personally use on a regular basis. These commands are those that I use to do things I need to do, or those that I use to help end users get done what they need to. Next, I will discuss the Linux system itself. There are dozens of scripts scattered all through the system that contain many of these commands. By talking about them here, you will be in a better position to understand existing scripts should you need to expand or troubleshoot them.

Because utilities are usually part of some larger process (such as installing a new hard disk or adding a new user), I am not going to talk about them here. I will get to the more common utilities as we move along. However, to whet your appetite, here is a list of programs used when working with files.

File Management

Command	Function
cd	change directory
chgrp	change the group of a file
chmod	change the permissions (mode) of a file
chown	change the owner of a file
cp	copy files
file	determine a file's contents
ls	list files or directories
ln	make a link to a file
mkdir	make a directory
mv	move (rename) a file
rm	remove a file
rmdir	remove a directory

File Manipulation

Command	Function
awk	pattern-matching, programming language
cat	display a file
cmp	compare two files
csplit	split a file
cut	display columns of a file
diff	find differences in two files
dircmp	compare two directories
find	find files
head	show the top portion of a file
more	display screenfuls of a file
perl	scripting language
sed	non-interactive text editor
sort	sort a file
tail	display bottom portion of a file
tr	translate chracters in a file
uniq	find unique or repeated lines in a file
xargs	process mutiple arguements

4.17.1 Examples of Commonly Used Utilities

Directory listings: `ls`

When doing a long listing of the directory you typically only want to see the date when that content of the file was last changed. This is the default behavior with the -l option. However there may be cases, where you want to see when other aspects of the file were changed such as the permissions. This is done using the -c option. If you leave off the -l option you may not see any dates at all. Instead the output is sorted by the time the file was changed.

Typically, when you do a simple ls of a directory, the only really useful piece of information you get is the filename. However, you could use the -p option to display a little bit more. For example, you might end up a something that looks like this:

```
Data/ letter.txt  script*  script2@ >
```

Here you can see that at the end of many of the files are a number of different symbols. The / forward slash indicates it is a directory, the @ says it is a symbolic link, and * asterisk says it is executable.

For many years, this is the extent of what you could do that is, differentiate file types by which symbol was displayed. However, if you have a somewhat newer system there is a lot more that you can do. If your terminal can display colors, it is possible to color-code output of ls. Newer versions of **ls** have the option --color=followed by when it should display colors. For example, never, always, or " auto". If set to auto, output will only be in color if you are connected to a terminal. If, for example, you used ls any script, it may not be useful to have the output displayed in color. In fact, it might mess up your script.

By default, a number of different file types and their associated colors are specified in the **/etc/DIR_COLORS** file. For example, dark red is used for executable files, light red is used for archives **tar**, **rpm** , dark blue is for directories, magenta is for image files and so forth. If you have a symbolic link that points nowhere i.e the target file does not exist the name will blink red. If you want to change the system defaults, copy **/etc/DIR_COLORS** to **.dir_colors** in your home directory.

Maybe you have a long list of files perhaps log files. You could do a long listing and check each one individually to find the most recent one. Instead, you could use the -t option to ls, which sorts the files by their modification time. That is when the data was last changed.

Removing files: rm

-i queries you before removing the file

-r recursively removes files

-f forces removal

The way files and directories are put together in Linux has some interesting side effects. In the section on files and filesystems, we talk about the fact that a directory is essentially just a list of the files and pointers to where the files are on the hard disk. If you were to remove the entry in the directory list, the system would not know where to find the file. That basically means you have removed the file. That means that even if you did not have write permission on a file, you could remove it if you had write permission on its parent directory. The same thing applies in reverse. If you did not have write permissions on the directory, you could not remove the file.

Copying files: cp

More than likely you'll sometimes need to make a copy of an existing file. This is done with the **cp** command, which typically takes two arguments, the source name and destination name. By default, the cp command does not work on directories. To do that, you would use the -r option which says to recursively copy the files.

Typically the **cp** command only takes two arguments, the source and destination of the copy. However, you can use more than two arguments if the last argument is a directory. That way you could copy multiple files into a directory with a single command.

Renaming and moving files: `mv`

To rename files, you use the **`mv`** command for move. The logic here is that you are moving the files from one name to another. You would also use this command if moving a file between directories. Theoretically one could say you are "renaming" the entire path to the file, therefore, "rename" might be a better command name. However, that's not the way it is. Like the **`cp`** command **`mv`** also takes the -i option to query you prior to overwriting an existing file.

Linking files: `ln`

Linux provides a couple of different ways of giving a file multiple names. One place this is frequently used is for scripts that either start a program or stop it, depending on the name. If you were to simply copy one file to another, and you needed to make a change, you would have to change both files. Instead, you would create a "link". Links are nothing more than multiple files, with different names, but referring to the exact same data on the hard disk.

There are actually two different kinds of links: "hard" and "soft". A hard link simply creates a new directory entry for that particular file. This new directory entry can be in the current directory, or any other directory on the same file system. This is an important aspect because Linux keeps track of files using a numbered table, with each number representing a single set of data on your hard disk. This number the inode will be unique for each file system. Therefore, you cannot have hard links between files on different file systems. We'll get into details of inodes in the section of the hard disk layout. You can actually see this number if you want by using the -i to the ls command. You might end up with output that looks like this.

```
184494 -rw-r--r--    2 root     root         2248 Aug 11 17:54 chuck
184494 -rw-r--r--    2 root     root         2248 Aug 11 17:54 jimmo
184502 -rw-r--r--    1 root     root          761 Aug 11 17:55 john
>
```

Look at the inode number associated with files jimmo and chuck; they are the same 184494. This means that the two files are linked together and therefore are the exact same file. If you were to change one file the other one would be changed as well.

To solve the limitation that links cannot cross filesystems, you would use a soft or "symbolic" link. Rather than creating a new directory entry, like in the case of a hard link, a symbolic link is actually a file that contains the pathname to the other file. Since a symbolic link contains the path, it can point to files on other file systems, including files on completely different machines for example, if you are using NFS.

The downside of symbolic links is that when you remove the target file for a symbolic link, your data is gone, even though the symbolic link still exists. To create either kind of link you use the ln command, adding the -s option when you want to create a symbolic link. The syntax is basically the same as the cp or mv command:

`ln [-s] source destination`

In this case "source" is the original file and "destination" is the new link.

In addition to being able to link files across file systems using symbolic links, symbolic links can be used to link directories. Creating links to directories is not possible with hard links.

Display the contents of files: `cat` `more` and `less`

You can display the contents of a file using the **`cat`** command. The syntax is vers simple:

```
cat filename
```

If the file is large it may scroll off the screen. In that case you won't be able to see it all at one time. In that case, you would probably use either the **`more`** or **`less`** commands. Both allow you to display the contents of a file, while **`less`** allows you to scroll forward and backward throughout the files. However, **`less`** is not found on every Unix operating system, so becoming familiar with **`more`** is useful.

One might think that cat isn't useful. However, it is often used to sent the contents of a file through another file using a pipe. For example:

```
cat filename | sed 's/James/Jim/g'
```

This would send the file through the **`sed`**, replacing all occurrences of "James" with "Jim".

Combining multiple files: `cat`

The **`cat`** command can also be used to combine multiple files. Here we need to consider to things. First, the cat command simply displays all of the files listed to standard output. So to display three files we might have this command:

```
cat filename1 filename2 filename3
```

In the section on pipes and redirection, we talked about being able to redirect standard output to a file using the greater-than symbol >. So combining these concepts we might end up with this:

```
cat filename1 filename2 filename3 > new_file
```

This sends the contents of the three files in the order given into the file "newfile". Note that if the file already exists, it will be overwritten. As we also discussed, you can also append to an existing file using two greater-than symbols >>.

Display the contents of files: `more`

Display the contents of files: `less`

4.18 Looking for Files

In the section on Interacting with the System we talked about using the ls command to look for files. There we had the example of looking in the sub-directory **`./letters/taxes`** for specific files. Using **`ls`**, command we might have something like this:

```
ls ./letters/*
```

What if the taxes directory contained a subdirectory for each year for the past five years, each of these contained a subdirectory for each month, each of these contained a subdirectory for federal, state, and local taxes, and each of these contained 10 letters?

If we knew that the letter we we're looking for was somewhere in the taxes subdirectory, the command

```
ls ./letters/taxes/*
```

would show us the sub-directories of taxes (federal, local, state), and it would show their contents. We could then look through this output for the file we were looking for.

What if the file we were looking for was five levels deeper? We could keep adding wildcards (*) until we reached the right directory, as in:

```
ls ./letters/taxes/*/*/*/*/*
```

This might work, but what happens if the files were six levels deeper. Well, we could add an extra wildcard. What if it were 10 levels deeper and we didn't know it? Well, we could fill the line with wildcards. Even if we had too many, we would still find the file we were looking for.

Fortunately for us, we don't have to type in 10 asterisks to get what we want. We can use the -R option to ls to do a recursive listing. The -R option also avoids the "argument list too long" error that we might get with wildcards. So, the solution here is to use the ls command like this:

```
ls -R ./letters/taxes | more
```

The problem is that we now have 1,800 files to look through. Piping them through more and looking for the right file will be very time consuming. If we knew that it was there, but we missed it on the first pass, we would have to run through the whole thing again.

The alternative is to have the more command search for the right file for you. Because the output is more than one screen, more will display the first screen and at the bottom display --More--. Here, we could type a slash (/) followed by the name of the file and press Enter. Now more will search through the output until it finds the name of the file. Now we know that the file exists.

The problem here is the output of the ls command. We can find out whether a file exists by this method, but we cannot really tell where it is. If you try this, you will see that more jumps to the spot in the output where the file is (if it is there). However, all we see is the file name, not what directory it is in. Actually, this problem exists even if we don't execute a search.

If you use more as the command and not the end of a pipe, instead of just seeing --More--, you will probably see something like

```
--More--(16%)>
```

This means that you have read 16 percent of the file.

However, we don't need to use **more** for that. Because we don't want to look at the entire output (just search for a particular file), we can use one of three commands that Linux provides to do pattern searching: **grep**, **egrep**, and **fgrep**. The names sound a little odd to the Linux beginner, but grep stands for global regular expression print. The other two are newer versions that do similar things. For example, egrep searches for patterns that are full regular expressions and fgrep searches for fixed strings and is a bit faster. We go into details about the grep command in the section on looking through files.

Let's assume that we are tax consultants and have 50 subdirectories, one for each client. Each subdirectory is further broken down by year and type of tax (state, local, federal, sales, etc.). A couple years ago, a client of ours bought a boat. We have a new client who also wants to buy a boat, and we need some information in that old file.

Because we know the name of the file, we can use grep to find it, like this:

```
ls -R ./letters/taxes | grep boat
```

If the file is called boats, boat.txt, boats.txt, or letter.boat, the grep will find it because grep is only looking for the pattern boat. Because that pattern exists in all four of those file names, all four would be potential matches.

The problem is that the file may not be called boat.txt, but rather Boat.txt. Remember, unlike DOS, UNIX is case-sensitive. Therefore, grep sees boat.txt and Boat.txt as different files. The solution here would be to tell grep to look for both.

Remember our discussion on regular expressions in the section on shell basics? Not only can we use regular expressions for file names, we can use them in the arguments to commands. The term regular expression is even part of grep's name. Using regular expressions, the command might look like this:

```
ls -R ./letters/taxes | grep [Bb]oat
```

This would now find both boat.txt and Boat.txt.

Some of you may see a problem with this as well. Not only does Linux see a difference between boat.txt and Boat.txt, but also between Boat.txt and BOAT.TXT. To catch all possibilities, we would have to have a command something like this:

```
ls -R ./letters/taxes | grep [Bb][Oo][Aa][Tt]
```

Although this is perfectly correct syntax and it will find the files, it does not matter what case the word "boat" is in, it is too much work. The programmers who developed grep realized that people would want to look for things regardless of what case they are in. Therefore, they built in the -i option, which simply says ignore the case. Therefore, the command

```
ls -R ./letters/taxes | grep -i boat
```

will not only find boats, boat.txt, boats.txt, and letter.boat, but it will also find Boat.txt and BOAT.TXT as well.

If you've been paying attention, you might have noticed something. Although the grep command will tell you about the existence of a file, it won't tell you where it is. This is just like piping it through **more**. The only difference is that we're filtering out something. Therefore, it still won't tell you the path.

Now, this isn't greps fault. It did what it was supposed to do. We told it to search for a particular pattern and it did. Also, it displayed that pattern for us. The problem is still the fact that the ls command is not displaying the full paths of the files, just their names.

Instead of ls, let's use a different command. Let's use find instead. Just as its name implies, find is used to find things. What it finds is files. If we change the command to look like this:

```
find ./letters/taxes -print | grep -i boat
```

This finds what we are looking for and gives us the paths as well.

Before we go on, let's look at the syntax of the find command. There are a lot of options and it does look foreboding, at first. We find it is easiest to think of it this way:

```
find <starting_where> <search_criteria> <do_something>
```

In this case, the "where" is **./letters/taxes**. Therefore, find starts its search in the **./letters/taxes** directory. Here, we have no search criteria; we simply tell it to do something. That something was to -print out what it finds. Because the files it finds all have a path relative to **./letters/taxes**, this is included in the output. Therefore, when we pipe it through grep, we get the path to the file we are looking for.

We also need to be careful because the find command we are using will also find directories named boat. This is because we did not specify any search criteria. If instead we wanted it just to look for regular files (which is often a good idea), we could change the command to look like this:

```
find ./letters/taxes -type f -print | grep -i boat
```

Here we see the option -type f as the search criteria. This will find all the files of type f for regular files. This could also be a d for directories, c for character special files, b for block special files, and so on. Check out the **find** man-page for other types that you can use.

Too complicated? Let's make things easier by avoiding grep. There are many different things that we can use as search criteria for find. Take a quick look at the man-page and you will see that you can search for a specific owner, groups, permissions, and even names. Instead of having grep do the search for us, let's save a step (and time) by having find do the search for us. The command would then look like this:

```
find ./letters./taxes –name boat –print
```

This will find any file named boat and list its respective path. The problem here is that it will only find the files named boat. It won't find the files boat.txt, boats.txt, or even Boat.

The nice thing is that find understands about regular expressions, so we could issue the command like this:

```
find ./letters./taxes –name '[Bb]oat' –print
```

(Note that we included the single quote (') to avoid the square brackets ([]) from being first interpreted by the shell.)

This command tells find to look for all files named both boat and Boat. However, this won't find BOAT. We are almost there.

We have two alternatives. One is to expand the find to include all possibilities, as in

```
find ./letters./taxes –name '[Bb][Oo][Aa][Tt]' –print
```

This will find all the files with any combination of those four letters and print them out. However, it won't find **boat.txt**. Therefore, we need to change it yet again. This time we have

```
find ./letters./taxes –name '[Bb][Oo][Aa][Tt]*' –print
```

Here we have passed the wildcard (*) to find to tell it took find anything that starts with "boat" (upper- or lowercase), followed by anything else. If we add an extra asterisk, as in

```
find ./letters./taxes –name '*[Bb][Oo][Aa][Tt]*' –print
```

we not only get **boat.txt**, but also **newboat.txt**, which the first example would have missed.

This works. Is there an easier way? Well, sort of. There is a way that is easier in the sense that there are less characters to type in. This is:

```
find ./letters/taxes –print | grep –i boat
```

Isn't this the same command that we issued before? Yes, it is. In this particular case, this combination of find and grep is the easier solution, because all we are looking for is the path to a specific file. However, these examples show you different options of find and different ways to use them. That's one of the nice things about Linux. There are many ways to get the same result.

Note that more recent versions of find do not require the -print options, as this is the default behavior.

Looking for files with specific names is only one use of find. However, if you look at the **find** man-page, you will see there are many other options you can use. One thing I frequently do is to look for files that are older than a specific age. For example, on many

systems, I don't want to hang on to log files that are older than six months. Here I could use the -mtime options like this:

```
find /usr/log/mylogs –mtime +180
```

Which says to find everything in the /usr/log/mylogs directory which is older than 180 days (Not exactly six months, but it works.) If I wanted, I could have used the -name option to have specified a particular file pattern:

```
find ./letters./taxes –name '*[Bb][Oo][Aa][Tt]*' –mtime +180
```

One problem with this is what determines how "old" a file is? The first answer for many people is that the age of a file is how long it has been since the file was created. Well, if I created a file two years ago, but added new data to it a minute ago, is it "older" than a file that I created yesterday, but have not changed since then? It really depends on what you are interested in. For log files, I would say that the time the data in that was last changed is more significant than when the file was created. Therefore, the -mtime is fitting as it bases its time on when the data was changed.

However, that's not always the case. Sometimes, you are interested in the last time the file was used, or accessed. This is when you would use the -atime option. This is helpful in find old files on your system that no one has used for a long time.

You could also use the -ctime option, which is based on when the files "status" was last changed. The status is changed when the permissions or file owner is changed. I have used this option in security contexts. For example, on some of our systems there are only a few places that contain files that should change at all. For example, **/var/log**. If I search on all files that were changed at all (content or status), it might give me an indication of improper activity on the system. I can run a script a couple of times an hour to show me the files that have changed within the last day. If anything shows up, I suspect a security problem (obviously ignoring files that are supposed to change.)

Three files that we specifically monitor are **/etc/passwd**, **/etc/group** and **/etc/shadow**. Interestingly enough, we *want* to have these files change once a month (/etc/shadow). This is our "proof" that the root password was changed as it should be at regular intervals. Note that we have other mechanisms to ensure that it was the root password that was changed and not simply changing something else in the file, but you get the idea. One place you see this mechanism at work is your /usr/lib/cron/run-crons file, which is started from /etc/crontab every 15 minutes.

One shortcoming of -mtime and the others is that it measures time in 24 hour increments starting from now. That means that you cannot find anything that was changed within the last hour, for example. For this newer versions of find have the -cmin, -amin and -mmin options, which measure times in minutes. So, to find all of the files changed within the last hour (i.e. last 60 minutes) we might have something like this:

```
find / –amin –60
```

In this example, the value was preceded with a minus sign (-), which means that we are looking for files with a value *less* than what we specified. In this case, we want values *less* than 60 minutes. In the example above, we use a plus-sign (+) before the value, which means values greater that what we specified. If you use neither one, then the time is *exactly* what you specified.

Along the same vein, are the options -newer, -anewer, -cnewer, which find files which are *newer* than the file specified.

Note also that these commands find *everything* in the specified path older or younger than what we specify. This includes files, directories, device nodes and so forth. Maybe this is what you want, but not always. Particularly if you are using the -exec option and what to search through each file you find, looking for "non-files" is not necessarily a good idea. To specify a file type, find provides you with the -type option. Among the possible file type are:

- b - block device
- c - character device
- d - directory
- p - named pipe (FIFO)
- f - regular file
- l - symbolic link
- s - socket

As you might expect, you can combine the -type option with the other options we discussed, to give you something like this:

```
find ./letters./taxes –name '*[Bb][Oo][Aa][Tt]*' –type f –mtime +180
```

The good news *and* the bad news at this point is that there are many, many more options you can use. For example. you can search for files based on their permissions (-perm), their owner (-users), their size (-size), and so forth. Many I occasionally use, some I have never used. See the find man-page for a complete list.

In addition, to the -execoption, there are a number of other ones that are applied to the files that are found (rather than used to restrict what files are found). Note that in most documentation, the options used to restrict the search are called *tests* and the options that perform an operation on the files are called *actions*. One very simple action is -ls, which does a listing of the files the same as using the -dils options to the ls command.

A variant of the -exec action is -ok. Rather than simply performing the action on each file, -ok with first ask you to confirm that it should do it. Pressing "Y" or "y" will run the command, pressing anything else will not.

With what we have discussed so far, you might run into a snag if there is more than one criterion you want to search on (i.e. more than one test). Find addresses that by allowing you to combine tests using either OR (-o -or) or AND (-a -and). Furthermore, you can negate the results of any tests (! -not). Let's say we wanted to find all of the HTML files that were not owned by the user jimmo. Our command might look like this:

```
find ./ -name *.html -not -user jimmo
```

This brings up an important issue. In the section on interpreting the command, we talk about the fact that the shell expands wildcards before passing them to the command to be executed. In this example, if there was a file in the current directory ending in .html, the shell would first expand the .html to that name *before* passing it to find. We therefore need to "protect" it before we pass it. This is done using single quotes and the resulting command might look like this:

```
find ./ -name '*.html' -not -user jimmo
```

For details on how quoting works, check out the section on quotes.

It is important to keep in mind the order in which things are evaluated. First, negation (-not !), followed by AND (and -a), then finally OR (-o -or). In order to force evaluation in a particular way, you can include expressions in parentheses. For example, if we wanted all of the files or directories owned by either root or bin, the command might look like this:

```
find / \( -type f -o -type d \) -a \( -user root -o -user bin
\)
```

The requires a little explanation. I said that you would use parentheses to group the tests together. However, they are preceded here with a back-slash. The reason is that the shell will see the parentheses and try to execute what is inside in a separate shell, which is not what we wanted.

4.19 Looking Through Files

In the section on looking for files, we talk about various methods for finding a particular file on your system. Let's assume for a moment that we were looking for a particular file, so we used the find command to look for a specific file name, but none of the commands we issued came up with a matching file. There was not a single match of any kind. This might mean that we removed the file. On the other hand, we might have named it yacht.txt or something similar. What can we do to find it?

We could jump through the same hoops for using various spelling and letter combinations, such as we did for yacht and boat. However, what if the customer had a canoe or a junk? Are we stuck with every possible word for boat? Yes, unless we know something about the file, even if that something is in the file.

The nice thing is that grep doesn't have to be the end of a pipe. One of the arguments can be the name of a file. If you want, you can use several files, because grep will take the first argument as the pattern it should look for. If we were to enter

```
grep [Bb]oat ./letters/taxes/*
```

we would search the contents of all the files in the directory **`./letters/taxes`** looking for the word Boat or boat.

If the file we were looking for happened to be in the directory **./letters/taxes**, then all we would need to do is run more on the file. If things are like the examples above, where we have dozens of directories to look through, this is impractical. So, we turn back to find.

One useful option to find is -exec. When a file is found, you use -exec to execute a command. We can therefore use find to find the files, then use -exec to run grep on them. Still, you might be asking yourself what good this is to you. Because you probably don't have dozens of files on your system related to taxes, let's use an example from files that you most probably have.

Let's find all the files in the **/etc** directory containing **/bin/sh**. This would be run as

```
find ./etc –exec grep /bin/sh {} \;
```

The curly braces ({ }) are substituted for the file found by the search, so the actual grep command would be something like

```
grep /bin/sh ./etc/filename
```

The "\;" is a flag saying that this is the end of the command.

What the find command does is search for all the files that match the specified criteria then run grep on the criteria, searching for the pattern [Bb]oat. (in this case there were no criteria, so it found them all)

Do you know what this tells us? It says that there is a file somewhere under the directory ./letters/taxes that contains either "boat" or "Boat." It doesn't tell me what the file name is because of the way the -exec is handled. Each file name is handed off one at a time, replacing the {}. It is as though we had entered individual lines for

```
grep [Bb]oat ./letters/taxes/file1
grep [Bb]oat ./letters/taxes/file2
grep [Bb]oat ./letters/taxes/file3
```

If we had entered

```
grep [Bb]oat ./letters/taxes/*
```

grep would have output the name of the file in front of each matching line it found. However, because each line is treated separately when using find, we don't see the file names. We could use the -l option to grep, but that would only give us the file name. That might be okay if there was one or two files. However, if a line in a file mentioned a "boat trip" or a "boat trailer," these might not be what we were looking for. If we used the -l option to grep, we wouldn't see the actual line. It's a catch-22.

To get what we need, we must introduce a new command: xargs. By using it as one end of a pipe, you can repeat the same command on different files without actually having to input the command multiple times.

In this case, we would get what we wanted by typing

```
find ./letters/taxes –print | xargs grep [Bb]oat
```

The first part is the same as we talked about earlier. The find command simply prints all the names it finds (all of them, in this case, because there were no search criteria) and passes them to xargs. Next, xargs processes them one at a time and creates commands using grep. However, unlike the -exec option to find, xargs will output the name of the file before each matching line.

Obviously, this example does not find those instances where the file we were looking for contained words like "yacht" or "canoe" instead of "boat." Unfortunately, the only way to catch all possibilities is to actually specify each one. So, that's what we might do. Rather than listing the different possible synonyms for boat, lets just take the three: boat, yacht, and canoe.

To do this, we need to run the find | xargs command three times. However, rather than typing in the command each time, we are going to take advantage of a useful aspect of the shell. In some instances, the shell knows when you want to continue with a command and gives you a secondary prompt. If you are running sh or ksh, then this is probably denoted as ">."

For example, if we typed

```
find ./letters/taxes –print |
```

the shell knows that the pipe (|) cannot be at the end of the line. It then gives us a > or ? prompt where we can continue typing

```
> xargs grep –i boat
```

The shell interprets these two lines as if we had typed them all on the same line. We can use this with a shell construct that lets us do loops. This is the for/in construct for sh and ksh, and the foreach construct in csh. It would look like this:

```
for j in boat ship yacht > do > find ./letters/taxes –print |
xargs grep –i $j > done
```

In this case, we are using the variable j, although we could have called it anything we wanted. When we put together quick little commands, we save ourselves a little typing by using single letter variables.

In the **bash/sh/ksh** example, we need to enclose the body of the loop inside the do-done pair. In the **csh** example, we need to include the end. In both cases, this little command we have written will loop through three times. Each time, the variable $j is replaced with one of the three words that we used. If we had thought up another dozen or so synonyms for boat, then we could have included them all. Remember also that the shell knows that the pipe (|) is not the end of the command, so this would work as well.

```
for j in boat ship yacht > do > find ./letters/taxes –print | >
xargs grep –i $j > done
```

Doing this from the command line has a drawback. If we want to use the same command again, we need to retype everything. However, using another trick, we can save the command. Remember that both the ksh and csh have history mechanisms to allow you to repeat and edit commands that you recently edited. However, what happens tomorrow when you want to run the command again? Granted, ksh has the **.sh_history** file, but what about sh and csh?

Why not save commands that we use often in a file that we have all the time? To do this, you would create a basic shell script, and we have a whole section just on that topic.

When looking through files, I am often confronted with the situation where I am not just looking for a single text, but possible multiple matches. Imagine a data file that contains a list of machines and their various characteristics, each on a separate line, which starts with that characteristic. For example:

```
Name: lin-db-01
IP: 192.168.22.10
Make: HP
CPU: 700
RAM: 512
Location: Room 3
>
```

All I want is the computer name, the IP address and the location, but not the others. I could do three individual greps, each with a different pattern. However, it would be difficult to make the association between the separate entries. That is, the first time I would have a list of machine's names, the second time a list of IP addresses and the third time a list of locations. I have written scripts before that handle this kind of situation, but in this case it would be easier to use a standard Linux command: **egrep**.

The egrep command is an extension of the basic grep command. (The 'e' stands for extended) In older versions of grep, you did not have the ability to use things like [:alpha:] to represent alphabetic characters, so extended grep was born. For details on representing characters like this check out the section in regular expressions.

One extension is the ability to have multiple search patterns that are checked simultaneously. That is, if any of the patterns are found, the line is displayed. So in the problem above we might have a command like this:

```
egrep "Name:|IP:|Location:" FILENAME
```

This would then list all of the respective lines *in order*, making association between name and the other values a piece of cake.

Another variant of grep is fgrep, which interprets the search pattern as a list of *fixed* strings, separated by newlines, any of which is to be matched. On some systems, grep, egrep and fgrep will all be a hard link to the same file.

I am often confronted with files where I want to filter out the "noise". That is, there is a lot of stuff in the files that I don't want to see. A common example, is looking through large shell scripts or configuration files when I am not sure exactly what I am looking for. I know when I

see it, but to simply **grep** for that term is impossible, as I am not sure what it is. Therefore, it would be nice to ingore things like comments and empty lines.

Once again we could use **egrep** as there are two expressions we want to match. However, this type we also use the -v option, which simply flips or inverts the meaning of the match. Let's say there was a start-up script that contained a variable you were looking for, You might have something like this:

```
egrep –v "^$|^#" /etc/rc.d/*|more
```

The first part of the expressions says to match on the beginning of the line (^) followed immediately by the end of the line ($), which turn out to be all empty lines. The second part of the expression says to match on all lines that start with the pound-sign (a comment). This ends up giving me all of the "interesting" lines in the file. The long option is easier to remember: --invert-match.

You may also run into a case where all you are interested in is which files contain a particular expression. This is where the -l option comes in (long version: --files-with-matches). For example, when I made some style changes to my web site I wanted to find all of the files that contained a table. This means the file had to contain the <TABLE> tag. Since this tag could contain some options, I was interested in all of the file which contained "<TABLE". This could be done like this:

```
grep –l '<TABLE' FILENAME
```

There is an important thing to note here. In the section on interpreting the command, we learn that the shell sets up file redirection before it tries to execute the command. If we don't include the less-than symbol in the single quotes, the shell will try to redirect the input from a file name "TABLE". See the section on quotes for details on this.

The -l option (long version: --files-with-matches) says to simply list the file names. Using the -L option (long version: --files-without-match) we have the same effect as using both the -v and the -l options. Note that in both cases, the lines containing the matches are *not* displayed, just the file name.

Another common option is -q(long: --quiet or --silent). This does not display anything. So, what's the use in that? Well, often, you simply want to know if a particular value exists in a file. Regardless of the options you use, grep will return 0 if any matches were found, and 1 if no matches were found. If you check the $? variable after running grep -q. If it is 0, you found a match. Check out the section on basic shell scripting for details on the $? and other variables.

Keep in mind that you do not need to use grep to read through files. Instead, it can be one end of a pipe. For example, I have a number of scripts that look through the process list to see if a particular process is running. If so, then I know all is well. However, if the process is not running, a message is sent to the administrators.

4.20 Basic Shell Scripting

In many of the other sections of the shell and utilities, we talked about a few programming constructs that you could use to create a quick script to perform some complex task. What if you wanted to repeat that task with different parameters each time? One simple solution is to is to re-type everything each time. Obviously not a happy thing.

We could use vi or some other text editor to create the file. However, we could take advantage of a characteristic of the cat command, which is normally used to output the contents of a file to the screen. You can also redirect the cat to another file.

If we wanted to combine the contents of a file, we could do something like this:

```
cat file1 file2 file3 >newfile
```

This would combine file1, file2, and file3 into newfile.

What happens if we leave the names of the source files out? In this instance, our command would look like this:

```
cat > newfile
```

Now, cat will take its input from the default input file, stdin. We can now type in lines, one at a time. When we are done, we tell cat to close the file by sending it an end-of-file character, Ctrl-D. So, to create the new command, we would issue the cat command as above and type in our command as the following:

```
for j in boat ship yacht do find ./letters/taxes –print | xargs grep –i $j done
```

<CTRL-D>

Note that here the secondary prompt, >, does not appear because it is cat that is reading our input and not the shell. We now have a file containing the five lines that we typed in that we can use as a shell script.

However, right now, all that we have is a file named newfile that contains five lines. We need to tell the system that it is a shell script that can be executed. Remember in our discussion on operating system basics that I said that a file's permissions need to be set to be able to execute the file. To change the permissions, we need a new command: chmod. (Read as "change mode" because we are changing the mode of the file.)

The chmod command is used to not only change access to a file, but also to tell the system that it should *try* to execute the command. I said "try" because the system would read that file, line-by-line, and would try to execute each line. If we typed in some garbage in a shell script, the system would try to execute each line and would probably report not found for every line.

To make a file execute, we need to give it execute permissions. To give everyone execution permissions, you use the chmod command like this:

```
chmod +x newfile
```

Now the file newfile has execute permissions, so, in a sense, it is executable. However, remember that I said the system would read each line. In order for a shell script to function correctly, it also needs to be readable by the person executing it. In order to read a file, you need to have read permission on that file. More than likely, *you* already have read permissions on the file since you created it. However, since we gave everyone execution permissions, let's give them all read permissions as well, like this:

```
chmod +r newfile
```

You now have a new command called newfile. This can be executed just like any the system provides for you. If that file resides in a directory somewhere in your path, all you need to do is type it in. Otherwise, (as we talked about before) you need to enter in the path as well. Keep in mind that the system does not need to be able to read binary programs. All it needs to be able to do is execute them. Now you have your first shell script and your first self-written UNIX command.

What happens if, after looking through all of the files, you don't find the one you are looking for. Maybe you were trying to be sophisticated and used "small aquatic vehicle" instead of boat. Now, six months later, you cannot remember what you called it. Looking through every file might take a long time. If only you could shorten the search a little. Because you remember that the letter you wrote was to the boat dealer, if you could remember the name of the dealer, you could find the letter.

The problem is that six months after you wrote it, you can no more remember the dealer's name than you can remember whether you called it a "small aquatic vehicle" or not. If you are like me, seeing the dealer's name will jog your memory. Therefore, if you could just look at the top portion of each letter, you might find what you are looking for. You can take advantage of the fact that the address is always at the top of the letter and use a command that is designed to look there. This is the head command, and we use it like this:

```
find ./letters/taxes –exec head {} \;
```

This will look at the first 10 (the default for head) lines of each of the files that it finds. If the addressee were not in the first ten lines, but rather in the first 20 lines, we could change the command to be

```
find ./letters/taxes –exec head –20 {} \;
```

The problem with this is that 20 lines is almost an entire screen. If you ran this, it would be comparable to running more on every file and hitting q to exit after it showed the first screen. Fortunately, we can add another command to restrict the output even further. This is the tail command, which is just the opposite of head as it shows you the bottom of a file. So, if we knew that the address resided on lines 15-20, we could run a command like this:

```
find ./letters/taxes –exec head –20 {} \; | tail –5
```

This command passes the first 20 lines of each file through the pipe, and then tail displays the last five lines. So you would get lines 15-20 of every file, right? Not quite.

The problem is that the shell sees these as two *tokens.* That is, two separate commands: find ./letters/taxes -exec head -20 { } \; and tail -5. All of the output of the find is sent to the pipe and it is the last five lines of this that tail shows. Therefore, if the find | head had found 100 files, we would not see the contents of the first 99 files!

The solution is to add two other shell constructs: while and read. The first command carries out a particular command (or set of commands) while some criteria are true. The read can read input either from the command line, or as part of a more complicated construction. So, using cat again to create a command as we did above, we could have something like this:

```
find ./letters/taxes –print | while read FILE do echo $FILE
head –20 $FILE | tail –5 done
```

In this example, the while and read work together. The while will continue so long as it can read something into the variable FILE; that is, so long as there is output coming from find. Here again, we also need to enclose the body of the loop within the do-done pair.

The first line of the loop simply echoes the name of the file so we can keep track of what file is being looked at. Once we find the correct name, we can use it as the search criteria for a find | grep command. This requires looking through each file twice. However, if all you need to see is the address, then this is a lot quicker than doing a more on every file.

If you have read through the other sections, you have a pretty good idea of how commands can be put together to do a wide variety of tasks. However, to create more complicated scripts, we need more than just a few commands. There are several shell constructs that you need to be familiar with to make complicated scripts. A couple (the while and for-in constructs) we already covered. However, there are several more that can be very useful in a wide range of circumstances.

There are several things we need to talk about before we jump into things. The first is the idea of arguments. Like binary programs, you can pass arguments to shell scripts and have them use these arguments as they work. For example, let's assume we have a script called myscript that takes three arguments. The first is the name of a directory, the second is a file name, and the third is a word to search for. The script will search for all files in the directory with any part of their name being the file name and then search in those files for the word specified. A very simple version of the script might look like this:

```
ls $1 | grep $2 | while read file do grep $3 ${1}/${file} done
```

The syntax is:

```
myscript directory file_name word
```

I discussed the while-do-done construct when I discussed different commands like find and grep. The one difference here is that we are sending the output of a command through a second pipe before we send it to the while.

This also brings up a new construct: ${1}/${file}. By enclosing a variable name inside of curly braces, we can combine variables. In this case, we take the name of the directory (${1}), and tack on a "/" for a directory separator, followed by the name of a file that grep found (${file}). This builds up the path name to the file.

When we run the program like this

```
myscript /home/jimmo trip boat
```

the three arguments /home/jimmo, trip, and boat are assigned to the positional parameters 1, 2, and 3, respectively. "Positional" because the number they are assigned is based on where they appear in the command. Because the positional parameters are shell variables, we need to refer to them with the leading dollar sign ($).

When the shell interprets the command, what is actually run is

```
ls /home/jimmo | grep trip | while read file do grep boat
/home/jimmo/${file} done
```

If we wanted, we could make the script a little more self-documenting by assigning the values of the positional parameters to variables. The new script might look like this:

```
DIR=$1 FILENAME=$2 WORD=$3 ls –1 $DIR | grep $FILENAME | while
read file do grep $WORD ${DIR}/${file} done
```

If we started the script again with the same arguments, first /home/jimmo would get assigned to the variable DIR, trip would get assigned to the variable FILENAME, and boat would get assigned to WORD. When the command was interpreted and run, it would still be evaluated the same way.

Being able to assign positional parameters to variables is useful for a couple of reasons. First is the issue of self-documenting code. In this example, the script is very small and because we know what the script is doing, we probably would not have made the assignments to the variables. However, if we had a larger script, then making the assignment is very valuable in terms of keeping track of things.

The next issue is that it might seem that many older shells can only reference 10 positional parameters. The first $0 refers to the script itself. What this can be used for, we'll get to in a minute. The others, $1-$9, refer to the arguments that are passed to the script. What happens if you have more than nine arguments? This is where the shift instructions come in. These move the arguments "down" in the positional parameters list.

For example, let's assume we changed the first part of the script like this:

```
DIR=$1 shift FILENAME=$1
```

On the first line, the value of positional parameter 1 is /home/jimmo and we assign it to the variable DIR. In the next line, the shift moves every positional parameter down. Because $0 remains unchanged, what was in $1 (/home/jimmo) drops out of the bottom. Now, the value of positional parameter 1 is trip, which is assigned to the variable FILENAME, and positional

parameter 2 (boat) is assigned to WORD.

If we had 10 arguments, the tenth would initially be unavailable to us. However, once we do the shift, what was the tenth argument is shifted down and becomes the ninth. It is now accessible through the positional parameter 9. If we had more than 10, there are a couple of ways to get access to them. First, we could issue enough shifts until the arguments all moved down far enough. Or, we could use the fact that shift can take as an argument the number of shifts it should do. Therefore, using

```
shift 9
```

makes the tenth argument positional parameter 1.

What about the other nine arguments? Are they gone? If you never assigned them to a variable, then yes, they are gone. However, if you assigned them to a variable *before* you made the shift, you still have access to their values. New versions of many shells (such as bash) can handle greater number of position parameters.

However, being able to shift positional parameters comes in handy in other instances, which brings up the issue of a new parameter: $*. This parameter refers to all the positional parameters (except $0). So, we had 10 positional parameters and did a shift 2 (ignoring whatever we did with the first two), the parameter $* would contain the value of the last eight arguments.

In our sample script above, if we wanted to search for a *phrase* and not just a single word, we could change the script to look like this:

```
DIR=$1 FILENAME=$2 shift 2 WORD=$* ls –1 $DIR | grep $FILENAME
| while read file do grep "$WORD" ${DIR}/${file} done
```

The first change was that after assigning positional parameters 1 and 2 to variables, we shifted twice, effectively removing the first two arguments. We then assigned the remaining argument to the variable WORD (WORD=$*). Because this could have been a phrase, we needed to enclose the variable in double-quotes ("$WORD"). Now we can search for phrases as well as single words. If we did not include the double quotes, the system would view our entry as individual arguments to grep.

Another useful parameter keeps track of the total number of parameters: $#. In the previous script, what would happen if we had only two arguments? The grep would fail because there would be nothing for it to search for. Therefore, it would be a good thing to keep track of the number of arguments.

We need to first introduce a new construct: if-then-fi. This is similar to the while-do-done construct, where the if-fi pair marks the end of the block (fi is simply if reversed). The difference is that instead of repeating the commands within the block while the specific condition is true, we do it only once, if the condition is true. In general, it looks like this:

```
if [ condition ] then do something fi
```

The conditions are all defined in the test man-page. They can be string comparisons, arithmetic comparisons, and even conditions where we test specific files, such as whether the files have write permission. Check out the test man-page for more examples.

Because we want to check the number of arguments passed to our script, we will do an arithmetic comparison. We can check if the values are equal, the first is less than the second, the second is less than the first, the first is greater than or equal to the second, and so on. In our case, we want to ensure that there are *at least* three arguments, because having more is valid if we are going to be searching for a phrase. Therefore, we want to compare the number of arguments and check if it is greater than or equal to 3. So, we might have something like this:

```
if [ $# -ge 3 ] then body_of_script fi
```

If we have only two arguments, the test inside the brackets is false, the if fails, and we do not enter the loop. Instead, the program simply exits silently. However, to me, this is not enough. We want to know what's going on, therefore, we use another construct: else. When this construct is used with the if-then-fi, we are saying that if the test evaluates to true, do one thing; otherwise, do something else. In our example program, we might have something like this:

```
if [ $# -ge 3 ]
then
   DIR=$1
   FILENAME=$2
   shift 2
   WORD=$*
   ls -1 $DIR | grep $FILENAME | while read file
   do
      grep "$WORD" ${DIR}/${file}
   done
else
   echo "Insufficient number of arguments"
fi
```

If we only put in two arguments, the if fails and the commands between the else and the fi are executed. To make the script a little more friendly, we usually tell the user what the correct syntax is; therefore, we might change the end of the script to look like this:

```
else echo "Insufficient number of arguments" echo "Usage: $0
<directory> <file_name> <word>" fi
```

The important part of this change is the use of the $0. As I mentioned a moment ago, this is used to refer to the program itself not just its name, but rather the way it was called. Had we hard-coded the line to look like this

```
echo "Usage: myscript <directory> <file_name> <word>"
```

then no matter how we started the script, the output would always be

```
Usage: myscript <directory> <file_name> <word>>
```

However, if we used $0 instead, we could start the program like this

```
/home/jimmo/bin/myscript /home/jimmo file
```

and the output would be

```
Usage: /home/jimmo/bin/myscript <directory> <file_name>
<word>>
```

On the other hand, if we started it like this

```
./bin/myscript /home/jimmo file
```

the output would be

```
Usage: ./bin/myscript <directory> <file_name> <word>>
```

One thing to keep in mind is that the else needs to be within the matching if-fi pair. The key here is the word *matching*. We could nest the if-then-else-fi several layers if we wanted. We just need to keep track of things. The key issues are that the ending fi matches the *last* fi and the else is enclosed within an if-fi pair. Here is how multiple sets might look:

```
if [ $condition1 = "TRUE" ] then if [ $condition2 = "TRUE" ]
then if [ $condition3 = "TRUE"] then echo "Conditions 1, 2 and
3 are true" else echo "Only Conditions 1 and 2 are true" fi
else echo "Only Condition 1 is true" fi else echo "No
conditions are true" fi
```

This doesn't take into account the possibility that condition1 is false, but that either condition2 or condition3 is true or that conditions 1 and 3 are true, but 2 is false. However, you should see how to construct nested conditional statements.

What if we had a single variable that could take on several values? Depending on the value that it acquired, the program would behave differently. This could be used as a menu, for example. Many system administrators build such a menu into their user's .profile (or .login) so that they never need to get to a shell. They simply input the number of the program that they want to run and away they go.

To do something like this, we need to introduce yet another construct: the case-esac pair. Like the if-fi pair, esac is the reverse of case. So to implement a menu, we might have something like this:

```
read choice case $choice in a) program1;; b) program2;; c)
program3;; *) echo "No such Option";; esac
```

If the value of choice that we input is a, b, or c, the appropriate program is started. The things to note are the in on the first line, the expected value that is followed by a closing parenthesis,

and that there are two semi-colons at the end of each block.

It is the closing parenthesis that indicates the end of the possibilities. If we wanted, we could have included other possibilities for the different options. In addition, because the double semi-colons mark the end of the block, we could have simply added another command before we got to the end of the block. For example, if we wanted our script to recognized either upper- or lowercase, we could change it to look like this:

```
read choice case $choice in a|A) program1 program2 program3;;
b|B) program2 program3;; c|C) program3;; *) echo "No such
Option";; esac
```

If necessary, we could also include a range of characters, as in

```
case $choice in [a-z] ) echo "Lowercase";; [A-Z] ) echo
"Uppercase";; [0-9] ) echo "Number";; esac
```

Now, whatever is called as the result of one of these choices does not have to be a UNIX command. Because each line is interpreted as if it were executed from the command line, we could have included anything as though we had executed the command from the command line. Provided they are known to the shell script, this also includes aliases, variables, and even shell functions.

A shell function behaves similarly to functions in other programming languages. It is a portion of the script that is set off from the rest of the program and is accessed through its name. These are the same as the functions we talked about in our discussion of shells. The only apparent difference is that functions created inside of a shell script will disappear when the shell exits. To prevent this, start the script with a . (dot).

For example, if we had a function inside a script called myscript, we would start it like this:

```
./myscript
```

One construct that I find very useful is select. With select, you can have a quick menuing system. It takes the form

```
select name in word1 word2 ... do list done
```

where each word is presented in a list and preceded by a number. Inputting that number sets the value of *name* to the word following that number. Confused? Lets look at an example. Assume we have a simple script that looks like this:

```
select var in date "ls -l" w exit do $var done
```

When we run this script, we get

```
1) date 2) ls -l 3) w 4) exit #?
```

The "#?" is whatever you have defined as the PS3 (third-level prompt) variable. Here, we have just left it at the default, but we could have set it to something else. For example:

```
export PS3="Enter choice: "
```

This would make the prompt more obvious, but you need to keep in mind that PS3 would be valid everywhere (assuming you didn't set it in the script).

In our example, when we input 1, we get the date. First, however, the word "date" is assigned to the variable "var." The single line within the list expands that variable and the line is executed. This gives us the date. If we were to input 2, the variable "var" would be assigned the word "ls -l" and we would get a long listing of the current directory (not where the script resides). If we input 4, when the line was executed, we would exit from the script.

In an example above we discussed briefly the special parameter $#. This is useful in scripts, as it keeps track of how many positional parameters there were and if there are not enough, we can report an error. Another parameter is $*, which contains all of the positional parameters. If you want to check the status of the last command you execute using the $? variable.

The process ID of the current shell is stored in the $$ parameter. Paired with this is the $! parameters, which is the process ID of the last command executed in the background.

One thing I sort of glossed over up to this point was the tests we made in the if-statements in the examples above. In one case we had this:

```
if [ $# -ge 3 ]
```

As we mentioned, this checks the number of command-line arguments ($#) and tests whether it is greater than or equal to 3. We could have written it like this:

```
if test $# -ge 3
```

With the exact same result. In the case of the bash, both [and test are built into the shell. However, with other shells, they are external commands (however they are typically together). If you look at either the test or bash man-page, you will see that there are many more things we can test. In our examples, we were either testing two strings or testing numerical values. We can also test many different conditions related to files, not just variables as we did in these examples.

It is common with many of the system scripts (i.e. those under /etc/rc.d) that they will first test if a particular file exists before proceeding. For example, a script might want to test if a configuration file exists. If so, it will read that file and use the values found in that file. Otherwise it will use default values. Sometimes these scripts will check whether a file exists and is executable. In both cases, a missing file could mean an error occurred or simply that a particular package was not installed.

4.21 Managing Scripts

A very common use of shell scripts that you write is to automate work. If you need to run the command by hand each time, it often defeats the intent of the automation. Therefore, it is also very common that commands are started from cron.

As Murphy's Law would have it, sometimes something will prevent the script from ending. However, each time cron starts, a new process is started, so you end up with dozens, if not hundreds of processes. Depending on the script, this could have a dramatic effect on the performance of your system. The solution is to make sure that the process can only start once, or if it is already running, you want to stop any previous instances.

So, the first question is how to figure out what processes are running, which is something we go into details about in another section. In short, you can use the ps command to see what processes are running:

```
ps aux | grep your_process_name | grep –v grep
```

Note that when you run this command, it will also appear in the process table. Since your process name is an argument to the **grep** command, grep ends up finding itself. The grep -v grep says to skip entries that containing the word "grep" which means you do not find the command you just issued. Assuming that the script is only started from **cron**, the only entries found will be those started by cron. If the return code of the command is 1, you know the process is running (or at least grep found a match.)

In your script, you check for the return code and if it is 1, the script exits, otherwise it does the intended work. Alternatively, you can make the assumption that if it is still running, there is a problem and you want to kill the process. You could use **ps**, **grep**, and **awk** to get the PID of that processes (or even multiple processes). However, it is a lot easier using the **pidof** command. You end up with something like this:

```
kill `pidof your_process_name`
```

The problem with that is the danger of killing a process that you hadn't intended. Therefore, you need to be sure that you kill the correct process. This is done by storing the PID of the process in a file and then checking for the existence of that file each time your scripts starts. If the file does not exist, it is assumed the process is not running, so the very next thing the script does is create the PID file. This could be done like this:

```
echo $$ > PID_file
```

This is already done by many system processes and typically these files are stored in **/var/run** and have the ending .pid. Therefore, the file containing the PID of your HTTP server is **/var/run/httpd.pid**. You can then be sure you get the right process with a command like this:

```
kill `cat PID_file`
```

Note that in your script, you should first check for the existence of the PID file before you try to kill the process. If the process does not exist, but the PID file does, maybe the process died. Depending on how long ago the process died, it is possible that the PID has been re-used and now belongs to a completely different process. So as an added safety measure you could verify that the PID belongs to the correct process.

To get some ideas on how existing scripts manage processes take a look at the init scripts in **/etc/rc.d**.

Details on if-then constructs in scripts can be found here.
Details on using back-quotes can be found here.
Details on file redirection can be found here.

4.22 Shell Odds and Ends

This section includes a few tidbits that I wasn't sure where to put.

You can get the shell to help you debug your script. If you place set -x in your script, each command with its corresponding arguments is printed as it is executed. If you want to just show a section of your script, include the set -x before that section, then another set +x at the end. The set +x turns off the output.

If you want, you can capture output into another file, without having it go to the screen. This is done using the fact that output generated as a result of the set -x is going to stderr and not stdout. If you redirect stdout somewhere, the output from set -x still goes to the screen. On the other hand, if you redirect stderr, stdout still goes to your screen. To redirect sterr to a file start, the script like this:

```
mscript 2>/tmp/output
```

This says to send file descriptor 2 (stderr) to the file **/tmp/output**.

To create a directory that is several levels deep, you do not have to change directories to the parent and then run mkdir from there. The mkdir command takes as an argument the path name of the directory you want to create. It doesn't matter if it is a subdirectory, relative path, or absolute path. The system will do that for you. Also, if you want to create several levels of directories, you don't have to make each parent directory before you make the subdirectories. Instead, you can use the -p option to mkdir, which will automatically create all the necessary directories.

For example, imagine that we want to create the subdirectory **./letters/personal/john**, but the subdirectory letters does not exist yet. This also means that the subdirectory personal doesn't exist, either. If we run **mkdir** like this:

```
mkdir -p ./letters/personal/john
```

then the system will create **./letters**, then **./letters/personal**, and then **./letters/personal/john**.

Assume that you want to remove a file that has multiple links; for example, assume that ls, lc, lx, lf, etc., are links to the same file. The system keeps track of how many names reference the file through the link count (more on this concept later). Such links are called hard links. If you remove one of them, the file still exists as there are other names that reference it. Only when we remove the last link (and with that, the link count goes to zero) will the file be removed.

There is also the issue of symbolic links. A symbolic link (also called a soft link) is nothing more than a path name that points to some other file, or even to some directory. It is not until the link is accessed that the path is translated into the "real" file. This has some interesting effects. For example, if we create a link like this

ln -s /home/jimmo/letter.john /home/jimmo/text/letter.john

you would see the symbolic link as something like this:

```
drw-r--r-- 1 jimmo support 29 Sep 15 10:06 letter.john-> /home/jimmo/letter.john>
```

Then,the file **/home/jimmo/text/letter.john** is a symbolic link to **/home/jimmo/letter.john**. Note that the link count on **/home/jimmo/letter.john** doesn't change, because the system sees these as two separate files. It is easier to think of the file **/home/jimmo/text/letter.john** as a text file that contains the path to **/home/jimmo/letter.john**. If we remove **/home/jimmo/letter.john**, **/home/jimmo/text/letter.john** will still exist. However, it will point to something that doesn't exist. Even if there are other hard links that point to the same file like **/home/jimmo/letter.john**, that doesn't matter. The symbolic link, **/home/jimmo/text/letter.john**, points to the *path* **/home/jimmo/letter.john**. Because the path no longer exists, the file can no longer be accessed via the symbolic link. It is also possible for you to create a symbolic link to a file that does *not* exist, as the system does not check until you access the file.

Another important aspect is that symbolic links can extend across file systems. A regular or hard link is nothing more than a different name for the same physical file and used the same inode number. Therefore it must be on the same filesystem. Symbolic links contain a path, so the destination can be on another filesystem (and in some cases on another machine). For more on inodes, see the section on filesystems.

The file command can be used to tell you the type of file. With DOS and Windows, it's fairly obvious by looking at the file's extension to determine the files type. For example, files ending in .exe are executables (programs), files ending in .txt are text files, and files ending in .doc are documents (usually from some word processor). However, a program in UNIX can just as easily have the ending .doc or .exe, or no ending at all.

The file command uses the file **/etc/magic** to make an assumption about the contents of a file. The file command reads the header (first part of the file) and uses the information in /etc/magic to make its guess. Executables of a specific type (a.out, ELF) all have the same basic format, so file can easily recognize them. However, there are certain similarities between C source code, shell scripts, and even text files that could confuse file.

For a list of some of the more commonly used commands, take a look here.

Chapter 5 Editing Files

Because my intent here is not to make you shell or awk programming experts, there are obviously things that we didn't have a chance to cover. However, I hope I have given you the basics to create your own tools and configure at least your shell environment the way you need or want it.

Like any tool or system, the way to get better is to practice. Therefore, my advice is that you play with the shell and programs on the system to get a better feeling for how they behave. By creating your own scripts, you will become more familiar with both vi and shell script syntax, which will help you to create your own tools and understand the behavior of the system scripts. As you learn more, you can add awk and sed components to your system to make some very powerful commands and utilities.

5.1 Vi

No one can force you to learn vi, just as no one can force you to do backups. However, in my opinion, doing both will make you a better administrator. There will come a time when having done regular backups will save your career. There may also come a time when knowing vi will save you the embarrassment of having to tell your client or boss that you can't accomplish a task because you need to edit a file and the only editor is the system default: vi.

On the other hand it is my favorite editor. In fact, most of my writing is done using vi. That includes both books and articles. I find it a lot easier than using a so-called wysiwyg editor as I generally don't care what the text is going to look like as my editors are going to change the appearance anyway. Therefore, whether I am writing on Linux, Solaris, or even Windows, I have the same, familiar editor.

Then there is the fact that the files edited with vi are portable to any word processor, regardless of the operating system. Plus it makes making global changes a whole lot easier.

5.1.1 vi Basics

The uses and benefits of any editor like vi are almost religious. Often, the reasons people choose one editor over another are purely a matter of personal taste. Each offers its own advantages and functionality. Some versions of UNIX provide other editors, such as emacs. However, the nice thing about vi is that every dialect of UNIX has it. You can sit down at any UNIX system and edit a file. For this reason more than any other, I think it is worth learning.

One problem vi has is that can be very intimidating. I know, I didn't like it at first. I frequently get into discussions with people who have spent less than 10 minutes using it and then have ranted about how terrible it was. Often, I then saw them spending hours trying to find a free or relatively cheap add-on so they didn't have to learn vi. The problem with that approach is that if they has spent as much time learning vi as they did trying to find an alternative, they actually could have become quite proficient with vi.

There is more to vi than just its availability on different UNIX systems. To me, vi is magic. Once you get over the initial intimidation, you will see that there is a logical order to the way the commands are laid out and fit together. Things fit together in a pattern that is easy to remember. So, as we get into it, let me tempt you a little.

Among the "magical" things vi can do:

- Automatically correct words that you misspell often
- Accept user-created vi commands
- Insert the output of UNIX commands into the file you are editing
- Automatically indent each line
- Shift sets of lines left or right
- Check for pairs of {}, and [] great for programmers
- Automatically wrap around at the end of a line
- Cut and paste between documents

I am not going to mention every single vi command. Instead, I am going to show you a few and how they fit together. At the end of this section, there is a table containing the various commands you can use inside vi. You can then apply the relationships to the commands I don't mention.

To see what is happening when you enter commands, first find a file that you can poke around in. Make a copy of the termcap file /etc/termcap in a temporary directory and then edit it cd /tmp; cp /etc/termcap . ; vi termcap. The termcap file contains a list of the capabilities of various terminals. It is usually quite large and gives you a lot of things to play with in vi.

Before we can jump into the more advanced features of vi, I need to cover some of the basics. Not command basics, but rather some behavioral basics. In vi, there are two modes: command mode and input mode. While you are in command mode, every keystroke is considered part of a command. This is where you normally start when you first invoke vi. The reverse is also true. While in input mode, everything is considered input.

Well, that isn't entirely true and we'll talk about that in a minute. However, just remember that there are these two modes. If you are in command mode, you go into input mode using a command to get you there, such as append or insert I'll talk about these in a moment. If you want to go from input mode to command mode, press Esc.

When vi starts, it goes into full-screen mode assuming your terminal is set up correctly and it essentially clears the screen see the following image. If we start the command as **`vi search`**, at the bottom of the screen, you see

```
"search" [New File]>
```

. Your cursor is at the top left-hand corner of the screen, and there is a column of tildes ~ down the left side to indicate that these lines are nonexistent.

In the image below we see a vi session started from a terminal window running under X-Windows. This is essentially the same thing you will see when starting vi from any command line.

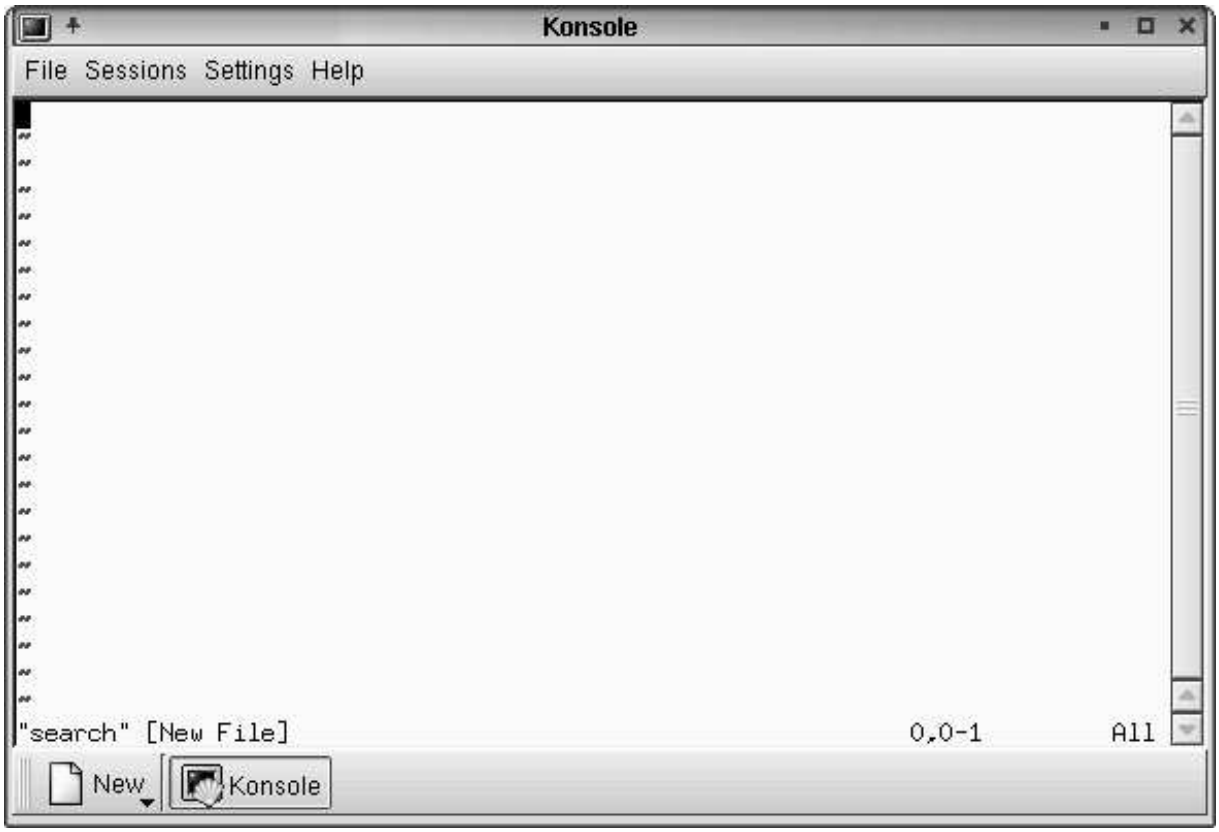


Image - Main vi window.

As with most text editors or word processors, vi gives you the ability to save the file you are editing without stopping the program. To issue the necessary command we first input a colon : when in command mode. When then press w for write and the press the entry key. This might look like the following figure:

Image - Writing a file in vi.

After you press the enter key, you end up with something like the following image:

Image - Writing a file in vi.

If you are editing a file that already existing and try to save it like this, you may get an error message that says the file is read only. You will also get this message, when trying to save a

file from "view", which is the "read-only" version of vi. To force the file to be written, you follow the w with an exclamation mark. :w!

The ex-mode or command mode also allows you to do many other things with the file itself. Among them are

- :q to quit the file :q! if the file has been changed and you don't want to save the changes
- :wq to write the file and quit
- :e to edit a new file or even the same file
- :r to read in a new file starting at the current location

5.1.2 Changing Text in vi

In addition to "standard" editing, there are a several special editing commands. Pressing dd will delete the entire line you are on; 5dd would then delete five complete lines. To open up a line for editing, we press o to open the line after the line you are currently on and O for the line before. Use x to delete the character (including numbers) that the cursor is on.

When we want to move something we just deleted, we put the cursor on the spot where we want it. Then press either p to put that text after the current cursor position or P to put it before the current position. A nice trick that I always use to swap characters is xp. The x deletes the character you are on and the p immediately inserts it. The result is that you swap characters. So if I had typed the word "into" as "inot," I would place the cursor on the "o" and type xp, which would swap the "o" and the "t."

To repeat the edit we just did, be it deleting 18 lines or inputting "I love you," we could do so by pressing "." (dot) from command mode. In fact, any edit command can be repeated with the dot.

To make a change, press c followed by a movement command or number and movement command. For example, to change everything from where you are to the next word, press cw. To change everything from where you are to the end of the line, press C or c$. If you do that, then a dollar sign will appear, indicating how much you intend to change.

If we go back into command mode (press Esc) before we reach the dollar sign, then everything from the current position to the dollar sign is removed. When you think about this, it is actually logical. By pressing C, you tell vi that you want to change everything to the end of the line. When you press Enter, you are basically saying that you are done inputting text; however, the changes should continue to the end of the line, thereby deleting the rest of the line.

To undo the last edit, what would we press? Well, whats the first letter of the word "undo"? Keep in mind that pressing u will only undo the last change. For example, lets assume we enter the following:

o to open a new line and go into input mode

I love

Esc to go back to command mode

a to append from current location

you

Esc to return to command mode

The result of what we typed was to open a new line with the text "I love you." We see it as one change, but from the perspective of vi, two changes were made. First we entered "I love," then we entered "you." If we were to press u, only "you" would be removed. However, if u undoes that last change, what command do you think returns the line to its original state? What else: U. As you are making changes, vi keeps track of the original state of a line. When you press U, the line is returned to that original state.

If you want to replace all of the text on the current line, you could simply delete the line and insert a new one. However, you could also **replace** the existing line by using the R (for replace) command. This puts vi into replace mode and each character you type replaces the existing characters as you write.

5.1.3 Moving Around in vi

Most editing and movement commands are single letters and are almost always the first letter of what they do. For example, to insert text at your current cursor position, press i. To append text, press a. To move forward to the beginning of the next word, press w. To move back to the beginning of the previous word, press b.

The capital letter of each command has a similar behavior. Use I to insert at the beginning of a line. Use A to start the append from the end of the line. To move "real" words, use W to move forward and B to move back.

Real words are those terminated by whitespaces (space, tab, newline). Assume we wanted to move across the phrase 'static-free bag'. If we start on the 's', pressing 'w', will move me to the '-'. Pressing 'w' again, we move to the 'f' and then to the 'b'. If we are on the 's' and press 'W', we jump immediately to the 'b'. That is, to the next "real" word.

Moving in vi is also accomplished in other ways. Depending on your terminal type, you can use the traditional method of arrow keys to move within the file. If vi doesn't like your terminal type, you can use the keys h-j-k-l. If we want to move to the left we press 'h'. If you think about it, this make sense since 'h' is on the left end of these four characters. To move right, press l. Again, this makes sense as the 'l' is on the right end.

Movement up and down is not as intuitive. One of the two remaining characters (j and k) will move us up and the other will move us down. But which one moves in which direction? Unfortunately, I don't have a very sophisticated way of remembering. If you look at the two letters physically, maybe it helps. If you imagine a line running through the middle of these characters, then you see that j hangs down below that line. Therefore, use j to move down. On

the other hand, k sticks up above the middle, so we use k to move up. However, in most cases, the arrow keys will work, so you won't need to remember. But it is nice to know them, as you can then leave your fingers on the keyboard.

As I mentioned, some keyboard types will allow you to use the arrow keys. However, you might be surprised by their behavior in input mode. This is especially true if you are used to a word processor where the arrow and other movement keys are the same all the time. The problem lies in the fact that most keyboards actually send more than one character to indicate something like a left-arrow or page-up key. The first of these is normally an escape (Esc). When you press one of these characters in input mode, the Esc is interpreted as your wish to leave input mode.

If we want to move to the first character on a line, we press '0' (zero) or '^'. To move to the last character, press $. Now, these are not intuitive. However, if you think back to our discussion on regular expressions, you'll remember that ^ (caret) represents the beginning of a line and $ (dollar sign) represents the end of a line. Although, these two characters do not necessarily have an intuitive logic, they do fit in with other commands and programs that you find on a Linux system.

We can also take advantage of the fact that vi can count as well as combine movement with this ability to count. By pressing a number before the movement command, vi will behave as if we had pressed the movement key that many times. For example, 4w will move us forward four words or 6j will move us six lines down.

If we want to move to a particular line we input the number and G. So, to move to line 43, we would press 42G, kind of like 42-Go! If instead of G we press Enter, we would move ahead that many lines. For example, if we were on line 85, pressing 42 and Enter would put us on line 127. (No, you don't have to count lines; vi can display them for you, as we'll see in a minute.)

As you might have guessed, we can also use these commands in conjunction with the movement keys (all except Ctrl-u and Ctrl-d). So, to delete everything from your current location to line 83, we would input d83G. (Note that delete begins with d.) Or, to change everything from the current cursor position down 12 lines, we would input c12+ or press c12 Enter.

5.1.4 Searching in vi

If you are trying to find a particular text, you can get vi to do it for you. You tell vi that you want to enter a search pattern by pressing / (slash). This will bring you down to the bottom line of the screen where you will see your slash. You then can type in what you want to look for. When you press Enter, vi will start searching from your current location down toward the bottom of the file. If you use press ? instead of /, then vi will search from your string toward the top of the file.

If the search is successful, that is, the string is found, you are brought to that point in the text. If you decide that you want to search again, you have three choices. You can press ? or / and input the search string again; press n, which is the first letter of the word "next"; or simply

press ? or / with no text following it for vi to continue the search in the applicable direction. If you wanted to find the next string that matches but in the opposite direction, what do you think the command would be? (Hint: the capital form of the "next" command.)

Once you have found what you are looking for, you can edit the text all you want and then continue searching. This is because the search string you entered is kept in a buffer. So, when you press /, ?, n, or N, the system remembers what you were looking for.

You can also include movement commands in these searches. First, you enclose the search pattern with the character used to search (/ or ?), then add the movement command. For example, if you wanted to search backward for the phrase "hard disk" and then move up a line, you would enter ?hard disk?-. If you wanted to search forward for the phrase "operating system" and then move down three lines, you would enter /operating system/+3.

All this time, we have been referring to the text patterns as search strings. As you just saw, you can actually enter phrases. In fact, you can use any regular expression you want when searching for patterns. For example, if you wanted to search for the pattern "Linux," but only when it appears at the beginning of a line, you would enter /^Linux. If you wanted to search for it at the end of the line, you would enter /Linux$.

You can also do more complicated searches such as /^new [Bb][Oo][Aa][Tt], which will search for the word "new" at the beginning of a line, followed by the word "boat" with each letter in either case.

No good text editor would be complete without the ability to not only search for text but to replace it as well. One way of doing this is to search for a pattern and then edit the text. Obviously, this starts to get annoying after the second or third instance of the pattern you want to replace. Instead, you could combine several of the tools you have learned so far.

For example, lets say that everywhere in the text you wanted to replace "Unix" with "UNIX." First, do a search on Unix with /Unix, tell vi that you want to change that word with cw, then input UNIX. Now, search for the pattern again with /, and simply press . (dot). Remember that the dot command repeats your last command. Now do the search and press the dot command again.

Actually, this technique is good if you have a pattern that you want to replace, but not every time it appears. Instead, you want to replace the pattern selectively. You can just press n (or whatever) to continue the search without carrying out the replacement.

What if you know that you want to replace every instance of a pattern with something else? Are you destined to search and replace all 50 occurrences? Of course not. Silly you. There is another way.

Here I introduce what is referred to as escape or ex-mode, because the commands you enter are the same as in the ex editor. To get to ex-mode, press : (colon). As with searches, you are brought down to the bottom of the screen. This time you see the : (colon). The syntax is

: <scope> <command>

An example of this would be:

:45,100s/Unix/UNIX/

This tells vi the scope is lines 45 *through* 100. The command is s/Unix/UNIX/, which says you want to substitute (s) the first pattern (Unix) with the second pattern (UNIX). Normally in English, we would say "substitute UNIX for Unix." However, the order here is in keeping with the UNIX pattern of source first, then destination (or, what it was is first, and what it will become is second, like mv source destination).

Note that this only replaces the first occurrence on each line. To get all occurrences, we must include g for global at the end of each line, like this:

:45,100s/Unix/UNIX/g

A problem arises if you want to modify only some of the occurrences. In this instance, you could add the modifier c for confirm. The command would then look like this:

:45,100s/Unix/UNIX/gc

This causes vi to ask for confirmation before it makes the change.

If you wanted to search and replace on every line in the file, you could specify every line, such as :1,48., assuming there were 48 lines in the file. (By the way, use Ctrl-g to find out what line you are on and how many lines there are in the file.) Instead of checking how many lines there are each time, you can simply use the special character $ to indicate the end of the file. (Yes, $ also means the end of the line, but in this context, it means the end of the file.) So, the scope of the command would look like :1,$.

Once again, the developers of vi made life easy for you. They realized that making changes throughout a file is something that is probably done a lot. They included a special character to mean the entire file: %. Therefore, the command is written as % = 1,$.

Here again, the search patterns can be regular expressions. For example, if we wanted to replace every occurrence of "boat" (in either case) with the word "ship," the command would look like this:

:%s/[Bb][Oo][Aa][Tt]/ship/g

As with regular expressions in other cases, you can use the asterisk (*) to mean any number of the preceding characters or a period (.) to mean any single character. So, if you wanted to look for the word "boat" (again, in either case), but only when it was at the beginning of a line and only if it were preceded by at least one dash, the command would look like this:

:%s/^--*[Bb][Oo][Aa][Tt]/ship/g

The reason you have two dashes is that the search criteria specified *at least* one dash. Because the asterisk can be *any* number, including zero, you must consider the case where it would mean zero. That is, where the word "boat" was at the beginning of a line and there were no

spaces. If you didn't care what the character was as long as there was at least one, you could use the fact that in a search context, a dot means any single character. The command would look like this:

```
:%s/^..*[Bb][Oo][Aa][Tt]/ship/g
```

5.1.5 vi Buffers

Remember when we first starting talking about searching, I mentioned that the expression you were looking for was held in a buffer. Also, whatever was matched by /[Bb][Oo][Aa][Tt] can be held in a buffer. We can then use that buffer as part of the replacement expression. For example, if we wanted to replace every occurrence of "UNIX" with "Linux," we could do it like this:

```
:%s/UNIX/Linux/g
```

The scope of this command is defined by the %, the shortcut way of referring to the entire text. Or, you could first save "UNIX" into a buffer, then use it in the replacement expression. To enclose something in a buffer, we enclose it within a matching pair of back slashes \(and \) to define the extent of a buffer. You can even have multiple pairs that define the extent of multiple buffers. These are reference by \#, where # is the number of the buffer.

In this example

```
:%s/\(UNIX\)/Linux \1/g
```

the text "UNIX," is placed into the first buffer. You then reference this buffer with \1 to say to vi to plug in the contents of the first buffer. Because the entire search pattern is the same as the pattern buffer, you could also have written it like this

```
:%s/\(UNIX\)/Linux &/g
```

in which the ampersand represents the entire search pattern.

This obviously doesn't save much typing. In fact, in this example, it requires more typing to save "UNIX" into the buffer and then use it. However, if what you wanted to save was longer, you would save time. You also save time if you want to use the buffer twice. For example, assume you have a file with a list of other files, some of them C language source files. All of them end in .c. You now want to change just the names of the C files so that the ending is "old" instead of .c. To do this, insert mv at the beginning of each line as well as produce two copies of the file name: one with .c and one with .old. You could do it like this:

```
:%s/^\(.*\)\.c/mv \1.c \1.old/g
```

In English, this line says:

- For every line (%)
- substitute (s)
- for the pattern starting at the beginning of the line (^), consisting of any number of characters (\(.*\)) (placing this pattern into buffer #1) followed by .c

- and use the pattern mv, followed by the contents of buffer #1 (\1), followed by a .c, which is again followed by the contents of buffer #1, (\1) followed by .old
- and do this for every line (g), (i.e., globally)

Now each line is of the form

```
mv file.c file.old >
```

Note the slash preceeding the dot in the expression "\.c". The slash "protects" the dot from being interpreted as the metacharacter for "any character". Instead, you want to search for a literal dot, so you need to protect it.

We can now change the permissions to make this a shell script and execute it. We would then move all the files as described above.

Using numbers like this is useful if there is more that one search pattern that you want to process. For example, assume that we have a three-column table for which we want to change the order of the columns. For simplicity's sake, lets also assume that each column is separated by a space so as not to make the search pattern too complicated.

Before we start, we need to introduce a new concept to vi, but one that you have seen before: [ ]. Like the shell, the square bracket pair ([ ]) of vi is used to limit sets of characters. Inside of the brackets, the caret (^) takes on a new meaning. Rather than indicating the beginning of a line, here it negates the character we are searching for. So we could type

```
%s/\([^  ]*\)  \([^  ]*\)
\([^  ]*\)/\3 \1 \2/g
```

Here we have three regular expressions, all referring to the same thing: \([^]*\). As we discussed above, the slash pair \(and \) delimits each of the buffers, so everything inside is the search pattern. Here, we are searching for [^]*, which is any number of matches to the set enclosed within the brackets. Because the brackets limit a set, the set is ^, followed by a space. Because the ^ indicates negation, we are placing any number of characters that is *not* a space into the buffer. In the replacement pattern, we are telling vi to print pattern3, a space, pattern1, another space, then pattern2.

In the first two instances, we followed the pattern with a space. As a result, those spaces were not saved into any of the buffers. We did this because we may have wanted to define our column separator differently. Here we just used another space.

I have often had occasion to want to use the pattern buffers more than once. Because they are not cleared after each use, you can use them as many times as you want. Using the example above, if we change it to

```
%s/\([^ ]*\) \([^ ]*\) \([^ ]*\)/\3 \1 \2 \1/g
```

we would get pattern3, then pattern1, then pattern2, and at the end, pattern1 again.

Believe it or not, there are still more buffers. In fact, there are dozens that we haven't touched on. The first set is the numbered buffers, which are numbered 1-9. These are used when we delete text and they behave like a stack. That is, the first time we delete something, say a

word, it is placed in buffer number 1. We next delete a line that is placed in buffer 1 and the word that was in buffer 1 is placed in buffer 2. Once all the numbered buffers all full, any new deletions push the oldest ones out the bottom of the stack and are no longer available.

To access these buffers, we first tell vi that we want to use one of the buffers by pressing the double-quote ("). Next, we specify then the number of the buffer, say 6, then we type either p or P to put it, as in "6p. When you delete text and then do a put without specifying any buffer, it automatically comes from buffer 1.

There are some other buffers, in fact, 26 of them, that you can use by name. These are the named buffers. If you can't figure out what their names are, think about how many of them there are (26). With these buffers, we can intentionally and specifically place something into a particular buffer. First, we say which buffer we want by preceding its name with a double-quote ("); for example, "f. This says that we want to place some text in the named buffer f. Then, we place the data in the buffer, for example, by deleting an entire line with dd or by deleting two words with d2w. We can later put the contents of that buffer with "fp. Until we place something new in that buffer, it will contain what we originally deleted.

If you want to put something into a buffer without having to delete it, you can. You do this by "yanking it." To yank an entire line, you could done one of several things. First, there is yy. Next, Y. Then, you could use y, followed by a movement commands, as in y-4, which would yank the next four lines (including the current one), or y/expression, which would yank everything from your current position up to and including expression.

To place yanked data into a named buffer (rather than the default buffer, buffer number 1), it is the same procedure as when you delete. For example, to yank the next 12 lines into named buffer h, we would do "h12yy. Now those 12 lines are available to us. Keep in mind that we do not have to store full lines. Inputting "h12yw will put the next 12 words into buffer h.

Some of the more observant readers might have noticed that because there are 26 letters and each has both an upper- and lowercase, we could have 52 named buffers. Well, up to now, the uppercase letters did something different. If uppercase letters were used to designate different buffers, then the pattern would be compromised. Have no fear, it is.

Instead of being different buffers than their lowercase brethren, the uppercase letters are the *same* buffer. The difference is that yanking or deleting something into an uppercase buffer appends the contents rather that overwriting them.

You can also have vi keep track of up to 26 different places with the file you are editing. These functions are just like bookmarks in word processors. (Pop quiz: If there 26 of them, what are their names?)

To mark a spot, move to that place in the file, type m for mark (what else?), then a single back quote (‘), followed by the letter you want to use for this bookmark. To go back to that spot, press the back quote (‘), followed by the appropriate letter. So, to assign a bookmark q to a particular spot, you would enter ‘q. Keep in mind that reloading the current file or editing a new one makes you lose the bookmarks.

Note that with newer version of vi (particularly vim) you don't press the backquote to set the mark, just m followed by the appropriate letter.

5.1.6 vi Magic

I imagine that long before now, you have wondered how to turn on all that magic I said that vi could do. Okay, let's do it.

The first thing I want to talk about is abbreviations. You can tell vi that when you type in a specific set of characters it is supposed to automagically change it to something else. For example, we could have vi always change USA to United States of America. This is done with the abbr command.

To create a new abbreviation, you must get into ex-mode by pressing the colon (:) in command mode. Next, type in abbr, followed by what you want to type in, and what vi should change it to. For example:

```
:abbr USA United States of America
```

Note that the abbreviation cannot contain any spaces because vi interprets everything after the second word as being part of the expansion.

If we later decide we don't want that abbreviation anymore, we enter

```
:unabbr USA
```

Because it is likely that we will want to use the abbreviation USA, it is not a good idea to use an abbreviation that would normally occur, such as USA. It would be better, instead, to use an abbreviation that doesn't occur normally, like Usa. Keep in mind, that abbreviations only apply to complete words. Therefore, something like the name "Sousa" won't be translated to "SoUSA." In addition, when your abbreviation is followed by a space, Tab, Enter, or Esc, the change is made.

Lets take this one step further. What if we were always spelling "the" as "teh." We could then create an abbreviation

```
:abbr teh the
```

Every time we misspell "the" as "teh," vi would automatically correct it. If we had a whole list of words that we regularly misspelled and created similar abbreviations, then every time we entered one of these misspelled words, it would be replaced with the correctly spelled word. Wouldn't that be automatic spell correction?

If we ever want to "force" the spelling to be a particular way (that is, turn off the abbreviation momentarily), we simply follow the abbreviation with a Ctrl-V. This tells vi to ignore the special meaning of the following character. Because the next character is a white space, which would force the expansion of the abbreviation (which makes the white space special in this case), "turning off" the white space keeps the abbreviation from being expanded.

We can also use vi to re-map certain sequences. For example, I have created a command so that all I need to do to save a file is Ctrl-W (for write). If I want to save the file and quit, I enter Ctrl-X with the "map" command.

The most common maps that I have seen have used control sequences, because most of the other characters are already taken up. Therefore, we need to side-step a moment. First, we need to know how to access control characters from within vi. This is done in either command mode or input mode by first pressing Ctrl-V and then pressing the control character we want. So to get Ctrl-W, I would type Ctrl-V, then Ctrl-W. This would appear on the screen as ^W. This looks like two characters, but if you inserted it into a text and moved over it with the cursor, you would realize that vi sees it as only one character. Note that although I pressed the lowercase w, it will appear as uppercase on the screen.

So, to map Ctrl-W so that every time we press it, we write our current file to disk, the command would be

map ^W :w^M

This means that when we press Ctrl-W, vi interprets it as though we actually typed :w and pressed Enter (the Ctrl-M, ^M). The Enter at the end of the command is a good idea because you usually want the command to be executed right away. Otherwise, you would have to press Enter yourself.

Also keep in mind that this can be used with the function keys. Because I am accustomed to many Windows and DOS applications in which the F2 key means to save, I map F2 to Ctrl-V, then F2. It looks like this:

map ^[[N :w^M (The ^[[N is what the F2 key displays on the screen)

If we want, we can also use shifted function characters. Therefore, we can map Shift-F2 to something else. Or, for that matter, we can also use shifted and control function keys.

It has been my experience that, for the most part, if you use Shift and Ctrl with non-function keys, vi only sees Ctrl and not Shift. Also, Alt may not work because on the system console, Alt plus a function key tells the system to switch to multiscreens.

I try not to use the same key sequences that vi already does. First, it confuses me because I often forget that I remapped something. Second, the real vi commands are then inaccessible. However, if you are used to a different command set (that is, from a different editor), you can "program" vi to behave like that other editor.

Never define a mapping that contains its own name, as this ends up recursively expanding the abbreviation. The classic example is :map! n banana. Every time you typed in the word "banana," you'd get

```
bababababababababababababababababa...>
```

and depending on what version you were running, vi would catch the fact that this is an infinite translation and stop.

5.1.7 Command Output in vi

It often happens that we want the output of UNIX commands in the file we are editing. The sledgehammer approach is to run the command and redirect it to a file, then edit that file. If that file containing the commands output already exists, we can use the :r from ex-mode to read it in. But, what if it doesn't yet exist. For example, I often want the date in text files as a log of when I input things. This is done with a combination of the :r (for read) from ex-mode and a *shell-escape*.

A shell-escape is when we start from one program and jump out of it (escape) to a shell. Our original program is still running, but we are now working in a shell that is a child process of that program.

To do a shell-escape, we need to be in ex-mode. Next, press the exclamation mark (!) followed by the command. For example, to see what time it is, type :!date. We then get the date at the bottom of the screen with the message to press any key to continue. Note that this didn't change our original text; it just showed us the output of the date command.

To read in a command's output, we need to include the :r command, as in :r!date. Now, the output of the date is read into the file (it is *inserted* into the file). We could also have the output replace the current line by pressing ! twice, as in !!date. Note that we are brought down to the last line on the screen, where there is a single !.

If we want, we can also read in other commands. What is happening is that vi is seeing the output of the command as a file. Remember that :r <file_name> will read a file into the one we are editing. Why not read from the output of a file? With pipes and redirection, both stdin and stdout can be files.

We can also take this one step further. Imagine that we are editing a file containing a long list. We know that many lines are duplicated and we also want the list sorted. We could do :%!sort, which, if we remember from our earlier discussion, is a special symbol meaning all the lines in the file. These are then sent through the command on the other side of the !. Now we can type

```
:%!uniq
```

to remove all the duplicate lines.

Remember that this is a shell-escape. From the shell, we can combine multiple commands using pipes. We can do it here as well. So to save time, we could enter

```
:%!sort | uniq
```

which would sort all the lines and remove all duplicate lines. If we only wanted to sort a set of lines, we could do it like this

```
:45,112!sort
```

which would sort lines 45 through 112. We can take this one step further by either writing lines 45-112 to a new file with :45,112w file_name or reading in a whole file to replace lines 45-112 with :45,112r file_name.

5.1.8 More vi Magic

If we need to, we can also edit multiple files. This is done like this:

```
vi file1 file2 file3
```

Once we are editing, we can switch between files with :n for the next file and :p for the previous one. Keep in mind that the file names do not wrap around. In other words, if we keep pressing :n and get to file3, doing it again does not wrap around and bring me to file1. If we know the name of the file, we can jump directly there, with the ex-mode edit command, as in

```
:e file3
```

The ability to edit multiple files has another advantage. Do you remember those numbered and named buffers? They are assigned for a single instance of vi, not on a per-file basis. Therefore, you can delete or yank text from one file, switch to the next and then insert it. This is a crude but effective cut and paste mechanism between files.

You can specify line numbers to set your position within a file. If you switch to editing another file (using :n or :r), or reload an original file (using :rew!), the contents of the deletion buffers are preserved so that you can cut and paste between files. The contents of all buffers are lost, however, when you quit vi.

5.1.9 vi Odds and Ends

You will find as you work with vi that you will often use the same vi commands over and again. Here too, vi can help. Because the named buffers are simply sequences of characters, you can store commands in them for later use. For example, when editing a file in vi, I needed to mark new paragraphs in some way as my word processor normally sees all end-of-line characters as new paragraphs. Therefore, I created a command that entered a "para-marker" for me.

First, I created the command. To do this, I opened up a new line in my current document and typed in the following text:

```
Para
```

Had I typed this from command mode, it would have inserted the text "Para" at the beginning of the line. I next loaded it into a named buffer with "pdd, which deletes the line and loads it into buffer p. To execute it, I entered @p. The @ is what tells vi to execute the contents of the buffer.

Keep in mind that many commands, abbreviations, etc., are transitive. For example, when I want to add a new paragraph, I don’t write Para as the only characters on a line. Instead, I use something less common: {P}. I am certain that I will never have {P} at the beginning of a

line; however, there are contexts where I might have Para at the beginning of a line. Instead, I have an abbreviation, Para, that I translated to {P}.

Now, I can type in Para at the beginning of a line in input mode and it will be translated to {P}. When I execute the command I have in buffer p, it inserts Para, which is then translated to {P}.

So why don't I just have {P} in buffer p? Because the curly braces are one set of movement keys that I did not mention yet. The { moves you back to the beginning of the paragraph and } moves you forward. Because paragraphs are defined by vi as being separated by a blank line or delimited by nroff macros, I never use them nroff is an old UNIX text processing language. Because vi sees the brackets as something special in command mode, I need to use this transitivity.

If you are a C programmer, you can take advantage of a couple of nifty tricks of vi. The first is the ability to show you matching pairs of parentheses , square brackets [], and curly braces {}. In ex-mode :, type set showmatch. Afterward, every time you enter the closing paren'thesis, bracket, or brace, you are bounced back to its match. This is useful in checking whether or not you have the right number of each.

We can also jump back and forth between these pairs by using %. No matter where we are within a curly braces pair {}, pressing % once moves us to the first opening brace. Press % again and we are moved to its match the closing brace. We can also place the cursor on the closing brace and press % to move us to the opening brace.

If you are a programmer, you may like to indent blocks of code to make things more readable. Sometimes, changes within the code may make you want to shift blocks to the left or right to keep the spacing the same. To do this, use << two less-than signs to move the text one "shift-width" to the left, and >> two greater-than signs to move the text one "shift-width" to the right. A "shift-width" is defined in ex-mode with set shiftwidth=n, where n is some number. When you shift a line, it moves left or right n characters.

To shift multiple lines, input a number before you shift. For example, if you input 23>>, you shift the next 23 lines one shiftwidth to the right.

There are a lot of settings that can be used with vi to make life easier. These are done in ex-mode, using the set command. For example, use :set autoindent to have vi automatically indent. To get a listing of options which have been changed from their default, simply input ":set" and you get something like in the following image:

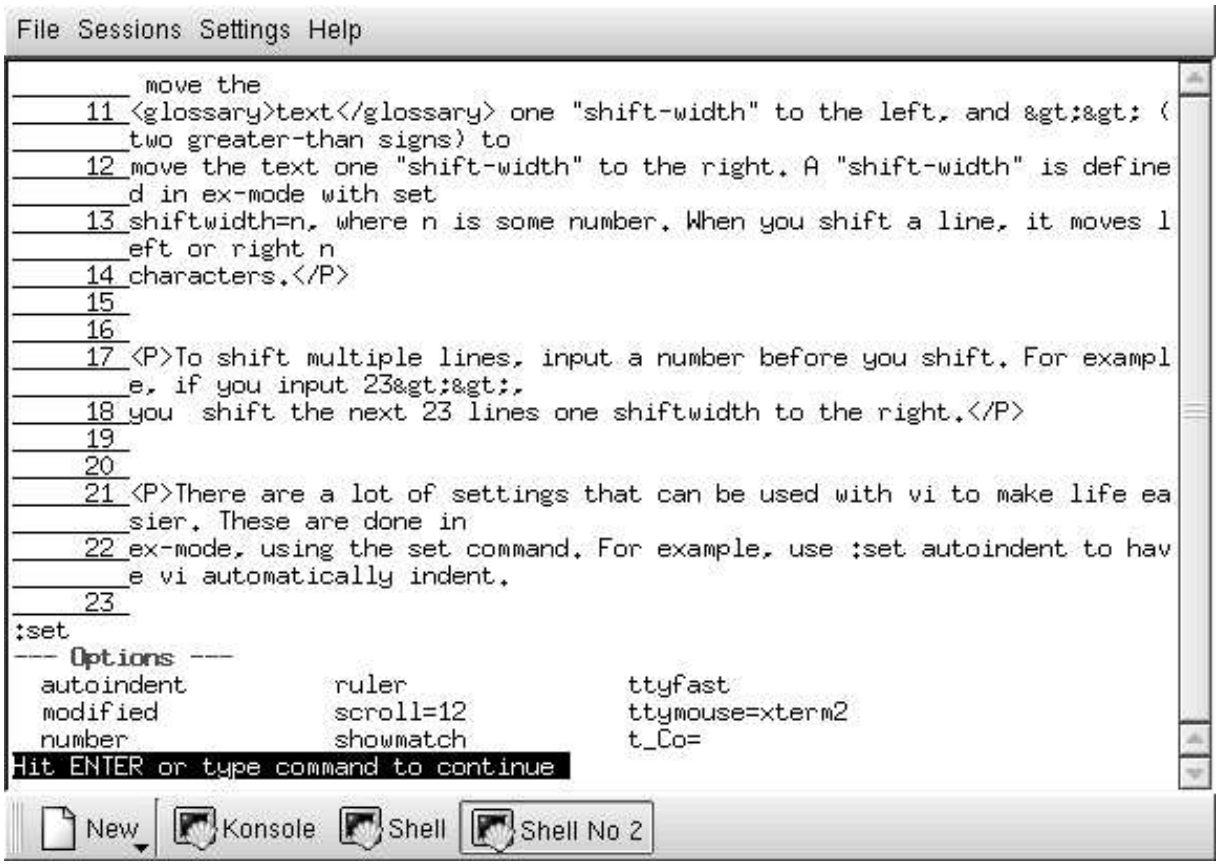


Image - Vi set command.

Inputting ":set all" will show you the value of all options. Watch out! There are a lot and typically spread across multiple screens. See the viC man-page for more details of the set command and options.

Some useful set commands include:

• wrapmargin=n	automatically "word wraps" when you get to within n spaces of the end of the line
• showmode	tells you whether you are in insert mode
• number	displays line numbers at the left-hand edge of the screen
• autowrite	Saves any changes that have been made to the current file when you issue the :n, :rew, or :! command
• ignorecase	Ignores the case of text while searching
• list	Prints end-of-line characters such as $ and tab characters such as ^I, which are normally invisible
• tabstop=n	Sets the number of spaces between each tab stop on the screen to n
• shiftwidth	Sets the number of spaces << and >> shifts each line

5.1.10 Configuring vi

When we first started talking about vi, I mentioned that there were a lot things we could do to configure it. There are mappings and abbreviations and settings that we can control. The problem is that once we leave vi, everything we added is lost.

Fortunately, there is hope. Like many programs, vi has its own configuration file: .exrc (note the dot at the front). Typically, vi just uses its standard settings and does not create this file. However, if this file resides in our home directory, it will be valid every time we start vi unless we have an .exrc file in our current directory which will then take precedence. Having multiple .exrc files is useful when doing programming as well as when editing text. When writing text, I don't need line numbers or autoindent like I do when programming.

The content and syntax of the lines is exactly the same as in vi; however, we don't have the leading colon. Part of the .exrc file in my text editing directory looks like this:

```
map! ^X :wqmap x :wqmap! ^W :wmap w :wset showmodeset wm=3abbr Unix UNIXabbr btwn betweenabbr teh theabbr refered referredabbr waht what  abbr Para {P}abbr inot into>
```

5.2 Sed

Suppose you have a file in which you need to make some changes. You could load up vi and make the changes that way, but what if what you wanted to change was the output of some command before you sent it to a file? You could first send it to a file and then edit that file, or you could use sed, which is a stream editor that is specifically designed to edit data streams.

If you read the previous section or are already familiar with either the search and replace mechanisms in vi or the editor ed, you already have a jump on learning sed. Unlike vi, sed is non-interactive, but can handle more complicated editing instructions. Because it is non-interactive, commands can be saved in text files and used over and over. This makes debugging the more complicated sed constructs much easier. For the most part, sed is

line-oriented, which allows it to process files of almost any size. However, this has the disadvantage that sed cannot do editing that is dependent on relative addressing.

Unlike the section on vi, I am not going to go into as many details about sed. However, sed is a useful tool and I use it often. The reason I am not going to cover it in too much detail is three-fold. First, much of what is true about pattern searches, addressing, etc., in vi is also true in sed. Therefore, I don't feel the need to repeat. Second, it is not that important that you become a sed expert to be a good system administrator. In a few cases, scripts on a Linux system will use sed. However, they are not that difficult to understand, provided you have a basic understanding of sed syntax. Third, sed is like any programming language, you can get by with simple things. However, to get really good, you need to practice and we just don't have the space to go beyond the basics.

In this section, I am going to talk about the basics of sed syntax, as well as some of the more common sed commands and constructs. If you want to learn more, I recommend getting *sed & awk*by Dale Dougherty from O'Reilly and Associates. This will also help you in the section on awk, which is coming up next.

The way sed works is that it reads input one line at a time, and then carries out whatever editing changes you specify. When it has finished making the changes, it writes them to stdout. Like commands such as grep and sort, sed acts like a filter. However, with sed you can create very complicated programs. Because I normally use sed as one end of a pipe, most of the sed commands that I use have the following structure:

```
first_cmd | sed <options> <edit_description>
```

This is useful when the edit descriptions you are using are fairly simple. However, if you want to perform multiple edits on each line, then this way is not really suitable. Instead, you can put all of your changes into one file and start up sed like this

```
first_cmd | sed –f editscript
```

or

```
sed –f editscript <inputfile
```

As I mentioned before, the addressing and search/replace mechanisms within sed are basically the same as within vi. It has the structure

```
[address1[,address2]] edit_description [arguments]
```

As with vi, addresses do not necessarily need to be line numbers, but can be regular expressions that sed needs to search for. If you omit the address, sed will make the changes globally, as applicable. The edit_description tells sed what changes to make. Several arguments can be used, and we'll get to them as we move along.

As sed reads the file, it copies each line into its *pattern space*. This pattern space is a special buffer that sed uses to hold a line of text as it processes it. As soon as it has finished reading the line, sed begins to apply the changes to the pattern space based on the edit description.

Keep in mind that even though sed will read a line into the pattern space, it will only make changes to addresses that match the addresses specified and does not print any warnings when this happens. In general, sed either silently ignores errors or terminates abruptly with an error message as a result of a syntax error, not because there we no matches. If there are no lines that contain the pattern, no lines match, and the edit commands are not carried out.

Because you can have multiple changes on any given line, sed will carry them each out in turn. When there are no more changes to be made, sed sends the result to its output. The next line is read in and the whole process starts over. As it reads in each line, sed will increment an internal line counter, which keeps track of the *total* number of lines read, not lines per file. This is an important distinction if you have multiple files that are being read. For example, if you had two 50-line files, from sed's perspective, line 60 would be the tenth line in the second file.

Each sed command can have 0, 1, or 2 addresses. A command with no addresses specified is applied to every line in the input. A command with one address is applied to all lines that match that address. For example:

```
/mike/s/fred/john/
```

substitutes the first instance of "john for "fred only on those lines containing "mike. A command with two addresses is applied to the first line that matches the first address, then to all subsequent lines until a match for the second address is processed. An attempt is made to match the first address on subsequent lines, and the process is repeated. Two addresses are separated by a comma.

For example

```
50,100s/fred/john/
```

substitutes the first instance of "john for "fred from line 50 to line 100, inclusive. (Note that there should be no space between the second address and the s command.) If an address is followed by an exclamation mark (!), the command is applied only to lines that do not match the address. For example

```
50,100!s/fred/john/
```

substitutes the first instance of "john for "fred everywhere except lines 50 to 100, inclusive.

Also, sed can be told to do input and output based on what it finds. The action it should perform is identified by an argument at the end of the sed command. For example, if we wanted to print out lines 5-10 of a specific file, the sed command would be

```
cat file | sed –n 5,10p
```

The -n is necessary so that every line isn't output in *addition* to the lines that match.

Remember the script we created in the first section of this chapter, where we wanted just lines 510 of every file. Now that we know how to use sed, we can change the script to be a lot more efficient. It would now look like this:

```
find ./letters/taxes -print | while read FILE do echo $FILE cat
$FILE | sed -n 5-10p done
```

Rather than sending the file through head and then the output through tail, we send the whole file through sed. It can keep track of which line is line 1, and then print the necessary lines.

In addition, sed allows you to write lines that match. For example, if we wanted all the comments in a shell script to be output to a file, we could use sed like this:

```
cat filename | sed -n /^#/w filename
```

Note that there must be exactly one space between the w and the name of the file. If we wanted to read in a file, we could do that as well. Instead of a **w** to write, we could use an **r** to read. The contents of the file would be appended after the lines specified in the address. Also keep in mind that writing to or reading from a file are independent of what happens next. For example, if we write every line in a file containing the name "John," but in a subsequent **sed** command change "John" to "Chris," the file would contain references to "John," as no changes are made. This is logical because **sed** works on each line and the lines are already in the file before the changes are made.

Keep in mind that every time a line is read in, the contents of the pattern space are overwritten. To save certain data across multiple commands, sed provides what is called the "hold space." Changes are not made to the hold space directly, rather the contents of either one can be copied into the other for processes. The contents can even be exchanged, if needed. The table below contains a list of the more common **sed** commands, including the commands used to manipulate the hold and pattern spaces.

Table sed Commands

a	append text to the pattern space
b	branch to a label
c	append text
d	delete text
D	delete all the characters from the start of the pattern space up to and including the first new line
g	overwrite the pattern space with the holding area
G	appends the holding area to the pattern space, separated with a new line
h	overwrite holding area with pattern space
H	append s the pattern space to the holding area, separated by a newlinewith a new line
i	insert text
l	list the contents of the pattern space
n	add a new line to the pattern space
N	append the next input line to the pattern space, separated lines with a new line
p	print the pattern space
P	print from the start of the pattern space up to and including the first new line
r	read in a file
s	substitute patterns
t	branch only if a substitution has been made to the current pattern space
w	writes to a file
x	interchange the contents of the pattern space and the holding area (the maximum number of addresses is two)

5.3 Awk

Another language that Linux provides and is standard on many (most?) UNIX systems is awk. The abbreviation awk is an acronym composed of the first letter of the last names of its developers: Alfred Aho, Peter Weinberger, and Brian Kernighan. Like sed, awk is an interpreted pattern-matching language. In addition, awk, like sed, can also read stdin. It can also be passed the name of a file containing its arguments.

The most useful aspect of awk (at least useful for me and the many Linux scripts that use it) is its idea of a field. Like sed, awk will read whole lines, but unlike sed, awk can immediately break into segments (fields) based on some criteria. Each field is separated by a *field separator*. By default, this separator is a space. By using the -F option on the command line or the FS variable within an awk program, you can specify a new field separator. For example, if you specified a colon (:) as a field separator, you could read in the lines from the /etc/password file and immediately break it into fields.

A programming language in its own right, awk has become a staple of UNIX systems. The basic purposes of the language are manipulating and processing text files. However, awk is also a useful tool when combined with output from other commands, and allows you to format that output in ways that might be easier to process further. One major advantage of awk is that it can accomplish in a few lines what would normally require dozens of lines in sh or csh shell script, or may even require writing something in a lower-level language, like C.

The basic layout of an awk command is

pattern { action }

where the action to be performed is included within the curly braces ({}). Like sed, awk reads one input a line at a time, aut awk sees each line as a record broken up into fields. Fields are separated by an input Field Separator (FS), which by default is a Tab or space. The FS can be changed to something else, for example, a semi-colon (;), with FS=;. This is useful when you want to process text that contains blanks; for example, data of the following form:

Blinn, David;42 Clarke Street;Sunnyvale;California;95123;33
Dickson, Tillman;8250 Darryl Lane;San Jose;California;95032;34
Giberson, Suzanne;102 Truck Stop Road;Ben Lomond;California;26
Holder, Wyliam; 1932 Nuldev Street;Mount Hermon;California;95431;42
Nathanson, Robert;12 Peabody Lane;Beaverton;Oregon;97532;33
Richards, John;1232 Bromide Drive;Boston;Massachusetts;02134;36
Shaffer, Shannon;98 Whatever Way;Watsonville;California;95332;24

Here we have name, address, city, state, zip code, and age. Without using ; as a field separator, Blinn and David;42 would be two fields. Here, we would want to treat each name, address city, etc., a single unit, rather than as multiple fields.

The basic format of an awk program or awk script, as it is sometimes called, is a pattern followed by a particular action. Like sed, each line of the input is checked by awk to see if it matches that particular pattern. Both sed and awk do well when comparing string values, However, whereas checking numeric values is difficult with sed, this functionality is an

integral part of awk.

If we wanted, we could use the data previously listed and output only the names and cities of those people under 30. First, we need an awk script, called awk.scr, that looks like this:

```
FS=; $6 < 30 { print $1, $3 }
```

Next, assume that we have a data file containing the seven lines of data above, called awk.data. We could process the data file in one of two ways. First

```
awk -f awk.scr awk.data
```

The -f option tells awk that it should read its instructions from the file that follows. In this case, awk.scr. At the end, we have the file from which awk needs to read its data.

Alternatively, we could start it like this:

```
cat awk.data | awk -f awk.scr
```

We can even make string comparisons. as in

```
$4 == "California" { print $1, $3 }
```

Although it may make little sense, we could make string comparisons on what would normally be numeric values, as in

```
$6 == "33" { print $1, $3 }
```

This prints out fields 1 and 3 from only those lines in which the sixth field equals the string 33.

Not to be outdone by sed, awk will also allow you to use regular expressions in your search criteria. A very simple example is one where we want to print every line containing the characters "on." (Note: The characters must be adjacent and in the appropriate case.) This line would look like this:

```
/on/ {print $0}
```

However, the regular expressions that awk uses can be as complicated as those used in sed. One example would be

```
/[^s]on[^;]/ {print $0}
```

This says to print every line containing the pattern on, but only if it is *not* preceded by an ^s nor followed by a semi-colon(^;). The trailing semi-colon eliminates the two town names ending in "on" (Boston and Beaverton) and the leading s eliminates all the names ending in "son." When we run awk with this line, our output is

```
Giberson, Suzanne;102 Truck Stop Road;Ben Lomond;California;96221;26
>
```

But doesn't the name "Giberson" end in "son"? Shouldn't it be ignored along with the others? Well, yes. However, that's not the case. The reason this line was printed out was because of the "on" in Ben Lomond, the city in which Giberson resides.

We can also use addresses as part of the search criteria. For example, assume that we need to print out only those lines in which the first field name (i.e., the persons last name) is in the first half of the alphabet. Because this list is sorted, we could look for all the lines between those starting with "A" and those starting with "M." Therefore, we could use a line like this:

```
/^A/,/^M/ {print $0}
```

When we run it, we get

What happened? There are certainly several names in the first half of the alphabet. Why didn't this print anything? Well, it printed exactly what we told it to print. Like the addresses in both vi and sed, awk searches for a line that matches the criteria we specified. So, what we really said was "Find the first line that starts with an A and then print all the lines up to and including the last one starting with an M." Because there was no line starting with an "A," the start address didn't exist. Instead, the code to get what we really want would look like this:

```
/^[A-M]/ {print $0}
```

This says to print all the lines whose first character is in the *range* A-M. Because this checks every line and isn't looking for starting and ending addresses, we could have even used an unsorted file and would have gotten all the lines we wanted. The output then looks like this:

```
Blinn, David;42 Clarke Street;Sunnyvale;California;95123;33 Dickson, Tillman;8250 Darryl Lane;San Jose;California;95032;34 Giberson, Suzanne;102 Truck Stop Road;Ben Lomond;California;96221;26 Holder, Wyliam; 1932 Nuldev Street;Mount Hermon;California;95431;42 >
```

If we wanted to use a starting and ending address, we would have to specify the starting letter of the name that actually existed in our file. For example:

```
/^B/,/^H/ {print $0}
```

Because printing is a very useful aspect of awk, its nice to know that there are actually two ways of printing with awk. The first we just mentioned. However, if you use printf instead of print, you can specify the format of the output in greater detail. If you are familiar with the C programming language, you already have a head start, as the format of printf is essentially the same as in C. However, there are a couple of differences that you will see immediately if you are a C programmer.

For example, if we wanted to print both the name and age with this line

```
$6 >30 {printf"%20s %5d\n",$1,$6}
```

the output would look like this:

Blinn, David	33
Dickson, Tillman	34
Holder, Wyliam	42
Nathanson, Robert	33
Richards, John	36

The space used to print each name is 20 characters long, followed by five spaces for the age.

Because awk reads each line as a single record and blocks of text in each record as fields, it needs to keep track of how many records there are and how many fields. These are denoted by the NR variable.

Another way of using awk is at the end of a pipe. For example, you may have multiple-line output from one command or another but only want one or two fields from that line. To be more specific, you may only want the permissions and file names from an ls -l output. You would then pipe it through awk, like this

```
ls –l | awk ’{ print $1" "$9 }’
```

and the output might look something like this:

```
-rw-r--r-- mike.letter -rw-r--r-- pat.note -rw-r--r-- steve.note -rw-r--r-- zoli.letter >
```

This brings up the concept of variables. Like other languages, awk enables you to define variables. A couple are already predefined and come in handy. For example, what if we didn’t know off the tops of our heads that there were nine fields in the ls -l output? Because we know that we wanted the first and the last field, we can use the variable that specifies the number of fields. The line would then look like this:

```
ls –l | awk ’{ print $1" "$NF }’
```

In this example, the space enclosed in quotes is necessary; otherwise, awk would print $1 and $NR right next to each other.

Another variable that awk uses to keep track of the number of records read so far is NR. This can be useful, for example, if you only want to see a particular part of the text. Remember our example at the beginning of this section where we wanted to see lines 5-10 of a file (to look for an address in the header)? In the last section, I showed you how to do it with sed, and now I’ll show you with awk.

We can use the fact that the NR variable keeps track of the number of records, and because each line is a record, the NR variable also keeps track of the number of lines. So, we’ll tell awk that we want to print out each line between 5 and 10, like this:

```
cat datafile | awk ’{NR >=5 && NR <= 10 }’
```

This brings up four new issues. The first is the NR variable itself. The second is the use of the double ampersand (&&). As in C, this means a logical AND. Both the right and the left sides of the expression must be true for the entire expression to be true. In this example, if we read a line and the value of NR is greater than or equal to 5 (i.e., we have read in at least five lines) *and* the number of lines read is not more than 10, the expression meets the logical AND

criteria. The third issue is that there is no print statement. The default action of awk, when it doesn't have any additional instructions, is to print out each line that matches the pattern. (You can find a list of other built in variables in the table below)

The last issue is the use of the variable NR. Note that here, there is no dollar sign ($) in front of the variable because we are looking for the value of NR, not what it points to. We do not need to prefix it with $ unless it is a field variable. Confused? Lets look at another example.

Lets say we wanted to print out only the lines where there were more than nine fields. We could do it like this:

```
cat datafile | awk '{ NF > 9 }'
```

Compare this

```
cat datafile | awk { print $NF }
```

which prints out the last field in every line. (You can find a list of other built in variable in the table below)

Up to now, we've been talking about one line awk commands. These have all performed a single action on each line. However, awk has the ability to do multiple tasks on each line as well as a task before it begins reading and after it has finished reading.

We use the BEGIN and END pair as markers. These are treated like any other pattern. Therefore, anything appearing after the BEGIN pattern is done before the first line is read. Anything after the END pattern is done after the last line is read. Lets look at this script:

```
BEGIN { FS=";"} {printf"%s\n", $1} {printf"%s\n", $2} {printf"%s, %s\n",$3,$4} {printf"%s\n", $5} END {print "Total Names:" NR}
>
```

Following the BEGIN pattern is a definition of the field separator. This is therefore done before the first line is read. Each line is processed four times, where we print a different set of fields each time. When we finish, our output looks like this:

```
Blinn, David 42 Clarke Street Sunnyvale, California  95123
Dickson, Tillman 8250 Darryl Lane San Jose, California95032
Giberson, Suzanne 102 Truck Stop Road Ben Lomond, California 96221
Holder, Wyliam 1932 Nuldev Street Mount Hermon, California 95431
Nathanson, Robert 12 Peabody Lane Beaverton, Oregon 97532
Richards, John 1232 Bromide Drive Boston, Massachusetts 02134
Shaffer, Shannon 98 Whatever Way Watsonville, California 95332
Total Names:7
>
```

Aside from having a pre-defined set of variables to use, awk allows us to define variables ourselves. If in the last awk script we had wanted to print out, lets say, the average age, we could add a line in the middle of the script that looked like this:

```
{total = total + $6 }>
```

Because $6 denotes the age of each person, every time we run through the loop, it is added to the variable total. Unlike other languages, such as C, we don't have to initialize the variables; awk will do that for us. Strings are initialized to the null string and numeric variables are initialized to 0.

After the END, we can include another line to print out our sum, like this:

```
{print "Average age: " total/NR}
```

Table awk Comparison Operators

Operator	Meaning
<	less than
<=	less than or equal to
==	equal to
!=	not equal to
>=	greater than or equal to
>	greater than

Table Default Values of awk Built-in Variables

Variable	Meaning	Default
ARGC	number of command-line arguments	-
ARGV	array of command-line arguments	-
FILENAME	name of current input file	-
FNR	record number in current file	-
FS	input field separator	space or tab
NF	number of fields in the current record	-
NR	number of records read	-
OFMT	numeric output format	%.6g
OFS	output field separator	space
ORS	output record separator	new line
RS	input record separator	new line

Is that all there is to it? No. In fact, we haven't even touched the surface. awk is a very complex programming language and there are dozens more issues that we could have addressed. Built into the language are mathematical functions, if and while loops, the ability to create your own functions, strings and array manipulation, and much more.

Unfortunately, this is not a book on UNIX programming languages. Some readers may be disappointed that I do not have the space to cover awk in more detail. I am also disappointed. However, I have given you a basic introduction to the constructs of the language to enable you to better understand the more than 100 scripts on your system that use awk in some way.

5.4 Perl

This section is old and incomplete, even as an introduction. Although it covers a number of different aspects of perl, this section really needs to be redone. Any volunteers?

If you plan to do anything serious on the Web, I suggest that you learn perl. In fact, if you plan to do anything serious on your machine, then learning perl is also a good idea. Although not available on a lot of commercial versions, perl is almost universally available with Linux.

Now, I am not saying that you shouldn't learn sed, awk, and shell programming. Rather, I am saying that you should learn all four. Both sed and awk have been around for quite a while, so they are deeply ingrained in the thinking of most system administrators. Although you could easily find a shell script on the system that didn't have elements of sed or awk in it, you would be very hard pressed to find a script that had no shell programming in it. On the other hand,

most of the scripts that process information from other programs use either sed or awk. Therefore, it is likely that you will eventually come across one or the other.

perl is another matter altogether. None of the standard scripts have perl in them. This does not say anything about the relative value of perl, but rather the relative availability of it. Because it can be expected that awk and sed are available, it makes sense that they are commonly used. perl may not be on your machine and including it in a system shell script might cause trouble.

In this section, I am going to talk about the basics of perl. We'll go through the mechanics of creating perl scripts and the syntax of the perl language. There are many good books on perl, so I would direct you to them to get into the nitty-gritty. Here we are just going to cover the basics. Later on, we'll address some of the issues involved with making perl scripts to use on your Web site.

One aspect of perl that I like is that it contains the best of everything. It has aspects of C, shell, awk, sed and many other things. perl is also free. The source code is readily available and the versions that I have came with configuration scripts that determined the type of system I had and set up the make-files accordingly. Aside from Linux, I was able to compile the exact same source on my Sun Solaris workstation. Needless to say, the scripts that I write at home run just as well at work.

I am going to make assumptions as to what level of programming background you have. If you read and understood the sections on sed, awk, and the shell, then you should be ready for what comes next. In this section, I am going to jump right in. I am not going to amaze you with demonstrations of how perl can do I/O, as that's what we are using it for in the first place. Instead, I am going to assume that you want to do I/O and jump right into how to do it.

Lets create a shell script called hello.pl. The pl extension has no real meaning, although I have seen many places where it is always used as an extension. It is more or less conventional to do this, just as text files traditionally have the extension .txt, shell scripts end in .sh, etc.

We'll start off with the traditional

```
print "Hello, World!\n";
```

This shell script consists of a single perl statement, whose purpose is to output the text inside the double-quotes. Each statement in perl is followed by a semi-colon. Here, we are using the perl print function to output the literal string "Hello, World!\n" (including the trailing new line). Although we don't see it, there is the implied file handle to stdout. The equivalent command with the explicit reference would be

```
print STDOUT "Hello, World!\n";
```

Along with STDOUT, perl has the default file handlers STDIN and STDERR. Here is a quick script that demonstrates all three as well as introduces a couple of familiar programming constructs:

```
while (<STDIN>)
```

```
{
if ( $_ eq "\n" )
{
print STDERR "Error: \n";
} else {
print STDOUT "Input: $_ \n";
}
}
```

Functioning the same as in C and most shells, the while line at the top says that as long as there is something coming from STDIN, do the loop. Here we have the special format (<STDIN>), which tells perl where to get input. If we wanted, we could use a file handle other than STDIN. However, we'll get to that in a little bit.

One thing that you need to watch out for is that you must include blocks of statements (such as after while or if statements) inside the curly braces ({}). This is different from the way you do it in C, where a single line can follow while or if. For example, this statement is not valid in perl:

```
while ( $a < $b )
$a++;
```

You would need to write it something like this:

```
while ( $a < $b ) {
$a++;
}
```

Inside the while loop, we get to an if statement. We compare the value of the special variable $_ to see if it is empty. The variable $_ serves several functions. In this case, it represents the line we are reading from STDIN. In other cases, it represents the pattern space, as in sed. If the latter is true, then just the Enter key was pressed. If the line we just read in is equal to the newline character (just a blank line), we use the print function, which has the syntax

```
print [filehandler] "text_to_print";
```

In the first case, filehandler is stderr and in the second case stdout is the filehandler. In each case, we could have left out the filehandler and the output would go to stout.

Each time we print a line, we need to include a newline (\n) ourselves.

We can format the print line in different ways. In the second print line, where the input is not a blank line, we can print "Input: " before we print the line just input. Although this is a very simple way of outputting lines, it gets the job done. More complex formatting is possible with the perl printf function. Like its counterpart in C or awk, you can come up with some very elaborate outputs. We'll get into more details later.

One more useful function for processing lines of input is split. The split function is used to, as its name implies, to split a line based on a field separator that you define. Say, for example, a space. The line is then stored in an array as individual elements. So, in our example, if we wanted to input multiple words and have them parsed correctly, we could change the script to look like this:

```
while (<STDIN>)

{

@field = split( ,$_);

if ( $_ eq "\n" )

{

print STDERR "Error: \n";

} else {

print STDOUT "$_ \n";

print $field[0];

print $field[1];

print $field[2];

}

}
```

The split function has the syntax

```
split(pattern,line);
```

where pattern is our field separator and line is the input line. So our line

```
@field = split( ,$_);
```

says to split the line we just read in (stored in $_) and use a space ( ) as the field separator. Each field is then placed into an element of the array field. The @ is needed in front of the variable field to indicate that it's an array. In perl, there are several types of variables. The first kind we have already met before. The special variable $_ is an example of a scalar variable. Each scalar variable is preceded by a dollar sign ($) and can contain a single value, whether a character string or a number. How does perl tell the difference? It depends on the

context. perl will behave correctly by looking at what you tell it to do with the variable. Other examples of scalars are

$name = "jimmo";

$initial = j;

$answertolifetheuniverseandeverything = 42;

Another kind of variable is an array, as we mentioned before. If we precede a variable with %, we have an array. But don't we have an array with @? Yes, so whats the difference? The difference is that arrays, starting with the @, are referenced by numbers, while those starting with the % are referenced by a string. We'll get to how that works as we move along.

In our example, we are using the split function to fill up the array @field. This array will be referenced by number. We see the way it is referenced in the three print statements toward the end of the script.

If our input line had a different field separator (for example, %), the line might look like this:

@field = split(%,$_);

In this example, we are outputting the first three words that are input. But what if there are more words? Obviously we just add more print statements. What if there are fewer words? Now we run into problems. In fact, we run into problems when adding more print statements. The question is, where do we stop? Do we set a limit on the number of words that can be input? Well, we can avoid all of these problems by letting the system count for us. Changing the script a little, we get

```
while (<STDIN>)

{

@field = split( ,$_);

if ( $_ eq "\n" )

{

print STDERR "Error: \n";

} else {

foreach $word (@field){

print $word,"\n";

}

}
```

}

In this example, we introduce the foreach construct. This has the same behavior as a for loop. In fact, in perl, for and foreach are interchangeable, provided you have the right syntax. In this case, the syntax is

foreach $variable (@array)

where $variable is our loop variable and @array is the name of the array. When the script is run, @array is expanded to its components. So, if we had input four fruits, our line might have looked like this:

foreach $word(apple,bananna,cherry,orange);

Because I don't know how many elements there are in the array field, foreach comes in handy. In this example, every word separated by a space will be printed on a line by itself, like this:

perl script.pl

one two three

one

two

three

^D

Our next enhancement is to change the field separator. This time we'll use an ampersand (&) instead. The split line now looks like this:

@field = split(&,$_);

When we run the script again with the same input, what we get is a bit

different:

perl script.pl

one two three

one two three

The reason why we get the output on one line is because the space is no longer a field separator. If we run it again, this time using &, we get something different:

perl script.pl

one&two&three

one

two

three

This time, the three words were recognized as separate fields.

Although it doesn't seem too likely that you would be inputting data like this from the keyboard, it is conceivable that you might want to read a file that has data stored like this. To make things easy, I have provided a file that represents a simple database of books. Each line is a record and represents a single book, with the fields separated by %.

To be able to read from a file, we must create a file handle. To do this, we add a line and change the while statement so it looks like this:

open (INFILE,"< bookdata.txt");

while (<INFILE>)

The syntax of the open function is

open(file_handle,openwhat_&_how);

The way we open a file depends on the way we want to read it. Here, we use standard shell redirection symbols to indicate how we want to read the specified file. In our example, we indicate redirection *from* the file bookdata.txt. This says we want to read *from* the file. If we wanted to open the file for writing, the line would look like this:

open (INFILE,"> bookdata.txt");

If we wanted to append to the file, we could change the redirections so the line would look like this:

open (INFILE,">> bookdata.txt");

Remember I said that we use standard redirection symbols. This also includes the pipe symbol. As the need presents itself, your perl script can open a pipe for either reading or writing. Assuming that we want to open a pipe for writing that sends the output through sort, the line might look like this:

open (INFILE,"| sort ");

Remember that this would work the same as from the command line. Therefore, the output is not being written to a file; it is just being piped through sort. However, we could redirect the output of sort , if we wanted. For example:

open (INFILE,"| sort > output_file");

This opens the file output_file for writing, but the output is first piped through sort . In our example, we are opening the file bookdata.txt for reading. The while loop continues through and outputs each line read. However, instead of being on a single line, the individual fields

(separated by &) are output on a separate line.

We can now take this one step further. Lets now assume that a couple of the fields are actually composed of subfields. These subfields are separated by a plus sign (+). We now want to break up every field containing + into its individual subfields.

As you have probably guessed, we use the split command again. This time, we use a different variable and instead of reading out of the input line ($_), we read out of the string $field. Therefore, the line would look like this:

```
@subfield = split(\+,$field);
```

Aside from changing the search pattern, I added the back slash (\) because + is used in the search pattern to represent one or more occurrences of the preceding character. If we don't escape it, we generate an error. The whole script now looks like this:

```
open(INFILE,"<bookdata.txt");

while (<INFILE>)

{

@data = split(&,$_);

if ( $_ eq "\n" )

{

print STDERR "Error: \n";

} else {

foreach $field (@data){

@subfield = split(\+,$field);

foreach $word (@subfield){

print $word,"\n";

}

}

}

}
```

If we wanted, we could have written the script to split the incoming lines at both & and +. This would have given us a split line that looked like this:

@data = split([&\+],$_);

The reason for writing the script like we did was that it was easier to separate subfields and still maintain their relationships. Note that the search pattern used here could have been any regular expression. For example, we could have split the strings every place there was the pattern Di followed by e, g, or r, but *not* if it was followed by i. The regular expression would be

Di[reg][^i]

so the split function would be:

@data = split(Di[reg][^i],$_);

At this point, we can read in lines from an ASCII file, separate the lines based on what we have defined as fields, and then output each line. However, the lines don't look very interesting. All we are seeing is the content of each field and do not know what each field represents. Let's change the script once again. This time we will make the output show us the field names as well as their content.

Lets change the script so that we have control over where the fields end up. We still use the split statement to extract individual fields from the input string. This is not necessary because we can do it all in one step, but I am doing it this way to demonstrate the different constructs and to reiterate that in perl, there is always more than one way do to something. So, we end up with the following script:

```
open(INFILE,"< bookdata.txt");

while (<INFILE>)

{

@data = split(&,$_);

if ( $_ eq "\n" )

{

print STDERR "Error: \n";

} else {

$fields = 0;

foreach $field (@data){

$fieldarray[$fields] = $field;

print $fieldarray[$fields++]," ";
```

```
}

}

}
```

Each time we read a line, we first split it into the array @data, which is then copied into the fields array. Note that there is no new line in the print statement, so each field will be printed with just a space and the newline read at the end of each input line will then be output. Each time through the loop, we reset our counter (the variable $fields) to 0.

Although the array is re-filled every time through the loop and we lose the previous values, we could assign the values to specific variables.

Lets now make the output a little more attractive by outputting the field headings first. To make things simpler, lets label the fields as follows

title, author, publisher, char0, char1, char2, char3, char4, char5

where char0-char5 are simply characteristics of a book. We need a handful of if statements to make the assignment, which look like this:

```
foreach $field (@data){

if ( $fields = = 0 ){

print "Title: ",$field;

}

if ( $fields = = 1 ){

print "Author: ",$field;

}

*

*

*

if ( $fields = = 8 ){

print "Char 5: ",$field;

}
```

Here, too, we would be losing the value of each variable every time through the loop as they get overwritten. Lets just assume we only want to save this information from the first line (our reasoning will become clear in a minute). First we need a counter to keep track of what line we are on and an if statement to enter the block where we make the assignment. Rather than a print statement, we change the line to an assignment, so it might look like this:

$title = $field;

When we read subsequent lines, we can output headers for each of the fields. We do this by having another set of if statements that output the header and then the value, which is based on its position.

Actually, there is a way of doing things a little more efficiently. When we read the first line, we can assign the values to variables on a single line. Instead of the line

foreach $field (@data) {

we add the if statement to check if this is the first line. Then we add the line

($field0,$field1,$field2,$field3,$field4,$field5,$field6,$field7,$field8)=

split(&,$_);

Rather than assigning values to elements in an array, we are assigning them to specific variables. (Note that if there are more fields generated by the split command than we specified variables for, the remaining fields are ignored.) The other advantage of this is that we saved ourselves a lot of space. We could also call these $field1, $field2, etc., thereby making the field names a little more generic. We could also modify the split line so that instead of several separate variables, we have them in a single array called field and we could use the number as the offset into the array. Therefore, the first field would be referenced like this:

$field[0]

The split command for this would look like this

@field=split(&,$_);

which looks like something we already had. It is. This is just another example of the fact that there are always several different ways of doing things in perl.

At this point, we still need the series of if statements inside of the foreach loop to print out the line. However, that seems like a lot of wasted space. Instead, I will introduce the concept of an associated list. An associated list is just like any other list, except that you reference the elements by a label rather than a number.

Another difference is that associated arrays, also referred to as associated lists, are always an even length. This is because elements come in pairs: label and value. For example, we have:

%list= (name,James Mohr, logname, jimmo, department,IS);

Note that instead of $ or @ to indicate that this is an array, we use %. This specifies that this is an associative array, so we can refer to the value by label; however, when we finally reference the value, we use $. To print out the name, the line would look like this:

print "Name:",$list{name};

Also, the brackets we use are different. Here we use curly braces ({}) instead of square brackets ([]).

The introduction of the associate array allows us to define field labels within the data itself and access the values using these labels. As I mentioned, the first line of the data file containing the field labels. We can use these labels to reference the values. Lets look at the program itself:

```
open(INFILE,"< bookdata.txt");

$lines=0;

while (<INFILE>)

{

chop;

@data = split(&,$_);

if ( $lines == 0 )

{

@headlist=split(&,$_);

foreach $field (0..@headlist-1){

%headers = ( $headlist[$field], );

}

$lines++;

} else {

foreach $field (0..@data-1){

$headers{$headlist[$field]}=@data[$field];

print $headlist[$field],": ", $headers{$headlist[$field]},"\n";

}

}

}
```

At the beginning of the script, we added the chop function, which "chops" off the last character of a list or variable and returns that character. If you don’t mention the list or variable, chop affects the $_ variable. This function is useful to chop off the newline character that gets read in. The next change is that we removed the block that checked for blank lines and generated an error.

The first time we read a line, we entered the appropriate block. Here, we just read in the line containing the field labels and we put each entry into the array headlist via the split function. The foreach loop also added some new elements:

```
foreach $field (0..@headlist-1){

%headers = ( $headlist[$field], );

}
```

The first addition is the element (0.. @headlist-1). Two numbers separated by two dots indicate a range. We can use @headlist as a variable to indicate how many elements are in the array headlist. This returns a human number, not a computer number (one that starts at 0). Because I chose to access all my variables starting with 0, I needed to subtract 1 from the value of @headlist. There are nine elements per line in the file bookdata.txt; therefore, their range is 0..9-1.

However, we don't need to know that! In fact, we don't even know how many elements there are to make use of this functionality. The system knows how many elements it read in, so we don't have to. We just use @headlist-1 (or whatever).

The next line fills in the elements of our associative array:

```
%headers = ( $headlist[$field], );
```

However, we are only filling in the labels and not the values themselves. Therefore, the second element of the pair is empty (). One by one, we write the label into the first element of each pair.

After the first line is read, we load the values. Here again, we have a foreach loop that goes from 0 to the last element of the array. Like the first loop, we don't need to know how many elements were read, as we let the system keep track of this for us. The second element in each pair of the associative list is loaded with this line:

```
$headers{$headlist[$field]}=@data[$field];
```

Lets take a look at this line starting at the left end. From the array @data (which is the line we just read in), we are accessing the element at the offset that is specified by the variable $field. Because this is just the counter used for our foreach loop, we go through each element of the array data one by one. The value retrieved is then assigned to the left-hand side.

On the left, we have an array offset being referred to by an array offset. Inside we have

```
$headlist[$field]
```

The array headlist is what we filled up in the first block. In other words, the list of field headings. When we reference the offset with the $field variable, we get the field heading. This will be used as the string for the associative array. The element specified by

$headers{$headlist[$field]}

corresponds to the field value. For example, if the expression

$headlist[$field]}

evaluated to title, the second time through the loop, the expression $headers{$headlist[$field} would evaluate to "2010: Odyssey Two."

At this point, we are ready to make our next jump. We are going to add the functionality to search for specific values in the data. Lets assume that we know what the fields are and wish to search for a particular value. For example, we want all books that have scifi as field char0. Assuming that the script was called book.pl, we would specify the field label and value like this:

perl book.pl char0=scifi

Or we could add #!/usr/bin/perl to the top of the script to force the system to use perl as the interpreter. We would run the script like this:

book.pl char0=scifi

The completed script looks like this:

```
($searchfield,$searchvalue) = split(=,$ARGV[0]);

open(INFILE,"< bookdata.txt");

$lines=0;

while (<INFILE>)

{

chop;

@data = split(&,$_);

if ( $_ eq "\n" )

{

print STDERR "Error: \n";

} else {

if ( $lines == 0 )

{

@headlist=split(&,$_);
```

```
foreach $field (0..@headlist-1){
%headers = ( $headlist[$field], );
}
$lines++;
} else { foreach $field (0..@data-1){
$headers{$headlist[$field]}=@data[$field];
if ( ($searchfield eq $headlist[$field] ) &&
($searchvalue eq $headers{$headlist[$field]} )) {
$found=1;
}
}
}
}
if ( $found == 1 )
{
foreach $field (0..@data-1){
print $headlist[$field],": ", $headers{$headlist[$field]},"\n";
}
}
$found=0;
```

< P>}

We added a line at the top of the script that splits the first argument on

the command line:

```
($searchfield,$searchvalue) = split(=,$ARGV[0]);
```

Note that we are accessing ARGV[0]. This is not the command being called, as one would expect in a C or shell program. Our command line has the string char0=scifi as its $ARGV[0]. After the split, we have $searchfield=char0 and $searchvalue=scifi.

Some other new code looks like this:

```
if ( ($searchfield eq $headlist[$field] ) &&

($searchvalue eq $headers{$headlist[$field]} )) {

$found=1;
```

Instead of outputting each line in the second foreach loop, we are changing it so that here we are checking to see if the field we input, $searchfield, is the one we just read in $headlist[$field] and if the value we are looking for, ($searchvalue), equals the one we just read in.

Here we add another new concept: logical operators. These are just like in C, where && means a logical AND and || is a logical OR. If we want a logical comparison of two variables and each has a specific value, we use the logical AND, like

```
if ( $a == 1 && $b = 2)
```

which says if $a equals 1 AND $b equals 2, execute the following block. If we wrote it like this

```
if ( $a == 1 || $b = 2)
```

it would read as follows: if $a equals 1 OR $b equals 2, execute the block. In our example, we are saying that if the search field ($searchfield) equals the corresponding value in the heading list ($headlist[$field]) AND the search value we input ($searchvalue) equals the value from the file ($headers{$headlist[$field]}), we then execute the following block. Our block is simply a flag to say we found a match.

Later, after we read in all the values for each record, we check the flag. If the flag was set, the foreach loop is executed:

```
if ( $found == 1 )

{

foreach $field (0..@data-1){

print $headlist[$field],": ", $headers{$headlist[$field]},"\n";

}
```

Here we output the headings and then their corresponding values. But what if we aren't sure of the exact text we are looking for. For example, what if we want all books by the author Eddings, but do not know that his first name is David? Its now time to introduce the perl function index. As its name implies, it delivers an index. The index it delivers is an offset of one string in another. The syntax is

```
index(STRING,SUBSTRING,POSITION)
```

where STRING is the name of the string that we are looking in, SUBSTRING is the substring that we are looking for, and POSITION is where to start looking. That is, what position to start from. If POSITION is omitted, the function starts at the beginning of STRING. For example

index(pie,applepie);

will return 5, as the substring pie starts at position 5 of the string applepie. To take advantage of this, we only need to change one line. We change this

if (($searchfield eq $headlist[$field]) &&

($searchvalue eq $headers{$headlist[$field]})) {

to this

if ((index($headlist[$field],$searchfield)) != -1 &&

index($headers{$headlist[$field]},$searchvalue) != -1) {

Here we are looking for an offset of -1. This indicates the condition where the substring is *not* within the string. (The offset comes before the start of the string.) So, if we were to run the script like this

script.pl author=Eddings

we would look through the field author for any entry containing the string Eddings. Because there are records with an author named Eddings, if we looked for Edding, we would still find it because Edding is a substring of "David Eddings."

As you might have noticed, we have a limitation in this mechanism. We must ensure that we spell things with the right case. Because Eddings is uppercase both on the command line and in the file, there is no problem. Normally names are capitalized, so it would make sense to input them as such. But what about the title of a book? Often, words like "the" and "and" are not capitalized. However, what if the person who input the data, input them as capitals? If you looked for them in lowercase, but they were in the file as uppercase, you'd never find them.

To consider this possibility, we need to compare both the input and the fields in the file in the same case. We do this by using the tr (translate) function. It has the syntax

tr/SEARCHLIST/REPLACEMENTLIST/[options]

where SEARCHLIST is the list of characters to look for and REPLACEMENTLIST is the characters to use to replace those in SEARCHLIST. To see what options are available, check the perl man-page. We change part of the script to look like this:

foreach $field (0..@data-1){

$headers{$headlist[$field]}=@data[$field];

```
($search1 = $searchfield) =~ tr/A-Z/a-z/;

($search2 = $headlist[$field] ) =~ tr/A-Z/a-z/;

($search3 = $searchvalue)=~tr/A-Z/a-z/;

($search4 = $headers{$headlist[$field]})=~tr/A-Z/a-z/;

if ( (index($search2,$search1) != -1) && (index($search4,$search3) != -1) ) {

$found=1;

}

}
```

In the middle of this section are four lines where we do the translations. This demonstrates a special aspect of the tr function. We can do a translation as we are assigning one variable to another. This is useful because the original strings are left unchanged. We must change the statement with the index function and make comparisons to reflect the changes in the variables.

So at this point, we have created an interface in which we can access a "database" and search for specific values.

When writing conditional statements, you must be sure of the condition you are testing. Truth, like many other things, is in the eye of the beholder. In this case, it is the perl interpreter that is beholding your concept of true. It may not always be what you expect. In general, you can say that a value is true unless it is the null string (), the number zero (0), or the literary string zero ("0").

One important feature of perl is the comparison operators. Unlike C, there are different operators for numeric comparison and for string comparison. They’re all easy to remember and you have certainly seen both sets before, but keep in mind that they are different. Table 0-8 contains a list of the perl comparison operators and Table 0-9 contains a list of perl operations.

Table -8 perl Comparison Operators

Numeric	String	Comparison
==	eq	equal to
!=	ne	not equal to
>	gt	greater than
<	lt	less than
>=	ge	greater than or equal to
<=	le	less than or equal to
<=>	cmp	not equal to and sign is returned (0 - strings equal, 1 - first string less, -1 - first string greater)

Another important aspect that you need to keep in mind is that there is really no such thing as a numeric variable. Well, sort of. perl is capable of distinguishing between the two without you interfering. If a variable is used in a context where it can only be a string, then that's they way perl will interpret it as a string.

Lets take two variables: $a=2 and $b=10. As you might expect, the expression $a < $b evaluates to true because we are using the numeric comparison operator <. However, if the expression were $a lt $b, it would evaluate to false. This is because the string "10" comes before "2" lexigraphically (it comes first alphabetically).

Besides simply translating sets of letters, perl can also do substitution. To show you this, I am going to show you another neat trick of perl. Having been designed as a text and file processing language, it is very common to read in a number of lines of data and processing them all in turn. We can tell perl that it should assume we want to read in lines although we don't explicitly say so. Lets take a script that we call fix.pl. This script looks like this:

```
s/James/JAMES/g;

s/Eddings/EDDINGS/g;
```

This syntax is the same as you would find in sed; however, perl has a much larger set of regular expressions. Trying to run this as a script by itself will generate an error; instead, we run it like this:

```
perl -p fix.pl bookdata.pl
```

The -p option tells perl to put a wrapper around your script. Therefore, our script would behave as though we had written it like this:

```
while (<>) {

s/James/JAMES/g;

s/Eddings/EDDINGS/g;

} continue {

print;

}
```

This would read each line from a file specified on the command line, carry out the substitution, and then print out each line, changed or not. We could also take advantage of the ability to specify the interpreter with #!. The script would then look like

```
#!/usr/bin/perl -p

s/James/JAMES/g;

s/Eddings/EDDINGS/g;
```

Another command line option is -i. This stands for "in-place," and with it you can edit files "in-place." In the example above, the changed lines would be output to the screen and we would have to redirect them to a file ourselves. The -i option takes an argument, which indicates the extension you want for the old version of the file. So, to use the option, we would change the first line, like this:

```
#!/usr/bin/perl -pi.old
```

With perl, you can also make your own subroutines. These subroutines can be written to return values, so that you have functions as well. Subroutines are first defined with the sub keyword and are called using &. For example:

```
#!/usr/bin/perl

sub usage {

print "Invalid arguments: @ARGV\n";

print "Usage: $0 [-t] filename\n";

}

if ( @ARGV < 1 || @ARGV > 2 ) {

&usage;

}
```

This says that if the number of arguments from the command line @ARGV is less than 1 or greater than 2, we call the subroutine usage, which prints out a usage message.

To create a function, we first create a subroutine. When we call the subroutine, we call it as part of an expression. The value returned by the subroutine/function is the value of the *last* expression evaluated.

Lets create a function that prompts you for a yes/no response:

```
#!/usr/bin/perl

if (&getyn("Do you *really* want to remove all the files in this directory? ")

eq "y\n" )

{

print "Don't be silly!\n"

}

sub getyn{

print @_;

$response = (<STDIN>);

}
```

This is a very simple example. In the subroutine getyn, we output everything that is passed to the subroutine. This serves as a prompt. We then assign the line we get from stdin to the variable $response. Because this is the last expression inside the subroutine to be evaluated, this is the value that is returned to the calling statement.

If we enter "y" (which would include the new line from the Enter key), the calling if statement passes the actual prompt as an argument to the subroutine. The getyn subroutine could then be used in other circumstances. As mentioned, the value returned includes the new line; therefore, we must check for "y\n." This is *not* "y" or "n," but rather "y#" followed by a newline.

Alternatively, we could check the response inside the subroutine. In other words, we could have added the line

```
$response =~ /^y/i;
```

We addressed the =~ characters earlier in connection with the tr function. Here as well, the variable on the left-hand side is replaced by the "evaluation" of the right. In this case, we use a pattern-matching construct: /^y/i. This has the same behavior as sed, where we are looking for a y at the beginning of the line. The trailing i simply says to ignore the case. If the first character begins with a y or Y, the left-hand side ($response) is assigned the value 1; if not, it becomes a null string.

We now change the calling statement and simply leave off the comparison to "y\n". Because the return value of the subroutine is the value of the last expression evaluated, the value

returned now is either "1" or ". Therefore, we don't have to do any kind of comparison, as the if statement will react according to the return value.

I wish I could go on. I haven't even hit on a quarter of what perl can do. Unfortunately, like the sections on sed and awk, more details are beyond the scope of this book. Instead, I want to refer you to a few other sources. First, there are two books from O'Reilly and Associates. The first is *Learning perl* by Randal Schwartz. This is a tutorial. The other is *Programming perl* by Larry Wall and Randal Schwartz. If you are familiar with other UNIX scripting languages, I feel you would be better served by getting the second book.

The next suggestion I have is that you get the perl CD-ROM from Walnut Creek CD-ROM (*www.cdrom.com*). This is loaded with hundreds of megabytes of perl code and the April 1996 version, which I used, contains the source code for perl 4 (4.036) and perl5 (5.000m). In many cases, I like this approach better because I can see how to do the things I need to do. Books are useful to get the basics and reminders of syntax, options, etc. However, seeing someone else's code shows me how to do it.

Another good CD-ROM is the Mother of PERL CD from InfoMagic (*www.infomagic.com*). It, too, is loaded with hundreds of megabytes of perl scripts and information.

There are a lot of places to find sample scripts while you are waiting for the CD to arrive. One place is the Computers and Internet: Programming Languages: Perl hierarchy at Yahoo. (*www.yahoo.com*). You can use this as a springboard to many sites that not only have information on perl but data on using perl on the Web (e.g., in CGI scripts).

Chapter 6 Basic Administration

It's difficult to put together a simple answer when I'm asked about the job of a system administrator. Every aspect of the system can fall within the realm of a system administrator. Entire books have been written about just the software side, and for most system administrators, hardware, networks, and even programming fall into their laps.

I work for the largest developer of online broker software in Germany. In addition to the software, we also run the data centers for several online brokers. I am responsible for monitoring the systems and providing reports on several levels, performance and many other things. I am expected to understand how our software works with all of its various components, how they work with third party products; as well as the workings of the network, firewalls, Solaris, Linux, Windows 2000 and XP, perl, shell scripting, and so forth.

There is very little here on my site that does not directly relate to my job as a system administrator. For the most part, you need to be a jack of all trades. Although Linux has come a long way in the last few years and you no longer need to be a "guru" to get it to work, knowing how to administer your system allows you to go beyond what is delivered to you out of the box.

In this chapter, we are just going to go through the basics. We won't necessarily be talking about individual steps or processes used by the administrator, but rather about functional *areas*. With this, I hope to be able to give you enough background to use the programs and utilities that the system provides for you.

6.1 Starting and Stopping the System

Almost every user and many administrators never see what happens while the system boots, and those who do often do not understand what they are seeing. Those who do often are not sure what is happening. From the time you flip the power switch to the time you get that first login: prompt, dozens of things must happen, many of which happen long before the system knows that it's running Linux. Knowing what is happening as the system boots and in what order it is happening is very useful when your system does not start the way it should.

In this chapter, I will first talk about starting your system. Although you can get it going by flipping on the power switch and letting the system boot by itself, there are many ways to change the behavior of your system as it boots. How the system boots depends on the situation. As we move along through the chapter, we'll talk about the different ways to influence how the system boots.

After we talk about how to start your system, we'll look at a few ways to alter your system's behavior when it shuts down.

6.1.1 The Boot Process

The process of turning on your computer and having it jump through hoops to bring up the operating system is called *booting*, which derives from the term *bootstrapping.* This is an allusion to the idea that a computer pulls itself up by its bootstraps, in that smaller pieces of

simple code start larger, more complex pieces to get the system running.

The process a computer goes through is similar among different computer types, whether it is a PC, Macintosh, or SPARC Workstation. In the next section, I will be talking specifically about the PC, though the concepts are still valid for other machines.

The first thing that happens is the Power-On Self-Test POST. Here the hardware checks itself to see that things are all right. It compares the hardware settings in the CMOS Complementary Metal Oxide Semiconductor to what is physically on the system. Some errors, like the floppy types not matching, are annoying, but your system still can boot. Others, like the lack of a video card, can keep the boot process from continuing. Often, there is nothing to indicate what the problem is, except for a few little "beeps."

Once the POST is completed, the hardware jumps to a specific, predefined location in RAM. The instructions located here are relatively simple and basically tell the hardware to go look for a boot device. Depending on how your CMOS is configured, the hardware first checks your floppy and then your hard disk.

When a boot device is found let’s assume that it’s a hard disk, the hardware is told to go to the 0th first sector cylinder 0, head 0, sector 0, then load and execute the instructions there. This is the master boot record, or MBR for you DOS-heads sometimes also called the master boot block. This code is small enough to fit into one block but is intelligent enough to read the partition table located just past the master boot block and find the active partition. Once it finds the active partition, it begins to read and execute the instructions contained within the first block.

It is at this point that viruses can affect/infect Linux systems. The master boot block has the same format for essentially all PC-based operating systems and it does is find and execute code at the beginning of the active partition. But if the master boot block contains code that tells it to go to the very last sector of the hard disk and execute the code there, which then tells the system to execute code at the beginning of the active partition, you would never know anything was wrong.

Let’s assume that the instructions at the very end of the disk are larger than a single 512-byte sector. If the instructions took up a couple of kilobytes, you could get some fairly complicated code. Because it is at the end of the disk, you would probably never know it was there. What if that code checked the date in the CMOS and, if the day of the week was Friday and the day of the month was 13, it would erase the first few kilobytes of your hard disk? If that were the case, then your system would be infected with the Friday the 13th virus, and you could no longer boot your hard disk.

Viruses that behave in this way are called "boot viruses," as they affect the master boot block and can only damage your system if this is the disk from which you are booting. These kinds of viruses can affect all PC-based systems. Some computers will allow you to configure them more on that later so that you cannot write to the master boot block. Although this is a good safeguard against older viruses, the newer ones can change the CMOS to allow writing to the master boot block. So, just because you have enabled this feature does not mean your system is safe. However, I must point out that boot viruses can only affect Linux systems if you boot

from an infected disk. This usually will be a floppy, more than likely a DOS floppy. Therefore, you need to be especially careful when booting from floppies.

Now back to our story...

As I mentioned, the code in the master boot block finds the active partition and begins executing the code there. On an MS-DOS system, these are the IO.SYS and MSDOS.SYS files. On an Linux system, this is often the LILO or Linux loader "program." Although IO.SYS and MSDOS.SYS are "real" files that you can look at and even remove if you want to, the LILO program is not. The LILO program is part of the partition, but not part of the file system; therefore, it is not a "real" file. Regardless of what program is booting your system and loading the kernel, it is generally referred to as a "boot loader".

Often, LILO is installed in the master boot block of the hard disk itself. Therefore, it will be the first code to run when your system is booted. In this case, LILO can be used to start other operating systems. On one machine, I have LILO start either Windows 95 or one of two different versions of Linux.

In other cases, LILO is installed in the boot sector of a given partition. In this case, it is referred to as a "secondary" boot loader and is used just to load the Linux installed on that partition. This is useful if you have another operating system such as OS/2 or Windows NT and you use the boot software from that OS to load any others. However, neither of these was designed with Linux in mind. Therefore, I usually have LILO loaded in the master boot block and have it do all the work.

Assuming that LILO has been written to the master boot record and is, therefore, the master boot record, it is loaded by the system BIOS into a specific memory location 0x7C00 and then executed. The primary boot loader then uses the system BIOS to load the secondary boot loader into a specific memory 0x9B000. The reason that the BIOS is still used at this point is that by including the code necessary to access the hardware, the secondary boot loader would be extremely large at least by comparison to its current size. Furthermore, it would need to be able to recognize and access different hardware types such as IDE and EIDE, as well as SCSI, and so forth.

This limits LILO, because it is obviously dependant on the BIOS. As a result, LILO and the secondary boot loader cannot access sectors on the hard disk that are above 1023. In fact, this is a problem for other PC-based operating systems, as well. There are two solutions to this problem. The original solution is simply to create the partitions so that the LILO and the secondary boot loader are at cylinder 1023 or below. This is one reason for the moving the boot files into the /boot directory which is often on a separate file system, that lies at the start of the hard disk.

The other solution is something called "Logical Block Addresses" LBA. With LBA, the BIOS "thinks" there are less sectors than there actually are. Details on LBA can be found in the section on hard disks.

Contrary to common belief, it is actually the secondary boot loader that provides the prompt and accepts the various options. The secondary boot loader is what reads the /boot/map file to determine the location of kernel image to load.

You can configure LILO with a wide range of options. Not only can you boot with different operating systems, but with Linux you can boot different versions of the kernel as well as use different root file systems. This is useful if you are a developer because you can have multiple versions of the kernel on a single system. You can then boot them and test your product in different environments. We'll go into details about configuring LILO in the section on Installing your Linux kernel.

In addition, I always have three copies of my kernel on the system and have configured LILO to be able to boot any one of them. The first copy is the current kernel I am using. When I rebuild a new kernel and install it, it gets copied to **/vmlinuz.old**, which is the second kernel I can access. I then have a copy called **/vmlinuz.orig**, which is the original kernel from when I installed that particular release. This, at least, contains the drivers necessary to boot and access my hard disk and CD-ROM. If I can get that far, I can reinstall what I need to.

Typically on newer Linux versions, the kernel is no longer stored in the root directory, but rather in the /boot directory. Also, you will find that it is common that the version number of the respective kernel is added onto the end. For example, /boot/vmlinuz.2.4.18, which would indicate that this kernel is version 2.4.18. What is important is that the kernel can be located when the system boots and not what it is called.

During the course of this writing this material, I often had more than one distribution of Linux installed on my system. It was very useful to see whether the application software provided with one release was compatible with the kernel from a different distribution. Using various options to LILO, I could boot one kernel but use the root file system from a different version. This was also useful on at least one occasion when I had one version that didn't have the correct drivers in the kernel on the hard disk and I couldn't even boot it.

Once your system boots, you will see the kernel being loaded and started. As it is loaded and begins to execute, you will see screens of information flash past. For the uninitiated, this is overwhelming, but after you take a closer look at it, most of the information is very straightforward.

Once you're booted, you can see this information in the file **/usr/adm/messages**. Depending on your system, this file might be in **/var/adm** or even **/var/log**, although **/var/log** seems to be the most common, as of this writing. In the messages file, as well as during the boot process, you'll see several types of information that the system logging daemon syslogd is writing. The syslogd daemon usually continues logging as the system is running, although you can turn it off if you want. To look at the kernel messages messages after the system boots, you can use the **dmesg** command.

The general format for the entries is:

time hostname program: message

wheretime is the system time when the message is generated, hostname is the host that generated the message, program is the program that generated the message, and message is the text of the message. For example, a message from the kernel might look like this:

```
May 13 11:34:23 localhost kernel ide0: do_ide_reset: success
>
```

As the system is booting, all you see are the messages themselves and not the other information. Most of what you see as the system boots are messages from kernel, with a few other things, so you would see this message just as

```
ide0: do_ide_reset: success>
```

Much of the information that the syslogd daemon writes comes from device drivers that perform any initialization routines. If you have hardware problems on your system, this is *very* useful information. One example I encountered was with two pieces of hardware that were both software-configurable. However, in both cases, the software wanted to configure them as the same IRQ. I could then change the source code and recompile so that one assigned a different IRQ.

You will also notice the kernel checking the existing hardware for specific capability, such as whether an FPU is present, whether the CPU has the hlt halt instruction, and so on.

What is logged and where it is logged is based on the **/etc/syslog.conf** file. Each entry is broken down into facility.priority, where facility is the part of the system such as the kernel or printer spooler and security and priority indicate the severity of the message. The facility.priority ranges from none, when no messages are logged, to emerg, which represents very significant events like kernel panics. Messages are generally logged to one file or another, though emergency messages should be displayed to everyone usually done by default. See the syslog.conf man-page for details.

One last thing that the kernel does is start the init process, which reads the **/etc/inittab** file. It looks for any entry that should be run when the system is initializing the entry has a sysinit in the third field and then executes the corresponding command. I'll get into details about different run-levels and these entries shortly.

The first thing init runs out of the inittab is the script **/etc/rc.d/rc.sysinit** , which is similar to the bcheckrc script on other systems. As with everything else under **/etc/rc.d**, this is a shell script, so you can take a look at it if you want. Actually, I feel that looking through and becoming familiar with which scripts does what and it what order is a good way of learning about your system.

Among the myriad of things done here are checking and mounting file systems, removing old lock and PID files, and enabling the swap space.

Note that if the file system check notes some serious problems, the rc.sysinit will stop and bring you to a shell prompt, where you can attempt to clean up by hand. Once you exit this shell, the next command to be executed aside from an echo is a reboot. This is done to ensure the validity of the file systems.

Next, init looks through inittab for the line with initdefault in the third field. The initdefault entry tells the system what run-level to enter initially, normally run-level 3 without X Windows or run-level 5 with X Windows. Other systems have the default run-level 1 to bring you into single-user or maintenance mode. Here you can perform certain actions without

worrying users or too many other things happening on your system. Note: You can keep users out simply by creating the file /etc/nologin. See the nologin man-page for details.

What kind of actions can you perform here? The action with the most impact is adding new or updating software. Often, new software will affect old software in such a way that it is better not to have other users on the system. In such cases, the installation procedures for that software should keep you from installing unless you are in maintenance mode.

This is also a good place to configure hardware that you added or otherwise change the kernel. Although these actions rarely impact users, you will have to do a kernel rebuild. This takes up a lot of system resources and degrades overall performance. Plus, you need to reboot after doing a kernel rebuild and it takes longer to reboot from run-level 3 than from run-level 1.

If the changes you made do not require you to rebuild the kernel say, adding new software, you can go directly from single-user to multi-user mode by running **`init 3`**. The argument to init is simply the run level you want to go into, which, for most purposes, is run-level 3. However, to shut down the system, you could bring the system to run-level 0 or 6. See the init man-page for more details.

Init looks for any entry that has a 3 in the second field. This 3 corresponds to the run-level where we currently are. Run-level 3 is the same as multi-user mode.

Within the inittab, there is a line for every run level that starts the script **`/etc/rc.d/rc`**, passing the run level as an argument. The **`/etc/rc.d/rc`** script, after a little housekeeping, then starts the scripts for that run level. For each run level, there is a directory underneath **`/etc/rc.d`**, such as rc3.d, which contains the scripts that will be run for that run level.

In these directories, you may find two sets of scripts. The scripts beginning with K are the kill scripts, which are used to shutdown/stop a particular subsystem. The S scripts are the start scripts. Note that the kill and start scripts are links to the files in **`/etc/rc.d/init.d`**. If there are K and S scripts with the same number, these are both linked to the same file.

This is done because the scripts are started with an argument of either start or stop. The script itself then changes its behavior based on whether you told it to start or stop. Naming them something slightly different allows us to start only the K scripts if we want to stop things and only the S scripts when we want to start things.

When the system changes to a particular run level, the first scripts that are started are the K scripts. This stops any of the processes that should not be running in that level. Next, the S scripts are run to start the processes that should be running.

Let’s look at an example. On most systems, run-level 3 is *almost* the same as run-level 2. The only difference is that in run-level 2, NFS is not running. If you were to change from run-level 3 to run-level 2, NFS would go down. In run-level 1 maintenance mode, almost everything is stopped.

6.1.2 Run Levels

Most users are only familiar with two run states or run levels. The one that is most commonly experienced is what is referred to a multiuser mode. This is where logins are enabled on terminals when the network is running and the system is behaving "normally." The other run level is system maintenance or single-user mode, when only a single user is on the system (root), probably doing some kind of maintenance tasks. Although it could be configured to allow logins by other users, usually the system is so configured that only one login is allowed on the system console.

On every system that I have encountered, Linux will automatically boot into run-level 3. This is the normal operating mode. To get to a lower run level (for example, to do system maintenance), the system administrator must switch levels manually.

It is generally said that the "system" is in a particular run-level. However, it is more accurate to say that the init process is in a particular run level, because init determines what other processes are started at each run-level.

In addition to the run levels most of us are familiar with, there are several others that the system can run in. Despite this fact, few of them are hardly ever used. For more details on what these run levels are, take a look at the init man-page.

The system administrator can change to a particular run level by using that run level as the argument to init. For example, running init 2 would change the system to run-level 2. To determine what processes to start in each run level, init reads the **`/etc/inittab`** file. This is defined by the second field in the **`/etc/inittab`** file. Init reads this file and executes each program defined for that run level in order. When the system boots, it decides what run level to go into based on the initdefault entry in **`/etc/inittab`**.

The fields in the inittab file are:

id	unique identity for that entry
rstate	run level in which the entry will be processed
action	tells init how to treat the process specifically
process	what process will be started

One thing I need to point out is that the entries in inittab are not run *exactly* according to the order in which they appear. If you are entering a run level other than S for the first time since boot-up, init will first execute those entries with a boot or bootwait in the *third* column. These are those processes that should be started before users are allowed access to the system, such as checking then mounting the status of the file systems.

In run-level 3, the /sbin/mingetty process is started on the terminals specified. The getty process gives you your login: prompt. When you have entered your logname for the first time, getty starts the login process, which asks you for your password. If your password is incorrect, you are prompted to input your logname again. If your password is correct, then the system starts your "login shell. " Note that what gets started may not be a shell at all, but some other

program. The term "login shell" is the generic term for whatever program is started when you login. This is defined by the *last* field of the corresponding entry in /etc/passwd.

Keep in mind that you can move in either direction, that is, from a lower to higher run level or from a higher to lower run level without having to first reboot. init will read the inittab and start or stop the necessary processes. If a particular process is not defined at a particular run level, then init will kill it. For example, assume you are in run-level 3 and switch to run-level 1. Many of the processes defined do not have a 1 in the second field. Therefore, when you switch to run-level 1, those processes and all their children will be stopped.

If we look at the scripts in rc1.d, we see there all the scripts are kill scripts, with the exception of one start script. It is this start script that actually kills all the processes. It does exec init -t1 S, which brings the system into maintenance mode in one (-t1) minute.

To shutdown the system immediately, you could run

```
init 0
```

which will bring the system immediately into run-level 0. As with run-level 1, there is only one start script for run-level 0. It is this script that kills all the processes, unmounts all the file systems, turns off swap, and brings the system down.

After it has started the necessary process from inittab, init just waits. When one of its "descendants" dies (a child process of a child process of a child process, etc., of a process that init started), init rereads the inittab to see what should be done. If, for example, there is a respawn entry in the third field, init will start the specified process again. This is why when you log out, you immediately get a new login: prompt.

Because init just waits for processes to die, you cannot simply add an entry to inittab and expect the process to start up. You have to tell init to reread the inittab. However, you can *force* init to reread the inittab by running init (or telinit) Q.

In addition to the run levels we discussed here, several more are possible. There are three "pseudo" run-levels a, b, and c. These are used to start specific programs as needed or "on demand". Any listed in inittab with the approapriate run-level will be started, however no actual run-level change occurs. If you’re curious about the details, take a look at the init(8) man-page or the section on init-scripts.

Action	Meaning
boot	Executed during system boot.
bootwait	Executed during system boot, but init waits until they have completed.
initdefault	The default run level init starts after the system boots.
ondemand	Executed when one of the "on demand" run levels is called (a,b, and c)
powerwait	Executed when the system power fails. Init will wait until the command completes.
powerfail	Executed when the system power fails, but init will not wait for completion.
powerokwait	Executed when init is informed that power has been restored.
powerfailnow	Executed when init is informed that the external battery is empty.
resume	Executed when init is told by the kernel that "Software Suspend"
sysinit	Executed during system boot before any boot or bootwait entries.
respawn	Restarted if the processes stops.
wait	Started once when the specific run-level is entered and init waits for completion.
once	Started once when the specific run-level is entered but init does not wait for completion.
ctrlaltdel	Execute when someone on the system console presses CTRL-ALT-DEL.

Table - List of inittab actions.

If necessary, you can add your own entries into **`/etc/inittab`**. However, what is typically done is that init-scripts are added to the appropriate directory for the run-level where you want to start it. Depending on your Linux distribution, you could simply copy it into /etc/rc.d and use the appropriate admin tool, like Yast2 to add the script to the appropriate directory. For more details see the section on init-scripts.

Note however, that simply changing **`/etc/inittab`** is not enough. You need to tell the init process to re-read it. Normally init will re-read the file when it changes run levels or by sending it a hangup signal with

```
kill -HUP <PID_OF_ID>.
```

Also running the command

```
telinit q
```

will tell init to reread it.

Be **extremely** careful if you edit the **`/etc/inittab`** file by hand. An editing mistake could prevent your system from booting into a specific run level. If you use the boot, bootwait or sysinit actions, you could prevent your system from booting at all. Therefore, like with any system file it is good idea to make a backup copy first. If you make a mistake that prevents a particular program from starting, and the action is respawned, init might get caught in a loop. That is, init tries to start the program, cannot, for whatever reason and then tries to start it

again. If init finds that it is starting the program more than 10 times within 2 minutes, it will treat this as an error and stops trying. Typically you will get messages in the system log that the process is "respawning too rapidly".

6.1.3 Init Scripts

If you were to just install the Linux operating system on your hard disk, you would not be able to do very much. What actually makes Linux so useful is all of the extra things which are brought with it. This is essentially true for every operating system.

What makes Linux so useful as well as powerful are all the of services, which are generally referred to as daemons. These daemons typically run without user intervention providing everything from printing to file services to Web pages and beyond. Because they are not part of the operating system proper they are normally loaded separately from the kernel. Although many of these services could be made part of the kernel, they are mostly separate programs. Because they are separate programs something needs to configure and start them.

In most cases, simply installing a particular package is sufficient to activate the appropriate daemon. However, there are times when you need to make changes to how these demons behave, which often means changing the way the program starts up. In order to be able to do that, you obviously need to know just how and where these daemons are started in the first place. That’s exactly what we’re going to talk about here.

Once the kernel is loaded, one of the last things it does is to start the init process. The job of the init process (or simply init) is to start all of the daemons at the appropriate time. What the appropriate time is depends on a number of different things. For example, you may be performing administrative tasks and you do not want certain daemons to be running. Although you can stop those demons you do not need, the system provides a mechanism to do this automatically.

To understand this mechanism we need to talk about something called "run states" or "run levels". Most users (and many administrators, for that matter) are familiar with only one run level. This is the run level in which the system is performing all of its normal functions. Users can login, submit print jobs, access Web pages, and do everything else one would expect. This run level is commonly referred to as multiuser mode. In contrast, maintenance or single user mode is normally recommended for administrative tasks.

Each run level is referred to by its number. When the system is not doing anything, that is the system is stopped, this is run level 0. Single user mode is run-level 1. Multiuser mode is actually multiple runs levels. Depending on which distribution or which version of Unix you ahve, this can be run-level 2, run-level 3 **and** run-level 5. Most Linux systems automatically booting into run-level 3 when the system starts. Run level 2 is very similar to run level 3, although a number of things do not run in level 2. In fact, on some systems (SCO UNIX for example), run level 2 is the standard multi-user mode. Run-level 5 is where the GUI starts automatically. (For more details on the run levels, take a look at the init(8) man-page.)

Like many other aspects of the system, init has its own configuration file: **/etc/inittab** (see the table below). This file contains the init table (inittab), which tells it init what to do and when to do it. Each activity init does is represented by a single line in the inittab, which consists of four entries, separated by a colon. The first field is a unique identifier for that entry, which enables init to keep track of each daemon as it runs. The second field is the run level in which each particular entry is run.

The third entry is the action, which tells init how to behave in regard to this entry. For example, some entries are only processed when the system boots. Others are automatically re-started should that particular process stop (such as terminal logins). The last entry is what program will be started and often a number of options for that program.

If you take look in inittab on your system you may notice something peculiar. More than likely, you are not going to find any entries for the system demons we have been talking about. The reason is quite simply that the daemons are not started through the inittab, but rather through scripts which are started from the inittab. These scripts we see as the entries labeled l0 through l6, for run levels 0 through 6 (the letter "ell", not the number one).

In the example below, the "action" is that init waits until the program has terminated before continuing on and processing other entries for this run level. This also means that the entry will only be processed once as the system enters that particular one level.

The key to all of this is the program which is run for each run level. In every case, it is the shell script rc, which is given the appropriate run level as an argument. This script is often called the "run level master script" as it is responsible for loading all of the other init scripts. Where this script lies and what it is called will be different for different Linux distributions. Under older versions of SuSe it in **/etc/rc.d**, but now it's in **/etc/init.d/**. Under Caldera the script resides under **/etc/rc.d**. Note that starting with version 8.0, SuSe also has an **/etc/rc.d** directory, which is actually a symbolic link to **/sbin/init.d**.

Not just the location of the script is different between distributions, but so is the actual code. However, the basic functionality is generally the same. That is, to start other scripts which finally start the daemons we have been talking about all along.

One of the key aspects is how the system determines which daemon to start in which run level. As you might guess, this is accomplished through the run-level that is passed as an argument to the RC script. At least that's part of it. In addition, the system needs a list of which scripts should be started in which run level. This is accomplished not by a text file, but rather by separating the programs or scripts into different directories, one for each run level.

If you look in the **/sbin/init.d** or **/etc/rc.d** directory you'll see a number of subdirectories of the form rc#.d, where # is a particular run level. For example, the directory rc3.d is for run level 3. Within the subdirectories are not the actual scripts, as you might have guessed, but rather symbolic links to the actual scripts. The primary reason for this is that a script can be started in more than one run level. If the files were not links, but rather copies, any change would have to be made to every copy. The reason they are symbolic links, is that they may point to files on other file systems which is only possible by using symbolic links.

With SuSe, the **/sbin/init.d** directory is also where the real scripts reside. On Caldera, the scripts reside under **/etc/rc.d/init.d**.

At first glance, the filenames may be a little confusing. Although it is fairly simple to figure out what daemon is started by looking at the name, the way these links are named takes a little explanation.

As you might guess, the link ending in "apache" points to the script which starts the Apache Web server.

However, you'll see there are two files with this ending. The really odd thing is that both of these links point to the exact same file. So, what's the deal?

Part of the explanation lies in the first letter of each of these links. As you see, each starts with either the letter S or the letter K. Those which begin with the letter S are used to start the particular service and those which begin with the letter K are used to stop or kill that same service.

That leaves us with just the numbers. These are used to define the order in which the scripts are run. When the files are listed, they automatically appear in numerical order. In this way, the system can ensure the scripts are run in the correct order. For example, you do not want to start the Apache Web server before you start the network. Therefore, the linked used to start the network is S05network whereas the link used to start Apache is S20apache as S05 comes before S20 no matter what comes afterwards.

Note also, the same applies when the system shuts down. K20apache is used to shut down the Apache server and K40network is used to shut down network. As in the first case, the network is not shutdown until after Apache has.

It is interesting to note that this system could work even if the name of the link consisted of just S or K and the appropriate number. That is, it would still work if the link told us nothing of the service being started. There is actually more to it than making things simpler for us non-computers. Having the names at the end allows the system to avoid unnecessary the unnecessary stopping and starting of the various services. When a lower level is entered, only those of services are started which were not started in previous run level. When leaving a run level, the only services that are stopped are those that are not started in the new level.

Let's look at an example. In the directory **/etc/init.d/rc3.d** (for run level 3), there are links used to both start and stop the network. However, this means the network will always be re-started when moving from run level 1 to run level 3. This also means the network will always be stopped when moving from run level 3 to run level 1. On the other hand, both links exist in rc.2 (for run level 2). Therefore, when leaving either run level 2 or 3 and moving to the other, the network is not stopped as there is a start link for it in the new run level. When entering the new run level, the network is not started, as there was already a start link for the previous level. However, in moving from a run level when network is running (e.g. 2,3 or 5) to run level 1, the network is stopped because there is no link to start the network in run level 1.

We're not done yet.

Since the links to both start and stop a service can be to the exact same file, the script needs some way of knowing whether it should start or stop the service. This is done by passing an argument to the script: start to start the service and stop to stop the service (simple, huh?). Inside each script, this argument is read (typically $1) and different activities are performed based on what the argument was.

Note that for many scripts, you can pass other arguments than just start and stop. For example, one common argument is restart. As its name implies, this is used to stop then start the service again, in other words, restart a running service. Many will also accept the argument status, which is used to deliver status information about that service.

```
# Default runlevel.
id:3:initdefault:

# System initialization.
si::sysinit:/etc/rc.d/rc.modules default
bw::bootwait:/etc/rc.d/rc.boot

# What to do in single-user mode.

~1:S:wait:/etc/rc.d/rc 1
~~:S:wait:/sbin/sulogin

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
```

Figure - Excerpts from the /etc/inittab file

On some systems, the scripts we have talked about so far are not the only scripts which are started when the system boots. Remember that init reads the inittab to find out what to do, so there are any number of things that "could" be started through the inittab, as compared to the rc-scripts. Even so, for people who are used to other versions of UNIX, the inittab looks pretty barren.

One type of script that is often run from the inittab deals with system initialization. For example, the boot script, which is found directly in **`/sbin/init.d`**. The entry in the inittab might look like this:

```
si:I:bootwait:/sbin/init.d/boot
```

The run level this script runs in is "I", which is not a traditional run level, but used by some distributions (i.e. SuSe) to indicate system initialization. However, because the action is bootwait, the run-level field is ignored. Bootwait means that this entry will be processed while the system boots, and init will wait until the command or script has completed.

In this case, the script is **/sbin/init.d/boot**, which performs basic system initialization such as starting the bdflush daemon (which writes dirty buffers to the disk), checking the filesystems (with fsck), mounting filesystems, starting the kernel module daemon (kerneld), and many other things. Other versions (as well as other UNIX dialects) may have several different entries in the inittab that combine to do the same work as the /sbin/init.d/boot script under SuSe Linux.

The counterpart to the **/sbin/init.d/boot** script **/sbin/init.d/halt**. These are the procedures that are carried out when the system is brought down. In general, these are the reverse of the procedures in the boot scripts, such as stopping kerneld and unmounting filesystems.

SuSe also uses the system configuration file /etc/rc.config. This file contains a large number of variables that are used to configure the various services. Reading this file and setting the variables is one of the first things done by the script **/sbin/init.d/rc**. The counterpart to this file on Caldera is **/etc/syconfig/daemons**. Instead of a single configuration file, you will find separate files for a number of different daemons.

Creating your own init scripts

Sometimes the scripts your particular distribution provides are not sufficient and you need to add your own. On a number of systems where I have needed to add my own system services, I have needed to create my own init scripts. The method that works on any system is to simply follow the conventions used by your distribution.

SuSe has realized the need for creating your own init scripts, so has provided a template for you. This is the file **/sbin/init.d/skeleton** and as its name implies, is a "skeleton" init script. In its default state, this is a completely runnable init script. At the same time it is completely useless as there is no daemon behind it. Instead, you simply uncomment the lines you need, change the name of the daemon or service and you are ready to run.

6.1.4 LILO-The Linux Loader

In the section on the boot process, we briefly discussed the Linux boot loader LILO. (Short for *LI*nux *LO*ader). This is basically a set of instructions to tell the operating system how to boot. These instructions include what operating system to boot and from what partition, as well as a number of different options. If LILO is installed in your master boot record, it can be used to boot basically any operating system that you can install on that hardware. For example, on my machine, I have had LILO boot various Windows versions (including NT), SCO Unix, Sun Solaris, and, of course, Linux. Actually, most of the work was done by the boot loader of the respective operating system, but LILO was used to load start the boot process.

In this section we are going to talk about some of the basics of LILO from a user's perspective. In the section on Installing your Linux kernel, we'll go into more details of configuring and installing LILO.

When your system reaches the point where LILO is executed, you are usually given the prompt:

```
LILO:
>
```

By simply pressing the enter key, LILO will execute it's default instructions, which usually means loading and executing a Linux kernel. However, starting Linux is not a requirement. In the before time, when I typically worked on Windows, rather than Linux, I had LILO as my boot loader, but it booted Windows 95 by default. Either I pressed the enter key at the LILO prompt or simply waited until LILO had reached the configured timeout (which I configured to be 10 seconds).

In order to boot different operating systems, you need to be able to tell LILO what to boot. This is done by simply inputting the appropriate text at the LILO prompt. This text is configured in the LILO configuration file (**/etc/lilo.conf**). A problem arises three months later when you have forgotten what text you used. Fortunately, to get a list of available options, all you need to do is press the TAB key, which will display the different texts. For example, I had three, which were labeled "win95", "linux" and "linuxgood". The win95 was the default (before I knew better), linux started my current kernel and linuxgood was a kernel that I had compiled with just the basic options that I knew worked and I used it as a failsafe should something go wrong when installing a new kernel. Interestingly enough, SuSE added their own LILO entry in the meantime, which they simply called "failsafe" with the same purpose as my entry.

In addition to accepting the tag, or label, for a specific entry, you can pass configuration options to LILO directly at the prompt. One thing I commonly pass is the location of the root filesystem. I used to have a couple of different Linux distributions on my system, particularly when a new version came out.I would install the new version on a different partition to make sure things worked correctly. I could then boot from either the new or old kernel and select which root filesystem I wanted. This might be done like this:

```
linux root=/dev/hda6
```

Here /dev/hda6 is the partition where my root filesystem is. Note that LILO does not do anything with these options, instead they are passed to the kernel. LILO is not very smart, but knows enough to pass anything given it at the LILO prompt to the kernel. You can also pass options to tell the kernel that the root filesystem is read-only (root=/dev/hda6,ro) or read-write (root=/dev/hda6,rw).

Another useful option is the keyword "single". This tells the kernel to boot into "single-user mode", which is also referred to as "maintenance mode". As the names imply, only a single user can log into the system and it is used to perform system maintenance.

If it runs into problems while booting, LILO will provide you information about what went wrong (albeit not as obviously as you might hope). When loading, the letters of "LILO" are not printed all at once, but rather as each phase of the boot process is reached. Therefore, you can figure out where the boot process stopped by how much of the word LILO is displayed. The following table shows you the various stages and what possible problems could be.

Characters	Description
none	LILO has not yet started. Either it was not installed or the partition is not active
L *errorcode*	The first stage boot loader has been loaded and started. However, the second stage boot loader cannot be loaded. The *errorcode* typically indicates a media problem, such as a hard disk error or incorrect hard disk geometry.
LI	The second stage boot loader was loaded, but could not be executed. Either a geometry mismatch has occurred or boot/boot.b was moved without running the map installer.
LIL	Second stage boot loader starts, but cannot load the descriptor table from the map file. Typcially a media failure or by a geometry mismatch.
LIL?	Second stage boot loader loaded at an incorrect address. Typically caused by a geometry mismatch or by moving /boot/boot.b without running the map installer.
LIL-	Descriptor table is corrupt. Typically caused by either a geometry mismatch or by moving /boot/boot.b without running the map installer.
LILO	Everything successfully loaded and executed.

Table - LILO boot stages and possible problems.

6.1.5 Stopping the System

For those of you who hadn't noticed, Linux isn't like DOS or Windows. Despite the superficial similarity at the command prompt and similarities in the GUI, they have little in common. One very important difference is the way you stop the system.

In DOS or Windows 95/98/ME, you are completely omnipotent. You know everything that's going on. You have complete control over everything. If you decide that you've had enough and flip the power switch, you are the only one doing so will effect. However, with dozens of people working on an Linux system and dozens more using its resources, simply turning off the machine is not something you want to do. Despite the fact that you will annoy quite a few people, it can cause damage to your system, depending on exactly what was happening when you killed the power. (Okay, you could also create problems with a DOS system, but with only one person, the chances are less likely).

On a multi-user system like Linux, many different things are going on. You many not see any disk activity, but the system may still have things its buffers which are waiting for the chance to write to the hard disk. If you turn off the power before this data is written, what is on the hard disk may be inconsistent.

Normally, pressing Ctrl-Alt-Del *will* reboot your system. You can prevent this by creating the file **`/etc/shutdown.allow`**, which contains a list (one entry per line) of users. If this file exists, the system will first check whether one of the users listed in shutdown.allow is logged in on the system console. If none are, you see the message

```
shutdown: no authorized users logged in.
```

To make sure that things are stopped safely, you need to shut down your system "properly." What is considered proper can be a couple of things, depending on the circumstances. Linux provides several tools to stop the system and allows you to decide what is proper for your particular circumstance. Flipping the power switch is *not* shutting down properly.

Note that the key combination Ctrl-Alt-Del is just a convention. There is nothing magic about that key combination, other than people are used to it from DOS/Windows. By default, the combination Ctrl-Alt-Del is assigned to the special keymap "Boot". This is typically defined by default in the file **/usr/src/linux/drivers/char/defkeymap.map**, which is the keyboard mapping the kernel uses when it boots. However, you can use the loadkeys program to change this if you need to.

If necessary, you could define that the combination Ctrl-Alt-Del is not assigned to anything, therefore it would not shutdown your system. However, should the system get stuck in a state that you cannot correct, shutting it down with Ctrl-Alt-Del is often the only safe alternative (as compared with simply flipping the power switch.)

When you press the "boot" key combination, the init program is sent the signal SIGINT. What init does will depend on how the **/etc/inittab** is configured. In the section on run levels, we talked about the various actions in **/etc/inittab** that tell init what to do when the key combination Ctrl-Alt-Del is pressed (one being ctraltdel). On my system it is defined as "/sbin/shutdown -r -t 4 now", which says to run the shutdown command immediately (now) and reboot (-r), waiting four seconds between the time the warning message is sent and the shutdown procedure is started (-t 4).

The first two tools to stop your system are actually two links in **/sbin**: **halt** and **reboot**, that link to the same file. If either of these is called and the system is not in run-level 0 or 6, then **shutdown** (also in **/sbin**) is called instead.

Running shutdown is really the safest way of bringing your system down, although you *could* get away with running init 0. This would bring the system down, but would not give the users any warning. Shutdown can be configured to give the users enough time to stop what they are working on and save all of their data.

Using the shutdown command, you have the ability not only to warn your users that the system is going down but also to give them the chance to finish up what they were doing. For example, if you were going to halt the system in 30 minutes to do maintenance, the command might look like this:

```
shutdown –h +30 "System going down for maintenance. Back up after lunch."
```

This message will appear on everyone’s screen immediately, then at increasing intervals, until the system finally goes down.

If you have rebuilt your kernel or made other changes that require you to reboot your system, you can use shutdown as well, by using the -r option.

Option	Description
-d	Do not write to the **/var/log/wtmp**.
-f	Force a reboot, i.e. do not call shutdown.
-i	Shutdown the network interfaces before halting or rebooting.
-n	Do not sync (write data to disk) before halting or rebooting.
-p	Power off after shutdown.
-w	Do not actually stop the system, just write to /var/log/wmtp.

Table - Options to halt and reboot.

Option	**Description**
-c	Cancel a shutdown that is in progress.
-f	Don't run **fsck** when the system reboots (i.e. a "fast" reboot).
-F	Force **fsck** on reboot.
-h	Halt the system when the shutdown is completed.
-k	Send a warning message, but do not actually shutdown the system.
-n	Shutdown without calling init. DEPRECATED.
-r	Reboot the system after shutdown.
-t *seconds*	Seconds to wait before starting the shutdown.
-z	Shutdown using "software suspend".

Table - Options to shutdown.

6.2 User Accounts

Users gain access to the system only after the system administrator has created *user accounts* for them. These accounts are more than just a user name and password; they also define the environment the user works under, including the level of access he or she has.

Users are added to Linux systems in one of two ways. You could create the necessary entries in the appropriate file, create the directories, and copy the start-up files manually. Or, you could use the adduser command, which does that for you.

Adding a user to a Linux system is often referred to as "creating a user" or "creating a user account". The terms "user" and "user account" are often interchanged in different contexts. For the most part, the term "user" is used for the person actually working on the system and "user account" is used to refer to the files and programs that create the user's environment when he or she logs in. However, these two phrases can be interchanged and people will know what you are referring to.

When an account is created, a shell is assigned along with the default configuration files that go with that shell. Users are also assigned a home directory, which is their default directory when they login, usually in the form /home/<username>. Note that the parent of the user's home directories may be different.

When user accounts are created, each user is assigned a User Name (login name or logname), which is associated with a User ID (UID). Each is assigned to at least one group, with one group designated as their *login group*. Each group has an associated Group ID (GID). The UID is a number used to identify the user. The GID is a number used to identify the login group of that user. Both are used to keep track of that user and determine what files he or she can access.

In general, programs and commands that interact with us humans report information about the user by logname or group name. However, most identification from the operating system's point of view is done through the UID and GID. The UID is associated with the user's logname. The GID is associated with the user's login group. In general, the group a user is a part of is only used for determining access to files.

User accounts are defined in **/etc/passwd** and groups are defined in **/etc/group**. If you look on your system, you will see that everyone can read both of these files. Years ago, my first reaction was that this was a security problem, but when I was told what this was all about, I realized that this was necessary. I was also concerned that the password be accessible, even in encrypted format. Because I know what my password is, I can compare my password to the encrypted version and figure out the encryption mechanism, right? Nope! Its not that easy.

At the beginning of each encrypted password is a *seed*. Using this seed, the system creates the encrypted version. When you login, the system takes the seed from the encrypted password and encrypts the password that you input. If this matches the encrypted password, you are allowed in. **Nowhere** on the system is the **unencrypted** password stored, nor do any of the utilities or commands generate it.

Next, lets talk about the need to be able to access this information. Remember that the operating system knows only about numbers. When we talked about operating system basics, I mentioned that the information about the owner and group of a file was stored as a number in the inode. However, when you do a long listing of a file (ls -l), you don't see the number, but rather, a name. For example, if we do a long listing of **/bin/mkdir**, we get:

```
-rwxr-xr-x   1 root     root         7593 Feb 25  1996 /bin/mkdir>
```

The entries are:

permissions links owner group size date filename

Here we see that the owner and group of the file is root. Because the owner and group are stored as numerical values in the inode table, the system *must* be translating this information before it displays it on the screen. Where does it get the translation? From the **/etc/passwd** and **/etc/group** files. You can see what the "untranslated" values are by entering

ls -ln /bin/mkdir

which gives us:

```
-rwxr-xr-x   1 0     0         7593 Feb 25  1996 /bin/mkdir>
```

If we look in **/etc/passwd**, we see that the 0 is the UID for root, and if we look in **/etc/group**, we see that 0 is also the GID for the group root, which are the numbers we got above. If the **/etc/passwd** and **/etc/group** files were not readable by everyone, then no translation could be made like this without some major changes to most of the system commands and utilities.

On a number of occasions, I have talked to customers who claimed to have experienced corruption when transferring files from one system to another. Sometimes it's with cpio, sometimes it's tar. In every case, files have arrived on the destination machine and have had either "incorrect" owners or groups and sometimes both. Sometimes, the "corruption" is so bad that there are no names for the owner and group, just numbers.

Numbers, you say? Isn't that how the system stores the owner and group information for the files? Exactly. What does it use to make the translation from these numbers to the names that we normally see? As I mentioned, it uses /etc/passwd and **/etc/group**. When you transfer files from one system to another, the only owner information that is transferred are the numbers. When the file arrives on the destination machine, weird things can happen. Lets look at an example.

At work, my user name was jimmo and I had UID 12709. All my files were stored with 12709 in the owner field of the inode. Lets say that I create a user on my machine at home, also named jimmo. Because there are far fewer users on my system at home than at work, jimmo ended up with UID 500. When I transferred files from work to home, the owner of all "my" files was 12709. That is, where there normally is a name when I do a long listing, there was the number 12709, not jimmo.

The reason for this is that the owner of the file is stored as a number in the inode. When I copied the files from my system at work, certain information from the inode was copied along with the file, including the owner. *Not* the user's name, but the numerical value in the inode. When the files were listed on the new system, there was no user with UID 12709, and therefore no translation could be made from the number to the name. The only thing that could be done was to display the number.

This makes sense because what if there were no user jimmo on the other system? What value should be displayed in this field? At least this way there is some value and you have a small clue as to what is going on.

To keep things straight, I had to do one of two things. Either I create a shell script that changed the owner on all my files when I transferred them or I figure out some way to give jimmo UID 12709 on my system at home. So I decided to give jimmo UID 12709.

Here, too, there are two ways I can go about it. I could create 12208 users on my system so the 12709th would be jimmo. (Why 12208? By default, the system starts with a UID 500 for

normal users.) This bothered me though, because I would have to remove the user jimmo with UID 500 then create it again. I felt that this would be a waste of time.

The other alternative was to change the system files. Now, there is nothing that Linux provides that would do that. I could change many aspects of the user jimmo; however, the UID was not one of them. After careful consideration, I realized that there was a tool that Linux provided to make the changes: vi. Because this information is kept in simple text files, you can use a text editor to change them. After reading the remainder of this chapter, you should have the necessary information to make the change yourself.

One thing I would like to point out is that vi is not actually the tool you should use. Although you could use it, something could happen while you are editing the file and your password file could get trashed. Linux provides you with a tool (that's actually available on many systems) specifically designed to edit the password file: vipw (for "vi password").

What vipw does is create a copy of the password file, which is what you actually edit. When you are finished editing, vipw replaces the /etc/passwd with that copy. Should the system go down while you are editing the file, the potential for problems is minimized. Note that despite its name, the editor that is called is defined by your EDITOR environment variable.

On many systems, the adduser program is used to add users (what else?). Note that when you create a user, you are assigned a value for the UID, usually one number higher than the previously assigned UID. Because adduser is a shell script, you can change the algorithm used, if you really want to.

When the first customer called with the same situation, I could immediately tell him why it was happening, how to correct it, and assure him that it worked.

You can also change a user's group if you want. Remember, however, that all this does is change the GID for that user in **/etc/passwd**. Nothing else! Therefore, all files that were created before you make the change will still have the old group.

You can change your UID while you are working by using the su command. What does su stand for? Well, that's a good question. I have seen several different translations in books and from people on the Internet. I say that it means "switch UID, " as that's what it does. However, other possibilities include "switch user" and "super-user." This command sets your UID to a new one. The syntax is

su <user_name>

where <user_name> is the logname of the user whose UID you want to use. After running the command, you have a UID of that user.

The shortcoming with this is that all that is changed is the UID and GID; you still have the environment of the original user. If you want the system to "pretend" as though you had actually logged in, include a dash (-). The command would then be

su - <user_name>

What is actually happening is that you are running a new shell as that user. (Check the ps output to see that this is a new process.) Therefore, to switch back, you don't need to use su again, but just exit that shell.

We need to remember that a shell is the primary means by which users gain access to the system. Once they do gain access, their ability to move around the system (in terms of reading files or executing programs) depends on two things: permissions and privileges.

In general, there is no need to switch groups. A user can be listed in more than one group in /etc/group and the system will grant access to files and directories accordingly.

Permissions are something that most people are familiar with if they have ever worked on an Linux (or similar) system before. Based on what has been granted, different users have different access to files, programs, and directories. You can find out what permissions a particular file has by doing a long listing of it. The permissions are represented by the first 10 characters on the line. This is something that we covered in a fair bit of detail in the section on shell basics, so there is no need to repeat it here.

Removing users is fairly straightforward. Unfortunately, I haven't found a utility that will remove them as simply as you can create them. Therefore, you will need to do it manually. The simplest way is to use vipw to remove the users entry from /etc/passwd and to remove its home directory and mailbox.

However, this is not necessarily the best approach. I have worked in companies where once a user was created, it was never removed. This provides a certain level of accountability.

Remember that the owner is simply a number in the inode table. Converting this number to a name is done through the entry in **`/etc/passwd`**. If that entry is gone, there can be no conversion. If a new user were to get the UID of an old, removed user, it may suddenly have access to a file that it shouldn't (i.e., a file owned by the old user that it now owns).

Even if no new users get that UID, what do you do if you find an "unowned" file on your system, that is, one with just a number as the owner and without associated entry in /etc/passwd? What you do is up to your company, but I think it is safer to "retire" that user.

You could remove its home directory and mailbox. However, change its password to something like NOLOGIN. This password is shorter than an encrypted password, so it is *impossible* that any input password will encrypt to this. Then change its login shell to something like **`/bin/true`**. This closes one more door. By making it /bin/true, no error message will be generated to give a potential hacker a clue that there is "something" about this account. Alternatively, you could replace the login shell with a message to say that the account has been disabled and the owner should report to have it re-activated. This helps to dissuade would-be hackers.

Another useful tool for thwarting hackers is password shadowing. With this, the encrypted password is not kept in **`/etc/passwd`**, but rather **`/etc/shadow`**. This is useful when someone decides to steal your password file. Why is this a problem? I will get into details about it later, but lets say now that the password file could be used to crack passwords and gain access to the system.

Because you must have the **/etc/passwd** file word-readable to make translations from UID to user name, you cannot protect it simply by changing the permission. However, the **/etc/shadow** password, where the real password is stored, is not readable by regular users and therefore is less of a security risk. (I say "less" because if an intruder gets in as root, all bets are off).

6.2.1 logging in

Users gain access to the system through "accounts." This is the first level of security. Although it is possible to configure applications that start directly on specific terminals, almost everyone has logged into an Linux system at least once. More that likely, if you are one of those people who never login, you never see a shell prompt and are probably not reading this book.

Most Linux systems have a standard login. The figure below shows what the login process looks like. You see the name of the system, followed by a brief message the contents of **/etc/issue** and the login prompt, which usually consists of the system name and the word login. This is a text file, so you can edit it as you please. Because it is read dynamically, the changes will appear the next time someone tries to log in. After the contents of **/etc/issue**, you see the login prompts, such as

```
jmohr!login: >
```

When you login, you are first asked your user name and your password. Having been identified and your password verified, you are allowed access to the system. This often means that the system starts a shell for you. However, many programs can be used in place of a shell.

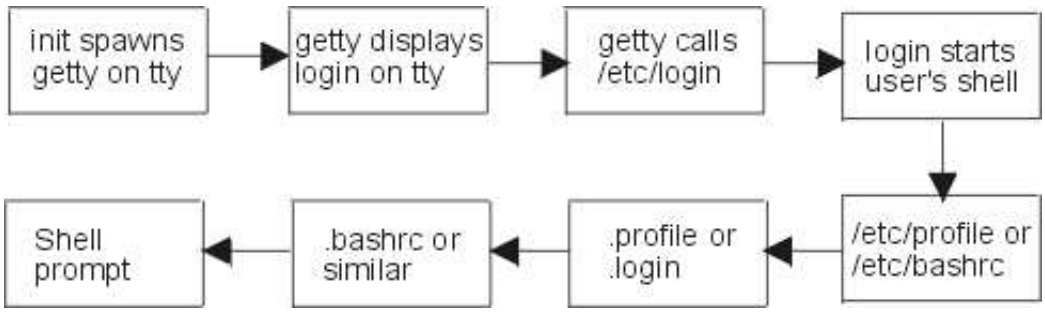


Image - The login process.

One entry in the password file is your home directory, the directory that you have as your current directory when you log in. This is also the place to which the shell returns you if you enter cd with no arguments.

After determining your login shell and placing you in your home directory, the system will set up some systemwide defaults. If you have a Bourne or Bourne Again-shell, these are done through the **/etc/profile** file. If bash is your login shell, the system runs through the commands stored in the .profile in your home directory then the .bashrc file, provided they exist. If you have sh, then there is no equivalent for the .bashrc file. If you have a Z-shell, the system defaults are established in the **/etc/zprofile** file. The system then executes the commands in the **.zshrc** and **.zlogin** files in your home directory, provided they exist.

See the appropriate man-page and the section on shell basics for more details.

During the login process, you are shown several pieces of information about the local system. Before the login prompt, you usually see the contents of the **/etc/issue** file, as I mentioned earlier. After your login is successful, you will normally see a message about the last login and the message of the day. The message of the day is the contents of the file **/etc/motd**.

In some cases, all of this information is bothersome. For example, many businesses have either menus that their users log into or applications that start from their users **.profile** or **.login**. In some cases, the information is of little value.

In some cases, even knowing that this is an UNIX system could be a problem. There are many hackers in the world who would just love the chance to try to crack your security. By not even telling them what kind of system you have, you reduce the amount by which they are tempted. At least, that's one more piece of information that they need to figure out. Therefore, we need a way to disable these messages.

The two obvious ways are by using **/etc/issue** and **/etc/motd**. By default, both of these files contain information about your system. By either changing the contents or removing the files altogether, you can eliminate that source of information.

Another way is the login: prompt itself. Again, by default, this prompt contains the name of your system. This may not concern most system administrators, however, in cases where security is an issue, I might like to disable it. The prompt comes from the **/etc/gettydefs** file. The gettydefs file contains information the getty program uses when it starts the login program on a terminal. The more common lines in the gettydefs file contain an entry that looks like this:

```
@S login:
```

Take a look at the

```
login:>
```

prompt and you will see that it also contains the literal string login: immediately following the name of the system. The name of the system comes from @S. By changing either of the parts or both, you can change the appearance of your login prompt, even removing the name of the system, if you want.

The getty1m man-page contains a list of the different information that you can include with the login: prompt. If you are providing PPP services, I recommend that you do not cahnge anything in your login prompt, such as the date/time or the port name. This makes creating chat scripts difficult, as the users trying to login will not know what to expect.

At this point, we are left with the last login messages. Unfortunately, these are not contained in files that are as easily removed as **/etc/motd** and **/etc/issue**. However, by *creating* a file, the file .hushlogin in your home directory, we can remove them. It has no contents; rather, the existence of this file is the key. You can create it simply by changing to a users home directory yours, if you are that user and running

touch .hushlogin

Often administrators want to keep users' knowledge of the system as limited as possible. This is particularly important for systems with a high level of security in which users start applications and never see the shell prompt. One give-away to what kind of system you are on is the following line when you login:

```
Last login: ...>
```

System administrators often call support asking for a way to turn this feature off. Fortunately, there is a way. This, too, is disabled by creating the .hushlogin file. Once this functionality is enabled, you can simplify things by having this file created every time a new user is created. This is done by simply adding the .hushlogin file to the **/etc/skel** directory. As with every other file in this directory, it will be copied to the user's home directory whenever a new user is created.

One thing to consider before you turn this feature off is that seeing when the last login was done may indicate a security problem. If you see that the last login was done at a time when you were not there, it may indicate that someone is trying to break into your account.

You can see who is currently logged in by running either the who or w command. These commands are kept in the file utmp in your system log directory **/usr/adm**, **/var/log**, etc. Once the system reboots, this information is gone.

You can also see the history of recent logins by using the last command. This information is kept in wtmp in the system log directory. This command is kept between reboots and, depending on how active your system gets, I have seen this file grow to more than a megabyte. Therefore, it might not be a bad idea to truncate this file at regular intervals. Note that some Linux distributions do this automatically.

One way to limit security risks is to keep the root account from logging in from somewhere other than the system console. This is done by setting the appropriate terminals in /etc/securetty. If root tries to log into a terminal that is not listed here, it will be denied access. It is a good idea to list only terminals that are on the system console tty1, tty2, etc..

If you really need root access, you can use telnet from a regular account and then su to root. This then provides a record of who used su.

6.3 Terminals

Unless your Linux machine is an Internet server or gateway machine, there probably will be users on it. Users need to access the system somehow. Either they access the systems across a network using a remote terminal program like telnet, rlogin, or access file systems using NFS. Also, like users typically do on Windows, they might log in directly to the system. With Linux, this (probably) is done from a terminal and the system must be told how to behave with the specific terminal that you are using.

Increasingly people are using graphical user interfaces (GUIs) to do much of their work. With many distributions a lot of the work is still done using the command line, which means they need a terminal, whether or not it is displayed within a graphical window.

In live environments that use Linux (such as where I work), you do not have the access to a graphical interface on all systems (for security reasons, among other things). Therefore, the only way to remotely administer the system is through telnet, which typically requires a terminal window. In cases like this, it is common to move from one operating system type to another (Linux to Sun, or vis-versa). Therefore, knowledge of terminal settings capabilities is often very useful.

When we talk about terminals, we are not just talking about the old fashioned CRT that is hooked up to your computer through a serial port. Instead, we are talking about any command-line (or shell) interface to the system. This includes serial terminals, telnet connections and even the command-line window that you can start from your GUI.

6.3.1 Terminal Capabilities

If you are interacting with the system solely through command line input, you have few occasions to encounter the terminal capabilities. As the name implies, terminal capabilities determine what the terminal is capable of. For example, can the terminal move the cursor to a specific spot on the screen?

The terminal capabilities are defined by one of two databases. Older applications generally use termcap, while newer ones use terminfo. For the specifics on each, please see the appropriate man-page. Here I am going to talk about the concept of terminal capabilities and what it means to you as a user.

Within each of these databases is a mapping of the character or character sequence the terminal xxpects for certain behavior. For example, on some terminals, pressing the backspace key sends a Ctrl-? character. On others, Crtl-H is sent. When your TERM environment variable is set to the correct one for your terminal, pressing the backspace key sends a signal to the system which, in turn, tells the application that the backspace characteristic was called. The application is told not just that you pressed the key with the left arrow (FACE="Symbol">¬) on it. Instead, the application is told that that key was the backspace. It is then up to the application to determine what is to be done.

The key benefit of a system like this is that you do not have to recompile or rewrite your application to work on different terminals. Instead, you link in the appropriate library to access either termcap or terminfo and wait for the capability that OS will send to you. When the application receives that capability (*not* the key), it reacts accordingly.

There are three types of capabilities. The first capabilities are Boolean, which determine whether that terminal has a particular feature. For example, does the terminal have an extra "status" line? The next type is numeric values. Examples of this capability are the number of columns and lines the terminal can display. In some cases, this may not remain constant, as terminals such as the Wyse 60 can change between 80- and 132-column mode. Last are the string capabilities that provide a character sequence to be used to perform a particular

operation. Examples of this would be clearing the line from the current cursor position to the end of the line and deleting the contents of an entire line (with or without removing the line completely).

Despite that there are hundreds of possible capabilities, any given terminal will have only a small subset of capabilities. In addition, many of the capabilities do not apply to terminals, but rather to printers.

Both the termcap and terminfo databases have their own advantages and disadvantages. The termcap database is defined by the file **/etc/termcap**, an ASCII file that is easily modified. In contrast to this is the terminfo database, which starts out as an ASCII file but must be compiled before it can be used.

The termcap entries can be converted to terminfo with the captoinfo command and then compiled using tic, the terminfo compiler. The tic utility will usually place the compiled version in a directory under **/usr/lib/terminfo** based on the name of the entry. For example, the ANSI terminal ends up in **/usr/lib/terminfo/a** and Wyse terminals end up in **/usr/lib/terminfo/w**.

6.3.2 Terminal Settings

Whenever you work with an application, what you see is governed by a couple of mechanisms. If you have a serial terminal, the flow of data is controlled by the serial line characteristics, including the baud rate, the number of data bits, parity, and so on. One aspect that is often forgotten or even unknown to many users is the terminal characteristics, which are used to control the physical appearance on the screen. However, most of the characteristics still apply, even if you are not connected through a serial terminal.

The reason is that these conventions date back to the time of tele-typewriters. You had a keyboard on one end of the connection connected to a printer that printed out every single character you typed. At that time, it was essential that both ends knew what characteristics the connection had. Even as technology advanced there was still a need to ensure both sides communicated in the exact same way. Since you could not guarantee that the default settings were the same on both ends, you needed a way to change the characteristics so that both ends matched.

As I mentioned elsewhere, the serial line characteristics are initially determined by the gettydefs file. The characteristics are often changed within the users' startup scripts (.profile, .login, etc.). In addition, you can change them yourself by using the **stty** command. Rather than jumping to changing them, lets take a look at what our current settings are, which we also do with the stty command. With no arguments, stty might give us something like this:

```
speed 38400 baud; line = 0;
-brkint ixoff -imaxbel
-iexten -echoctl
>
```

This is pretty straightforward. Settings that are Boolean values (on or off) are listed by themselves if they are on (ixoff) or have a minus sign in front if they are turned off (-brkint). Settings that can take on different values (like the baud rate) appear in two formats: one in which the value simply follows the setting name (speed 38400 baud) and one in which an equal sign is between them (line=0).

In general, if a setting has discrete values, like the baud rate, there is no equal sign. There is only a discrete number of baud rates you could have (i.e., there is no 2678 baud). If the stty setting is for something that could take on "any" value (like the interrupt key), then there is an equal sign. Normally, the interrupt key is something like Ctrl-C or the Delete key. However, it could be the f key or the Down-Arrow or whatever.

This example shows the more "significant" terminal (stty) settings. The top line shows the input and output speed of this terminal, which is 38400. On the second line, we see that sending a break sends an interrupt signal (-brkint).

Setting these values is very straightforward. For Boolean settings (on or off), the syntax is simply

stty <setting>

to turn it on or

stty -<setting> (note the minus sign in front)

to turn it off.

For example, if I wished to turn on input stripping (in which the character is stripped to 7 bits), the command would look like this:

stty istrip

Settings that require a value have the following syntax:

stty <setting> <value>

So, to set the speed (baud rate) to 19200, the syntax would look like this:

stty speed 19200

To set the interrupt character to Ctrl-C, we would enter

stty intr ^C

Note that ^C is not two separate characters. Instead, when you type it, hold down the Ctrl key and press "c." In most documentation you will see that the letter appears as capital although you actually press the lowercase letter. Sometimes you want to assign the particular characteristic to just a single key. For example, it is often the case that you want to use the backspace key to send an "erase" character. What the erase character does is tell the system to erase the last character, which is exactly what the backspace is supposed to do. Just like the case where you press the control key and the character, stty settings for single keys are done

the same way. For example, you would type "stty erase " and the press the backspace key (followed by the enter key, or course). What you would see might look like this:

```
stty erase ^?
```

The ^? is typically what the backspace key will send (at least that is the visual representation of what the backspace sends). You can get the same result by press CTRL-?.

If the default output does not show the particular characteristic you are looking for, you can use the -a option to show all the characteristics. You might end up with output like this:

```
speed 38400 baud; rows 25; columns 80; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = <undef>;
eol2 = <undef>; start = ^Q; stop = ^S; susp = ^Z; rprnt = ^R; werase = ^W;
lnext = ^V; flush = ^O; min = 1; time = 0;
-parenb -parodd cs8 hupcl -cstopb cread -clocal -crtscts
-ignbrk -brkint -ignpar -parmrk -inpck -istrip -inlcr -igncr icrnl ixon ixoff
-iuclc -ixany -imaxbel opost -olcuc -ocrnl onlcr -onocr -onlret -ofill
-ofdel nl0 cr0 tab0 bs0 vt0 ff0
isig icanon -iexten echo echoe echok -echonl -noflsh -xcase -tostop -echoprt
-echoctl echoke
>
```

Here were see a number of well-known characteristics, such as the baud rate, the numbers of rows and columns, interrupt character, end of file character and so on. Some of which we talked about in the section on working with the system. For details on what the rest of these mean, please see the **stty**(1L) man-page.

In principle, you can set any key to any one of the terminal characteristics. For example, I could set the interrupt key to be the letter g:

```
stty intr g
```

Although this not does make too much sense, it is possible. What does make more sense is to set the characteristic to something fitting for your keyboard. For example, you might be using telnet to move between system. The key sequence that your backspace sends may not be ^? (often it is ^H) and you want to set it accordingly (or the case is reversed, as we discussed above.)

To save, change, and then restore the original values of your **stty** settings, use the -g option. This option outputs the stty settings as a strings of hexadecimal values. For example, I might get something like this:

```
stty -g
```

```
500:5:d050d:3b:7f:1c:8:15:4:0:0:0:0:0:1a:11:13:0:0:0:0:0:0:0:0:0
>
```

We can run the stty command to get these values and make the changes, then run stty again and use these values as the argument. We don’t have to type in everything manually; we simply take advantage of the fact that variables are expanded by the shell before being passed to the command. You could use this to add an additional password to your system:

```
echo "Enter your password: \c"
oldstty=`stty -g`
stty -echo intr ^-
read password
stty $oldstty
>
```

Assign the output of the stty command to the variable old, then change the stty settings so that the characters you input are not echoed to the screen and the interrupt key is disabled (this is done with **stty -echo intr ^-**). Then read a line from the keyboard and reset the stty settings to their old value.

6.4 Printers and Interfaces

Under Linux, printing is managed and administered by several commands and files located in various parts of the system. The primary administrative directory is **/usr/spool/**. Each printer that you have configured has its own subdirectory, /usr/spool/lpd/<name>, where <name> is the name of the printer. In this subdirectory, you will find status information about the printer, as well as information about the jobs currently being printed.

The actual printing is done by the lpd daemon. On system start-up, lpd is started through one of the rc scripts (normally somewhere under **/etc/rc.d**). As it starts, lpd looks through the printer configuration file, **/etc/printcap**, and prints any files still queued (normally after a system crash).

In each spool directory is a lock file that contains the process id (PID) of the lpd process. The PID helps keep multiple printer daemons from running and potentially sending multiple jobs to the same printer at the same time. The second line in the lock file contains the control file for the current print job.

Management of the print system, or print spool, is accomplished through the **lpc** utility. This is much more than a "command" because it performs a wide range of functions. One function is enabling printing on a printer. By default, there is probably one printer defined on your system (often lp). The entry is a very simple print definition that basically sends all the characters in the file to the predefined port. (For the default printer on a parallel port, this is probably /dev/lp1.)

When a job is submitted to a local printer, two files are created in the appropriate directory in **/usr/spool**. (For the default printer, this would be **/usr/spool/lp1**). The first file, starting with cf, is the control file for this print job. Paired with the cf file is the data file, which starts with df and is the data to be printed. If you are printing a pre-existing file, the df file will be a copy of that file. If you pipe a command to the lpr command, the df file will contain the output of the command. Using the -s option, you can force the system to create a symbolic link to the file to be printed.

The cf file contains one piece of information on each of several lines. The first character on each line is an abbreviation that indicates the information contained. The information contained within the cf file includes the name of the host from which the print job was submitted (H), the user/person who submitted the job (P), the job name (J), the classification

of the print job (C), the literal string used on the banner page to identify the user (L), the file containing the data (this is the df file) (f), which file to remove or "unlink" when the job is completed (U), and the name of the file to include on the banner page (N). If you check the lpd man-page, you will find about a dozen more pieces of information that you could include in the cf file. However, this list represents the most common ones.

In the same directory, you will find a status file for that printer. This file is called simply "status" and normally contains a single line such as

```
printing disabled>
```

If you were to re-enable the printer, the line would then change to

```
lp is ready and printing>
```

Looking at this line, you might have noticed something that might seem a little confusing. (Well, at least it confused me the first time). That is, we've been talking about the directory lp1 all along, but this says the printer is lp. Does this mean that we are talking about two separate printers? No, it doesn't. The convention is to give the directory the same name as the printer, but there is no rule that says you have to. You can define both the printer name and the directory any way you want.

This is probably a good time to talk about the printer configuration file, **/etc/printcap**. This file contains not only the printer definitions but the printer "capabilities" as well. In general, you can say the printcap file is a shortened version of the termcap file (/etc/termcap), which defines the capabilities of terminals.

In the printcap file, you can define a wide range of capabilities or characteristics, such as the length and width of each line, the remote machine name (if you are remote printing), and, as we discussed, the name of the spool directory. I will get into shortly what each of the entries means.

As we talked previously, the lpc command is used to manage the print spooler. Not only can you use it to start and stop printing, but you can use it to check the status of all the printer queues and even change the order in which jobs are printed.

There are two ways of getting this information and to manage printer queues. The first is to call lpc by itself. You are then given the lpc> prompt, where you can type in the command you want, such as start, disable, or any other administrative command. Following the command name, you must either enter "all," so the command will be for all printers, or the name of the printer.

The lpc program will also accept these same commands as arguments. For example, to disable our printer, the command would be

```
lpc disable lp1
```

For a list of options, see the lpc man-page. A list of printer queue commands can be found in Table 0-1.

One aspect of the Linux print system that might be new to you is that you enable or disable the printing functionality within the kernel. Even though printer functionality is configured, you may not be able to print if you have hardware conflicts. When your run 'make configure' one of the options is to enable printing.

Once you have added the printer support to the kernel, the first thing you should do is test the connectivity by using the ls command and sending the output to the printer device. This will probably be /dev/lp0, /dev/lp1, or /dev/lp2, which corresponds to the DOS device LPT1, LPT2, and LPT3, respectively. For example, to test the first parallel port you could use

```
ls > /dev/lp0
```

What results is:

```
INSTALL@   dead.letter     linux@  lodlin15.txt    lodlin15.zip
        mbox    sendmail.cf        tests/
>
```

However, if you were to issue the command without the redirection, it would probably look like this:

```
INSTALL@
dead.letter
linux@
lodlin15.txt
lodlin15.zip
mbox
sendmail.cf
tests/
>
```

The reason for this is that the ls command puts a single new-line character at the end of the line. Normally, the shell sees that new-line character and is told to add a carriage return onto the line. However, the printer has been told. Therefore, when it reaches the end of the line with the sendmail.cf, just a new line is sent. Therefore, the printer drops down to the next (new) line and starts printing again. This behavior is called "stair-stepping" because the output looks like stair steps. When a carriage return is added, the shell returns back to the left of the screen as it adds the new line.

Table - Print Queue Commands

Command	Function
/usr/sbin/lpc	Printer control program
/usr/sbin/lpd	Print spooler daemon
/usr/bin/lpr	Print program
/usr/bin/lpq	Print queue administration program
/usr/bin/lprm	Remove jobs from print queue
/usr/sbin/pr	Convert text files for printing

6.4.1 advanced formatting

Being able to output to paper is an important issue for any business. Just having something on paper is not all of the issue. Compare a letter that you type on a typewriter to what you print with a word processor. With a word processor, you can get different sizes or types of fonts and sometimes you can even create drawings directly in the word processor.

Many of you who have dealt with UNIX before might have the misconception that UNIX is only capable of printing simple text files. Some of you might have seen UNIX systems with a word processor that did fancy things with the output. Fortunately for us, these fancy tricks are not limited to the word processing packages. Using vi and a couple of commonly available tools, you can output in a wide range of styles.

Readily available from a number of sites, the TeX or LaTeX pronounced Tech and Lahtech text formatting package can be used to create professional-looking output. Many academic and research institutions running UNIX use LaTeX as their primary text processing system. Not only is it free but the source code is also available, allowing you to extend it to suit your needs. In many cases, the only way to get it onto your system is to get the source code and compile it.

Like the *roff family, TeX is input directly by the writer. These source files are then run through a processor that formats the output based on codes that were input. This process generates a device independent file, usually with the extension .dvi. The .dvi files are analogous to .o files in C because they need to be manipulated further to be useful. Unfortunately, this does not work for every kind of printer.

If your printer does not understand the .dvi file, the dvips program will convert the .dvi file to PostScript. If your printer doesn’t support PostScript, you can use ghostview to output to a format your printer can understand.

Included on your system provided you installed the TeX package is the dvips program, which converts the .dvi files to PostScript. These PostScript files can be printed out on any compatible printer.

At first this may sound a little confusing and annoying. You have to use so many tools that just to get a simple printout. First, if all you really need is a simple printout, you probably won't need to go through all of these steps. This demonstrates that no matter what standard you choose to use, there are Linux tools available to help you get your job done.

Many different programs are available to allow you to print out, view, and manipulate PostScript files. Ghostscript is a program used to view PostScript files. These need not be files that you generated on your local machine, but any PostScript files you have. Ghostscript can also be used to print PostScript files to print the file to non-PostScript-compatible printers.

Ghostscript supports the resolutions that most printers can handle. However, if you are printing to a dot-matrix printer, you need to be especially careful about getting the right resolution because it is not normally the standard 300 DPI.

I have to pause here to remind you about working with PostScript files and printers. Sometimes the printer is PostScript-compatible, but you have to tell it to process the file as PostScript and not as raw text. This applies to older models of certain laser jet printers. Once, I wanted to print out a 50-page document and forgot to set the flag to say that it was a PostScript file. The result was that instead of 50 pages, I ended up with more than 500 pages of PostScript source.

Under Linux, printers are not the only way you can get words on paper. As of this writing, there are at least three packages with which you can fax documents from your Linux system. First, however, you must have a fax modem with which you can connect.

Here I need to side-step for a minute. The older type of fax, Class 1 faxes, did not have as much processing power distributed in the hardware. Instead, the software took over this job. It works fine on single-user systems like Windows, but under pre-emptive multitasking systems like Linux, you can run into timing problems. Pre-emptive multitasking is where the operating system decides which process will run and therefore could pause the fax program at a crucial moment. More details can be found in the section on processes.

In addition to Class 1, faxes fall into different groups. To work correctly, the fax software needs to convert the document you are sending into a group-III-compatible image. This can be done with Ghostscript.

The GNU netfax program accepts several different file formats as of this writing, PostScript, dvi, and ASCII. Originally available from *prep.ai.mit.edu*, it is no longer supported by the GNU. More extensive than netfax is HylaFlex renamed from FlexFax available to avoid trademark conflicts. This is available as of this writing with ftp from *sgi.com* under /sgi/fax/. With this package, not only can you send faxes, but you can configure it to receive them as well.

Man-pages are something that you may need to print. If you have files in ASCII format the cat pages, this is not an issue. However, with pages that have been formatted with *roff formatting, you have a couple of choices. The man program has the ability to process files with *roff formatting. By redirecting the output on man to a file often piping it through col, you can get clean ASCII text that you can then print.

6.4.2 printcap

As with the termcap file, each entry in the printcap file is separated by a colon. Boolean characteristics, such as suppressing the header (sh), exist by themselves. Characteristics that can take on a value, such as the name of the output device, are followed by an equal sign (=) and the value (lp=/dev/lp1). For a complete list of characteristics, see the printcap man-page.

Each entry in the **/etc/printcap** file consists of single logical line. There is one entry for each printer on your system. To make the entry easier to read, you can break each logical line into several physical lines. As an example, lets look at the entry for the default, generic printer:

```
lp:lp=/dev/lp1:sd=/usr/spool/lp1:sh
```

The first part of the line is the name of the printer, in this case, lp. Each field is separated from the others with a colon, so in this example, there are three fields (plus the printer name).

If we were to break this example into multiple physical lines, it might look like this:

```
lp:\
   :lp=/dev/lp1:\
   :sd=/usr/spool/lp1:\
   :sh
```

At the end of each physical line, there is a back-slash to tell lpd that the logical line continues. You'll also see that each field now has a colon before it and after it.

Although it is not necessary, you may find a file minfree in each of the spool directories. This is a simple text file that contains the number of disk blocks that should be left to keep the print spooler from filling up the disk. As a safety mechanism on a system with a lot of print jobs, the spool directory can be put on a separate file system. Should it fill up, the rest of the system won't suffer.

Often, data is sent directly to the printer devices, either because it is supposed to be raw ASCII text or because the program that created the data did its own formatting. This is referred to as raw data as the system doesn't do anything with it.

Sometimes the data is sent by the lpd daemon through another program that processes the data in preparation of sending it to the printer. Such programs are called filters. The stdin of the input filters receive what the lpd puts out. The stdout of the filter then goes to printer. Such filters are often called input filters and are specified in the printcap file with if=.

Because of this behavior, a print filter can be anything that understands the concept of stdin and stdout. In most cases on Linux, the input filters that I have seen are simply shell scripts. However, they can also be perl scripts.

With the exception of an input filter or a log file (which is specified using lf=), I have rarely used any other option for local printing. However, using the printcap file, you can configure your printer to print on a remote system, which is the subject of the next section.

6.4.3 remote printing

Setting up your system to print from another machine requires just a couple of alterations in your printcap file. Use the rm= field to specify the remote machine and the rp= field to specify the remote printer on that machine. Sending the print job to the printer is the last thing that happens, so any other options, including input filters, are also honored.

On the destination side, you must be allowed to access the other machine. If you are already a trusted host and have an entry in **`/etc/hosts.equiv`**, then there is no problem. If not, you will be denied access. (This is a good time to start thinking about a log file.)

If the sole reason the remote machine needs to trust your machine is to do remote printing, I would recommend *not* including it in the hosts.equiv file. This opens up more holes. Instead, put your host name in the file **`/etc/hosts.lpd`**. The only thing this file does is decide who can access the printers remotely. Putting remote machine names here is much safer.

6.5 System Logging

I am regularly confronted by Windows NT users who are overwhelmed by how much information you can collect and process using the Windows NT Event Viewer. It is so nice, they maintain, that occurrences (events) are sorted by system, security and applications. They go on with how much you can filter the entries and search for specific values.

The problem is, that's where it stops. With the exception of a few security related events, what you are able to log (or not log) is not configurable under Windows NT. You get whatever Microsoft has decided is necessary. No more and no less. You can filter what is displayed, but there is little you can do to restrict what is logged.

With Linux the situation is completely different. Not only can you tell the system what the system should log but exactly where it should log it. On the other hand, Windows NT always logs specific events to a specific file. In addition, Windows NT differentiates between only three different types of logs. This means you may need to wade through hundreds if not thousands of entries looking for the right one. Not only can you say what is logged and what not, you can specifically define where to log any given type of message, including sending all (or whatever part you define) to another machine, and even go so far as to execute commands based on the messages being logged.

6.5.1 Syslogd

The workhorse of the Linux logging system is the system logging daemon or syslogd. This daemon is normally started from the system start-up (rc) scripts when the system goes into run level 1. Once running, almost any part of the system, including applications, drivers, as well as other daemons can make log entries. There is even a command line interface so you can make entries from scripts or anywhere else.

With Windows NT, each system maintains its own log files. There is no central location where they are all stored. Although the Event Viewer can access event logs on other machines, this can often take a great deal of time especially when there are a lot of entries and

you have a slow connection.

Instead, syslogd can be configured to send all (or just some) of the messages to a remote machine, which processes them and writes them to the necessary files. It is thus possible that all the log messages of a particular type from all machines in your network are stored in a single file, which make accessing and administering them much easier.

Another advantage is due to the fact that syslogd stores configuration information and log entries in text files. Therefore, it is a simple matter of writing a script that parses the entries and splits them into separate files, or processes them in other ways.

Part of this ability lies in the standard format of each log entry. Although it is possible that a rogue program could write information in any order, all system daemons and most programs follow the standard, which is:

```
date time system facility message
```

Here "system" is the host name which generated the message. The "facility" is a component of the system generating the message. This could be anything like the kernel itself, system daemons and even applications. Finally, there is the text of the message itself. Here are two messages on the system jupiter. One is from syslogd and the other from the kernel:

```
Jun  5 09:20:52 jupiter syslogd 1.3-0: restart.
>

Jun  5 09:20:55 jupiter kernel: VFS: Mounted root (ext2 file system) readonly.
>
```

As you can see, even if you could not separate the log entries into different files, it would be fairly easy to separate them using a script.

Configuring syslogd

What is done and when it is done is determined by the syslogd configuration file, syslog.conf, which is usually in /etc. (I have never seen it anywhere else.) This is a typical Linux configuration file with one item (or rule) per line and comment lines begin with a pound-sign (#). Each rule consists of selector portion, which determines the events to react to and the action portion, which determines what is to be done.

The selector portion is itself broken into two parts, which are separated by a dot. The facility part says what aspect of the system is to be recorded and the priority says what level of messages to react to. The selector has the general syntax:

```
facility.priority
```

You can see a list of facilities in table 1 and a list of the priorities in table 2.

For both facilities and priorities there is a "wildcard" that can be used (an asterisk - *) which means any facility or any priorities. For example, *.emerg would mean all emergency messages. mail.* would mean all messages coming from the mail facility. Logically, *.* means all priorities of messages from all facilities.

The word "none" is used to refer to no priority for the specified facility. For example, the selector mail.none would say not to perform the action for any mail event. At first, this might not make sense. Why not simply leave off that facility? The answer lies in the previous paragraph. Using the wildcard, you could say that all info messages were to be logged to a certain file. However, for obvious reasons, you want all of the security (regardless of the priority) written to another file.

Another possibility is to specify a sub-set of facilities, rather than all of them. This is done by separating the facilities with a comma and then the priority follows the last facility listed. For example, to refer to information messages for mail, uucp and news, the selector entry would look like this:

```
mail,uucp,news.info
```

One thing I need to point out here is that when you specify a priority, you are actually specifying everything at that priority or *higher*. Therefore, in this example, we are selecting all of the priorities at info and higher.

There are three primary things you can do with these events (the actions). Probably the most common action is to write them to a file. However, there is more to this than it appears. Remember that Linux (as well as other UNIX dialects) treat devices as files. Therefore, you can send the logging messages to a specific device.

Here, I not talking about sending them a tape drive (although that might not be a bad idea). Instead, I am talking about something like the system console (/dev/console). It is a common practice to send emergency messages to the system console, where someone will see the messages no matter to what console terminal they are logged on. In other cases, kernel messages are sent to one of the console terminals (e.g. /dev/tty7). You might end with something like this:

```
kernel.*                /dev/tty7
```

When writing to files, you want to consider that the system will actually write the information to the disk with each event. This ensures the entry actually makes it to the file if the system should crash. The problem is that writing to the harddisk takes time. That’s why the system normally saves up a number of writes before sending them all to the disk.

If overall system performance becomes an important factor in regard to logging, you can tell syslogd **not** to sync the disk each time it writes to a log file. This is done by putting a minus sign (-) in front of the file name, like this:

```
lpr.info        -/var/adm/printer.log
```

If you disable syncing the log file like this, one important thing to remember is that you stand the chance of losing information. If the system goes down for some reason before the information is written to the file, you may lose an important clue as to why the system went down. One solution would be to have a central log server where all of the information is sent and where you do **not** disable syncing. That way no matter what, you have a record of what happened.

Sending the log messages to another machine is done by using an at-sign (@) in front of the machine name as the action. For example:

```
*.emerg                    @logserver
```

This sends all emergency message to the machine logserver. I would suggest that you do not create a log server that is connected to the Internet. A ill-intended person might be able to bring the system to a halt or at least affect its performance by flooding it with erroneous log messages.

Another useful feature is the ability to send messages to named pipes. This is done by preceding the name of the pipe by the pipe-symbol (|). I find this a useful way of sending log messages to other programs, where I can process them further. Named pipes are created using the mkfifo(1) command and must exist prior to syslogd starting.

Another action is the ability to send messages to particular users, provided they are logged in at the moment. To do this you simply put their username as the action. To send it to multiple users, separate the names by a comma. This might give you something like this:

```
*.emerg                    root,jimmo
```

By using an asterisk in place of the list of user names, you can send a message to everyone logged in.

In some cases, you want multiple actions for a specific facility or priority. This is no problem. You simply create multiple rules. One common example is broadcasting all of the emergency messages to every user, as well as writing them to a log file **and** sending them to another server in case the local machine crashes. This might be done like this:

```
*.emerg              /var/adm/messages*.emerg                    **.emerg                    @logserver
```

Previously, I mentioned the ability to cause a single action based on the same kind of messages for multiple facilities. This is still an example of a *single* selector resulting in a specific action. Taking this one step further, you might want multiple selectors all to result in a specific action. Although it could be done with multiple rules, it possible to have multiple selectors all on the same line. This is done by separating the selectors with a semi-colon (;).

```
*.emerg;kernel.critical                    root,jimmo
```

This would notify the users root and jimmo for all emergency messages as well as critical messages from the kernel facility.

The Linux syslogd has added a couple of functions that are not available in other versions of UNIX. By preceding a priority with an equal-sign (=), you tell syslogd only to react to that one priority. This is useful since syslogd normally reacts to everything with that priority and higher. One place where this is useful is when you want all debugging messages to be logged to a specific file, but everything logged to another file.

You can also explicitly exclude priorities by preceding them with an exclamation mark. Note that this will exclude the priorities listed as well as anything higher. You can combine the equal-sign and exclamation mark equal-sign and exclamation mark and therefore exclude a

specific priority. If you do so, you need to precede the equal sign with the exclamation mark as what you are saying is not to include anything that equal a particular priority.

All of these features can be combined in many different ways. For example, you can have multiple selectors, which include as well as exclude specific priorities. For example:

```
*.warn;mail.=info;lpr.none;uucp.!crit           /dev/tty07
```

This would send warning messages from all priorities to the system console terminal /dev/tty7, plus the mail log messages at only the info priority, no printer messages at all, and finally excluding just the uucp critical messages. Granted this is a rather contrived example, but it does show you how complex you can get.

Note that multiple selectors on a single line can cause some confusion when there are conflicting components within a selector. The thing to keep in mind is that the last component takes precedence. In the previous example, we specified warning messages for all facilities and then "overwrote" portions of that for the mail, lpr and uucp facilities.

6.5.2 Managing System Logs

Often times it is useful to log messages from scripts. This can done using the logger command (usually found in **/usr/bin**). Without any options it takes the user name as the facility and "notice" as the priority. However, you can specify both a facility and priority from the command line by using -p option for example:

```
logger –p kern.warning The kernel has been recompiled.
```

This would send the specified message to the same place other kernel messages are sent. For details on the other options, see the logger(1) man-page.

One common problem is what to do with all of the log messages. If you do a lot of logging (particularly if everything is sent to a central server), you can fill up your filesystem faster than you think. The most obvious and direct solution is to remove them as after a specific length of time or when they reach a particular size.

It is a fairly simple matter to write a shell script that is started from cron, which looks at the log files and takes specific actions. The nice thing is that you do not have to. Linux provides this functionality for you in the form of the logrotate command.

As its name implies, the goal of the logrotate program is to "rotate" log files. This could be as simple as moving a log file to a different name and replacing the original with an empty file. However, there is much more to it.

Two files define how logrotate behaves. The state file (specified with the -s or --state option) basically tells logrotate when the last actions were taken. The default is **/var/state/logrotate**.

The configuration file tells logrotate when to rotate each of the respective files. If necessary, you can have multiple configuration files which can all be specified on the same command line or you include configuration files within another one.

The logrotate configuration file is broken into two parts. At the beginning are the global configuration options, which apply to all log files. Next, there are the configuration sections of each of the individual files (the logfile definitions). Note that some options can be global or for a specific log file, which then overwrites the global options. However, there are some that can only be used within a logfile definition.

A very simple logrotate configuration file to rotate the **/var/log/messages** might look like this:

```
errors root@logserver
compress

/var/log/messages {
   rotate 4
   weekly
   postrotate
        /sbin/killall -HUP syslogd
   endscript
}
```

At the top are two global options, followed by a logfile definition for **/var/log/messages**. In this case, we could have included the global definitions within the log file definition. However, there is normally more than one logfile definition.

The first line says that all error messages are sent (mailed) to root at the logserver. The second line says that log files are to be compressed after they are rotated.

The logfile definition consists of the logfile name and the directives to apply, which are enclosed within curly brackets. The first line in the logfile definition says to rotate the 4 times before being removed. The next line says to rotate the files once a week. Together these two lines mean that any given copy of the **/var/log/messages** file will be saved for 4 weeks before it is removed.

The next three lines are actually a set. The postrotate directive says that what follows should be done immediately after the log file has been rotated. In this case, syslogd is sent a HUP signal to restart itself. There is also a prerotate directive, which has the same basic functionality, but does everything **before** the log is rotated.

It is also possible to specify an entire directory. For example, you could rotate all of the samba logs by specifying the directory /var/log/samba.d/*.

As I mentioned, you can also rotate logs based on their size. This is done by using the size= option. Sitting size=100K would rotate logs larger than 100 Kb and 100M would rotate logs larger than 100 Mb.

Although you can ease the management of your log files with just the options we discussed, there are an incredible number of additional options which you can use. Table 3 contains a list of options you can use with a brief explanation. For more details see the logrotate(1) man-page.

Table 1
authpriv
cron
daemon
kern
lpr
mail
mark
news
security
syslog
user
uucp
local0 through local7.

The facility "security" should no longer be used and the "mark" facility is used internally and should not be used within applications. The facilities local0 through local8 are intended for local events on you local system when there is no other applicable facility.

Table 2 - Syslogd Priorities in increasing significance
debug
info
notice
warning or warn
err or error
crit
alert
emerg or panic

The priorities error, warn and panic are deprecated and should no longer used.

Table - logrotate options

compress/nocompress - compresses or does not compress old versions of logs.

delaycompress - Wait until the next cycle to compress the previous log.

create mode owner group - Log file is recreated with this mode, owner and group. (nocreate overrides this.)

daily, weekly, monthly - Rotate logs in the indicated interval.

errors address - Send errors to the address indicated.

ifempty - Rotate the logs even if they are empty. (notifempty overrides this.)

include file_or_directory - Include the indicate file at this point. If a directory is given, all real files in that directory are read.

mail address - Logs rotate out of existence are mailed to this address. (nomail overrides this)

olddir directory - old logs are moved to this directory, which must be on the same physical device. (noolddir overrides this.)

postrotate/endscript - delimits commands run after the log is rotated. Both must appear on a line by themselves.

prerotate/endscript - delimits commands before after the log is rotated. Both must appear on a line by themselves.

rotate count - Rotates the log times before being removed.

size size - Log files greater than are removed.

tabooext [+] list - list of files not to include. A plus-sign means the files are added to the list rather than replacing it.

6.6 Backups

In the section on backing up and restoring files under Working with the System, we talked briefly about the process of backing up files and how to restore them. However, simply knowing what tools you need is usually not enough. You might not have enough time or space to do a complete backup, or restoring from a complete backup is not efficient. An advantage of doing a complete backup every day is that it is very simple. If everything fits on a single tape, you stick in a tape when you are done for the day and have something like cron schedule a backup in the middle of the night. If you have more than will fit on one tape, there are hardware solutions, such as multiple tape drives or a tape loader.

Rather that doing a complete back up every day, there are a number of different strategies that you can employ to keep your data safe. For example, one way is to back up all of the data at regular intervals and then once a day backup only the files that have changed since this full backup. If you need to restore, you can load your master back and one extra tape.

Alternatively, you could make a full backup and then each day, only backup the files that changed on that day. This is can be a problem if you have made changes to files on several different days and need to load each time. This can be very time consuming.

What this basically says it that you need to make some kind of decision about what kind of backup strategy you will use. Also consider that the backup strategy should have backups being done at a time which has the least influence on users, for example in the middle of the night or on weekends.

Details of all this, I actually save for the section on problem solving.

6.7 cron

cron is a commonly confusing and misconfigured aspect of the operating system. Technically, cron is just the clock daemon (**/usr/sbin/cron** or perhaps **/usr/sbin/crond**) that executes commands at specific times. However, a handful of configuration files and programs go into making up the cron package. Like many system processes, cron never ends.

The controlling files for cron are the cron-tables or crontabs. The crontabs are often located in **/var/spool/cron/crontab**. However, on SuSE you will find them in **/var/spool/cron/tabs**. The names of the files in this directory are the names of the users that submit the cron jobs.

Unlike other UNIX dialects, the Linux cron daemon does not sleep until the next cron job is ready. Instead, when cron completes one job, it will keep checking once a minute for more jobs to run. Also, you should not edit the files directly. You can edit them with a text editor like vi, though there is the potential for messing things up. Therefore, you should use the tool that Linux provides: crontab. (see the man-page for more details)

The crontab utility has several functions. It is the means by which files containing the cron jobs are submitted to the system. Second, it can list the contents of your crontab. If you are root, it can also submit and list jobs for *any* user. The problem is that jobs cannot be submitted individually. Using crontab, you must submit all of the jobs at the same time.

At first, that might sound a little annoying. However, let's take a look at the process of "adding" a job. To add a cron job, you must first list out the contents of the existing crontab with the -l option. If you are root and wish to add something to another user's crontab, use the -u option followed by the user's logname. Then redirect this crontab to a file, which you can then edit. (Note that on some systems crontab has -e (for "edit"), which will do all the work for you. See the man-page for more details.)

For example, lets say that you are the root user and want to add something to the UUCP user's crontab. First, get the output of the existing crontab entry with this command:

```
crontab -l -u uucp >/tmp/crontab.uucp
```

To add an entry, simply include a new line. Save the file, get out of your editor, and run the crontab utility again. This time, omit the -l to list the file but include the name of the file. The crontab utility can also accept input from stdin, so you could leave off the file name and crontab would allow you to input the cronjobs on the command line. Keep in mind that any previous crontab is removed no matter what method you use.

The file **/tmp/crontab.uucp** now contains the contents of UUCPs crontab. It might look something like this:

```
39,9 * * * * /usr/lib/uucp/uudemon.hour > /dev/null
10 * * * * /usr/lib/uucp/uudemon.poll > /dev/null
45 23 * * * ulimit 5000; /usr/lib/uucp/uudemon.clean > /dev/null
48 10,14 * * 1-5 /usr/lib/uucp/uudemon.admin > /dev/null
```

Despite its appearance, each crontab entry consists of only six fields. The first five represent the time the job should be executed and the sixth is the actual command. The first five fields are separated by either a space or a tab and represent the following units, respectively:

- minutes (0-59)
- hour (0-23)
- day of the month (1-31)
- month of the year (1-12)
- day of the week (0-6, 0=Sunday)

To specify all possible values, use an asterisk (*). You can specify a single value simply by including that one value. For example, the second line in the previous example has a value of 10 in the first field, meaning 10 minutes after the hour. Because all of the other four time fields are asterisks, this means that the command is run every hour of every day at 10 minutes past the hour.

Ranges of values are composed of the first value, a dash, and the ending value. For example, the fourth line has a range (1-5) in the day of the week column, meaning that the command is only executed on days 1-5, Monday through Friday.

To specify different values that are not within a range, separate the individual values by a comma. In the fourth example, the hour field has the two values 10 and 14. This means that the command is run at 10 a.m. and 2 p.m.

Note that times are additive. Lets look at an example:

```
10 * 1,16 * 1-5 /usr/local/bin/command
```

The command is run 10 minutes after every hour on the first and sixteenth, as well as Monday through Friday. If either the first *or* the sixteenth were on a weekend, the command would still run because the day of the month field would apply. However, this does not mean that if the first is a Monday, the command is run twice.

The crontab entry can be defined to run at different intervals than just *every* hour or *every*day. The granularity can be specified to every two minutes or every three hours without having to put each individual entry in the crontab.

Lets say we wanted to run the previous command not at 10 minutes after the hour, but every ten minutes. We could make an entry that looked like this.:

```
0,10,20,30,40,50 * 1,16 * 1-5 /usr/local/bin/command
```

This runs every 10 minutes: at the top of the hour, 10 minutes after, 20 minutes after, and so on. To make life easier, we could simply create the entry like this:

```
*/10 * 1,16 * 1-5 /usr/local/bin/command
```

This syntax may be new to some administrators. (It was to me.) The slash (/) says that within the specific interval (in this case, every minute), run the command every so many minutes; in this case, every 10 minutes.

We can also use this even when we specify a range. For example, if the job was only supposed to run between 20 minutes after the hour and 40 minutes after the hour, the entry might look like this:

```
20-40 * 1,16 * 1-5 /usr/local/bin/command
```

What if you wanted it to run at these times, but only every three minutes? The line might look like this:

```
20-40/3 * 1,16 * 1-5 /usr/local/bin/command
```

To make things even more complicated, you could say that you wanted the command to run every two minutes between the hour and 20 minutes after, every three minutes between 20 and 40 minutes after, then every 5 minutes between 40 minutes after and the hour.

```
0-20/2,21-40/3,41-59/5 * 1,16 * 1-5 /usr/local/bin/command
```

One really nice thing that a lot of Linux dialects do is allow you to specify abbreviations for the days of the week and the months. Its a lot easier to remember that fri is for Friday instead of 5.

With the exception of certain errors in the time fields, errors are not reported until cron runs the command. All error messages *and* output is mailed to the users. At least that's what the crontab man-page says and that is basically true. However, as you see in the previous examples, you are redirecting stdout to /dev/null. If you wanted to, you could also redirect stderr there and you would never see whether there were any errors.

Output is mailed to the user because there is no real terminal on which the cronjobs are being executed. Therefore, there is no screen to display the errors. Also, there is no keyboard to accept input. Does that mean you cannot give input to a cron job? No. Think back to the discussion on shell scripts. We can redefine stdin, stdout and stderr. This way they can all point to files and behave as we expect.

One thing I would like to point out is that I do not advocate doing redirection in the command field of the crontab. I like doing as little there as possible. Instead, I put the absolute path to a shell script. I can then test the crontab entry with something simple. Once that works, I can make changes to the shell script without having to resubmit the cronjob.

Keep in mind that cron is not exact. It synchronizes itself to the top of each minute. On a busy system in which you lose clock ticks, jobs may not be executed until a couple minutes after the scheduled time. In addition, there may be other processes with higher priorities that delay cron jobs. In some cases, (particularly on very busy systems) jobs might end up being skipped if they are run every minute.

Access is permitted to the cron facility through two files, both in **`/etc`**. If you have a file cron.allow, you can specify which users are allowed to use cron. The cron.deny says who are specifically not allowed to use cron. If neither file exists, only the system users have access. However, if you want everyone to have access, create an entry cron.deny file. In other words, no one is denied access.

It is often useful for root to run jobs as a different user without having to switch users (for example, using the su command). Most Linux dialects provide a mechanism in the form of the **/etc/crontab** file. This file is typically only writable by root and in some cases, only root can read it (which is often necessary in high security environments). The general syntax is the same as the standard crontabs, with a couple of exceptions.

The first difference is the header, which you can see here:

```
SHELL=/bin/sh
PATH=/usr/bin:/usr/sbin:/sbin:/bin:/usr/lib/news/bin
MAILTO=root
#
# check scripts in cron.hourly, cron.daily, cron.weekly, and cron.monthly
#

59 *  * * *     root  rm -f /var/spool/cron/lastrun/cron.hourly
14 0  * * *     root  rm -f /var/spool/cron/lastrun/cron.daily
29 0  * * 6     root  rm -f /var/spool/cron/lastrun/cron.weekly
44 0  1 * *     root  rm -f /var/spool/cron/lastrun/cron.monthly
```

The SHELL variable defines the shell under which each command will run. The PATH variable is like the normal PATH environment variable and defines the search path. The MAILTO variable says who should get email messages, which includes error messages and the standard output of the executed commands.

The structure of the actual entries is pretty much the same with the exception of the user name (root in each case here). This way, the root users (or whoever can edit /etc/crontab) can define which user executes the command. Keep in mind that this can be a big security hole. If someone can write to this file, they can create an entry that runs as root and therefore has complete control of the system.

The next command in the cron "suite" is at. Its function is to execute a command at a specific time. The difference is that once the at job has run, it disappears from the system. As for cron, two files, at.allow and at.deny, have the same effect on the at program.

The batch command is also used to run commands once. However, commands submitted with batch are run when the system gets around to it, which means when the system is less busy, for example, in the middle of the night. Its possible that such jobs are spread out over the entire day, depending on the load of the system.

One thing to note is the behavior of at and batch. Both accept the names of the commands from the command line and not as arguments to the command itself. You must first run the command to be brought to a new line, where you input the commands you want execute. After each command, press Enter. When you are done, press Ctrl-D.

Because these two commands accept commands from stdin, you can input the command without having to do so on a new line each time. One possibility is to redirect input from a file. For example

```
at now +1 hour < command_list
```

where command_list is a file containing a list of commands. You could also have at (or batch) as the end of a pipe

```
cat command_list | at now + 1 hour
```

```
cat command_list | batch
```

Another interesting thing about both at and batch is that they create a kind of shell script to execute your command. When you run at or batch, a file is created in **/usr/spool/cron/atjobs**. This file contains the system variables that you would normally have defined, plus some other information that is contained in /usr/lib/cron.proto. This essentially creates an environment as though you had logged in.

6.8 User Communication

If you are running a multiuser system like Linux, you should expect to find other users on your system. (I guess that's why it is a multi-user system.) Although there are many built-in mechanisms to keep users separated, sometimes you will want to communicate with other users.

Linux provides several tools to do this, depending on exactly what you want to accomplish. If you simply want to send a quick message to someone, for example, to remind him or her of a meeting, you might use the write program, which sends (writes) a message to his or her terminal.

In contrast to some other systems (say, the winpop mechanism under Windows), each line is sent when you press Enter. If you are on the receiving end of the message, the system lets you know who sent you the message.

If the person you are trying to contact is logged in more than once, you need to specify the terminal to which you want to send the message. So, if I wanted to talk to the user jimmo on terminal tty6, the command would look like this:

```
write jimmo tty6
```

If you omit the terminal, write is kind enough to let you select which terminal to which you want to send the message.

It might happen that someone tries the above command and receives the following message:

```
write: jimmo has messages disabled.>
```

This message means that jimmo has used the mesg command to turn off such messages. The syntax for this command is

```
mesg n
```

to turn it off and

mesg y

to turn it on. Unless the system administrator has decided otherwise, the command is on by default. I have worked on some systems in which the administrator changed the default to off.

An extension of write is the wall command. Instead of simply writing the message to a single user, wall writes as if it were writing on a (where else) wall. That is, everyone can see the message when it is written on a wall, and so can every user. The wall command is often used by root to send messages about system status (e.g. when the system is about to be shutdown. Even if a user has disabled messages, the root user can still send them messages using wall.

If you want to have an interactive session, you could send write messages back and forth. On the other hand, you could use the talk program that was designed to do just that. When talk first connects to the other user, that other user sees on his or her screen

```
Message from TalkDaemon@source_machine...
talk: connection requested by callers_name@his_machine
talk: respond with: talk callers_name@his_machine
>
```

As the message indicates, to respond, you would enter

talk callers_name@his_machine

You might have noticed that you can use talk to communicate with users on other machines. If you omitted the machine name, talk would try to contact the user on the local machine (localhost). The preceding message would simply say

```
talk: connection requested by callers_name@localhost
>
```

You can also disable talk by using the mesg command.

It is common practice to use a couple of terms from radio communication when using talk. Because you cannot always tell when someone is finished writing, it is common to end the line with -o (or use a separate line) to indicate that your turn is "over." When you are finished with the conversation and wish to end it, use oo (over and out).

Both of these mechanisms have some major problems if the user is not logged in: they don’t work! Instead, there’s mail or, more accurately, electronic mail (or e-mail).

On most UNIX systems (including Linux), e-mail is accessed through the mail command. Depending on your system, the mail program may be linked to something else. On my system, the default was to link to /usr/bin/mail.

There are several different programs for sending and viewing mail. You could use one mail program (or mailer) to send the message and another to read it. Often the program that you use to read your mail is called a mail reader or, simply, reader. Before we go on to the more advanced mail programs, I want to talk about the most common mail program and the one that

is most likely to be on your system. (From here on, I will be referring to e-mail simply as mail.)

Mail comes in units called *messages*. Whether you use UUCP or the Internet, mail is sent back and forth in messages. However, once the message has reached its destination, it is usually tacked onto the end of an existing mail file. There is usually one mail file per user, but that single file contains all of a user's messages (that is, all those that haven't yet been deleted).

To read your mail, you can use three primary character-based programs: elm, pine, and the default reader, mail. Actually, you can use all three programs to send mail as well as read it. Each program has its own advantages and disadvantages. Although the mail interface looks menu-driven, it simply scrolls the information across the screen. Both elm and pine have much more complex menuing systems. Because of this, mail is easier to learn, but you can do much more with the other two programs.

All three programs understand the concept of a "folder" in which you can store messages. This allows you to develop a hierarchy of files that is no different from the normal file system. How the folders are created and managed depends on the program you are using. Therefore, I would suggest that once you decide to use a specific program, stick with it because the files may not be compatible.

In keeping with the basic premise of this book, I must treat these programs as applications. Therefore, I won't go into any more detail about them. Instead, I suggest that you install all three and see which one suits your needs best. If you have the space, you may consider providing all three for your users. The man-pages provide a great deal of information and each program has its own on-line help.

If you are using the X-Windowing System and a desktop environment such as the KDE, you have a much larger and varied choice, such as my favorite Kmail. Prior to using kmail, I was using Netscape Communicator. Although the Netscape Communicator has many useful features, Kmail had the features I really need. Plus, I use the KDE as my desktop environment and Kmail fits into the KDE architecture. (I will talk more about the KDE and many of the programs when I get the time.)

6.9 Webmin

Linux advocates regularly hear from fans of other operating systems about how unstable Linux is. They say there is no support for Linux and there are no applications. Some go so far as to claim Linux is nothing more that a collection of programs creating by a small group of hackers and inexperienced programmers.

The sheer number of Linux Internet servers attests to Linux's stability. Even a quick search of any of a number of Linux sites leads to hundreds of companies tht provide Linux support. The fact that major software vendors like Corel and Oracle have already ported their products to Unix, demonstrates that the applications are there. Looking glancing even at some of the names responsible for the Linux source code shows the quality of the people working on the various components of Linux.

All of these aspects seem to show that these statements of Linux opponents are blatantly untrue and demonstrate the ability of Linux to fit in well in most any environment. However, one place where Linux advocates often loose the battle is when talking about graphical administration tools. Especially when compared to Windows NT, Linux seems to be lagging behind.

Or so it seems.

In my mind, one of the of problems lies in the modularity of Linux. Although Linux is technically just the kernel, the name is now used to include the all of the files, scripts and programs that are delivered with the various distributions. However, no two distributions are identical and the tools each provides vary sometimes greatly. What this means is that the tools are often not always easy to find, which leads some people to believe that the tools do not exist. In some cases, the tools that are provided are lacking in some and functionality.

The real truth is that powerful graphical administration tools are not lacking. In fact, like many aspects of Linux, you actually have a choice of several different packages. It is just a simple matter of what tools you like working with.

One tool that I have grown fond of recently is Webmin, developed by Jamie Cameron. As you might be able to tell from its name, Webmin is used to administer your system using a Web browser. That means, you can administer your system from any system with a web browser. Webmin has taken this one step further by enabling you to administer a wide range of systems, including several different Linux distributions, Solaris, DEC OSF1, AIX, HP/UX, Irix and FreeBSD.

In essence, Webmin provides an mini-HTTP server written in perl, which creates the forms, processes the input, and executes the commands. Because you need to be root to make most of the administration changes to you system, Webmin needs to be able to do that as well. This means that Webmin runs by default with super-user privileges.

Some people may wince at the thought of allowing root access through a web browser. Although there are some potential security holes, Webmin has a number of different features which increase the overall security.

The first place where Webmin addresses the issue of security is by requiring an extra username and password to access the server. By default this is the user "admin" who has the same password as root. I would suggest that once you have Webmin installed, you change both the account name and the password.

Webmin also allows you to assign administration to different users. You can create additional users, to which you can then assign privileges to administer different aspects of your system. For example, it is possible to define a user or users who are allowed to just administer the printers, whereas another user can administer just DNS. This is done using the Webmin Users module, which also gives you an overview of which users have which privileges.

One of the basic security problems HTTP has is that the information is transferred across your network or the Internet in clear text. That is, it is possible to intercept the connection and read the administrator’s password. Webmin can easily protect against this by using the Secure

Socket Layer SSL, provided you have the Perl SSL libraries installed on your system.

The figure above shows you the initial start up page for Webmin. As you can see the interface is very simple, while at the same time being very practical. Behind each of the buttons is a different administrative function which is contained within a single Webmin module. One of the modules is used to administer the modules themselves. Part of the administration is to remove or install the modules as needed.

Because Webmin is modular, it is very easy to add your own modules, without the need of changing any of the existing scripts. Although the developer of a particular module needs to make sure the right components are available, anyone using the module can plop it in like a Netscape plug-in.

When witting your own modules, there are two requirements that need to be followed. First, there needs to be an icon for that module, which is stored as <module>/images/icon.gif. Second, there is the <module>/module.info. In both cases, <module> is the name of the particular module. The module.info file is a set of parameters, which take the form parameter

= value and contains information about the module, like its name, a description, what operating system it supports and so on.

By convention, the module, should produce a page which looks like the other Webmin modules. This can be done by using any programming language, such as C. However, one of the design goals of Webmin is to have it run unchanged on as many platforms as possible. Therefore, if you write a module in C, or the module uses any special programs, you will need to recompile on each new machine. By convention modules are written in perl, which makes them portable across platforms.

Webmin has a particular advantage for administrators who are either new to a particular aspects of administering a system or new to Linux in general, in that it already knows the syntax of the various configuration files. This ensures that that syntax is correct. I also know experienced administrators who use Webmin to setup the basic configuration and then edit the files by hand to make any additional changes.

The first step is to get Webmin from the Webmin home page, which is provided as a gzipped tar archive. When you unpack the archive it creates a sub-directory based on the version you have. For example, the current version as of this writing might create the directory **`/usr/local/webmin-0.73`**. This becomes the root directory for the HTTP server, so make sure you are extracting it in the right place before you go on.

Next change into the directory where the Webmin archive was extracted and run the script setup.sh. This is the primary setup/configuration script. This asks you a series of questions such as where to put the configuration directory for Webmin, which defaults to /etc/webmin.

The setup script also asks you your operating system, administrator's username and password, and other details about your existing configuration. Make sure that you choose the right operating system and, if available, the right version. This is extremely important as the location of the scripts and program, which Webmin uses, as well as their options, are be different among different operating systems. In addition, Webmin uses this information to determine what modules it should it include. If you don't get this right, Webmin won't work right.

During the setup process the script will also ask you if Webmin should be started when the system boots. This adds the Webmin startup script to the appropriate rc-directory i.e. **`/etc/rc.d/rc2.d`** to start in run-level 2. In addition, if you have a previous version of Webmin in the config directory, Webmin knows to upgrade it.

Part of the configuration process is to include the necessary modules. In many cases, the same module can be used for multiple operating systems with little or no changes. In other cases, there are specific modules for different operating systems. For example, there are separate modules to configure NFS on a number of different systems. This is one reason why it is important to chose the correct operating system during the setup.

If you look in the configuration directory, you will see that it contains a few scripts, text files and a large number of directories. Here you find the start and stop scripts, which are called from the appropriate rc-script if you have configured Webmin to start at boot time. The file miniserv.conf contains the configuration information for the mini-server, such as the port it

uses, hostname, whether SSL is used and so forth. Some of these values are assigned when you first setup Webmin and they can be changed using Webmin itself.

If you look at the directory name, it is fairly straightforward to figure out what each script does, even if you have never used Webmin before. There is a directory for each module, which contains the configuration information for that module. These directories mirror the directories under the server root directory, which contain all of the various perl scripts.

When you connect to the server using the defined port, the script index.cgi in the server root directory is run. This checks for which modules are installed and displays the necessary icons for each module. Since index.cgi is a script, the menu it presents is dynamic. Therefore, if a module is removed or added there is no need to edit any pages to reflect change you make.

The icons you see are hyperlinks to their respective directories. Here too the default page is the script index.cgi, which once again builds the page as appropriate based on the current configuration. These scripts are dynamic as well. Therefore, as I mentioned previously, it is possible to edit the normal system configuration files by hand and then re-load the configuration from Webmin. That means, there is no conflict if one administrator prefers to edit the files by hand and another chooses to use Webmin. When you access the particular module, the appropriate configuration files are read with any changes that have been made by hand.

With many of the modules, the first page is simply an overview of what can be configured. For example, clicking on the Samba button brings you the page in Figure 2. At the top is a list of the configured shares. Clicking on one allows you to configure that particular share. At the bottom of the page are the global configuration options.

There are two modules which I feel require special attention as they are not directly related to configuring your system. The first is the File Manager module which is just that. It is a Java applet, which provides you a full-featured file manager which affects the files and directories on the remote system the one being administered. This includes all of the expected features, such as copy, delete, move, rename, cut, paste, and so forth. You even have the ability to view text files.

Sometimes configuring the files through Webmin or even the File Manager is not enough. For example, you may need to execute commands on the remote machine. Webmin makes this a lot easier by providing you a Java telnet client. This means you don’t need to start an external program and can do it right from Webmin. Note that this is truly a telnet client, so if root is denied telnet access, it will also be denied through this Webmin applet.

As of this writing, there are 8 Webmin third party modules in addition to the over 30 modules that form the base product. The third party modules typically provide functionality which is only necessary for user with specific applications, such as managing the secure shell SSH, configuring the SAP router/proxy, administering MiniVend shops, and or managing Qmail or Zmail.

There is also a set of network utilities from Tim Niemueller http://www.niemueller.de/webmin-modules/nettools/ that use the Webmin interface to give you access to standard monitoring tools, such as ping, traceroute and nslookup. It also

provides an "IP subnet Calculator," which calculates the smallest possible network i.e. netmask for a given number of nodes.

Chapter 7 The X Windowing System

I've seen the X-Windows system described as the "distributed, graphical method of working," and that probably fits the best. It's distributed because you could run the display on your monitor in Virginia even though the program is actually running on a computer in California or Calcutta, and it's graphical because you see a lot of nice pictures on your screen.

Despite the extent to which it has spread in the UNIX world, the X-Windows system is not a UNIX product. The X-Windows system, affectionately called X, was developed by the Massachusetts Institute of Technology and runs on a wide range of computers, even MS-Windows-based versions.

The first version was developed at MIT in 1984. Several versions have been developed since, with the most current version, X version 11 (X11), first released in 1987. X11 has been adopted as the industry standard windowing system, with the support of a consortium of major computer industry companies such as DEC, HP, SUN, and IBM.

Although you could probably find a system that is still running release 5, the newest release (as of this writing) is release 6. You will see references to the release as X11Rn, where n is the release number. So, the current release would be X11R6.

In this section we are going to talk about the basics of the X-Windowing System, rather than the desktop environments like KDE and Gnome. The reason is quite simply that this material was first written in 1996 and neither KDE nor Gnome had really established itself. A lot of things have happened in the meantime and I just haven't gotten around to updating this. Any volunteers?

7.1 Configuring the X-Windows Server

On all current distributions (as far as I can tell), you will be getting a copy of the Xfree86 X-Windows system. Although this is almost completely compatible with commercial versions, this one is free like other products under the GNU public license.

Although you can get away with 4Mb of physical RAM and an additional 12Mb of swap space, you won't be happy. With this minimal configuration, you will probably get X started and a couple of windows open and then you will want to start swapping. Experience has taught me that without at least 16Mb of physical RAM, the system is too slow to be enjoyable to work in. Considering how low RAM prices have dropped, there really isn't any excuse any more to purchase more RAM.

When you install your copy of Linux, you will (should) be asked a series of questions about your video system to configure your X server. Even if you don't know what video chipset you use or the video card manufacturer, you can get away with using the standard SVGA card. However, the performance and appearance will dramatically improve if you are able to specify exactly what you have.

Even if there isn't an exact match, you can try something close and still get decent performance. If it is an exact match, I would recommend using a low resolution, like 640x480, to test the configuration. Once you are sure that everything works correctly, you can move to higher resolutions.

Once X-Windows is running, you can use the configuration program **xf86config**, which will again ask you a series of questions about your configuration. Here you really ought to know about the hardware, including your monitor. What hardware X-Windows supports is listed in the latest Xfree86 HOWTO, which you should find on your CD-ROM.

When you install your X server, note that you are not running just a single program. Instead, quite a few different programs are running. Which one runs depends on the options you specified during the configuration. Because most only run on a single video card or chipset, you definitely need to know about your hardware.

Keep in mind that just because you can run the Linux command line does not mean that Linux supports your video card. The command line is run in text mode, which uses well-known standard video modes to access the video card. However, once the X server is running, you are accessing the video card directly and need to know all the details.

Also available are several commercial X servers such as Accelerated-X and Metro-X, which provide better performance than the default X servers Xfree86 provides.

The primary configuration file for your X server is (normally) **/etc/XF86Config** or **/etc/X11/XF86Config**. This is a text file, which is generated new every time you run xf86config. This is broken down into three sections. The Screen section is the primary section and often comes last. It defines what you see on the screen based on the other two sections. The Device section describes your video card (which is often referred to as a video device). The Monitor section describes, as you might expect, your monitor.

Each section has a header line that defines what section it is and an EndSection line to close it up. The general form is

```
Section
"SectionName"
section info
EndSection
```

Because the X server decides what to show on the screen based on the Screen section, that is probably a good place for me to start. Within the Screen section, the server can give you several subsections for each of the "Display" types. The subsections are the logical configurations of your monitor and determine such things as the number of colors that can be displayed, the resolution, and whether there is a "logical" screen.

The Screen section on one of my machines looks like this:

```
Section
"Screen"
    Driver      "accel"
    Device      "SPEA Mercury 64"
```

```
    Monitor     "Sony17sf"
    Subsection "Display"
        Depth       8
        Modes
"800x600" "1024x768"
        ViewPort    0 0
        Virtual     800 600

EndSubsection
    Subsection "Display"
        Depth       16
        Modes
"800x600" "1024x768"
        ViewPort    0 0
        Virtual     1024 768

EndSubsection
    Subsection "Display"
        Depth       32
        Modes       "800x600"
        ViewPort    0 0
        Virtual     800 600

EndSubsection
EndSection
```

The Driver line indicates which X server will be used. In this case, I am using the "accel" driver for "accelerated" servers, which basically means that they have faster performance than other cards. The other kinds of drivers are vga2 (for vga cards in 2-color mode), vga16 (16-color vga), and svga (super-VGA, 256 color, 640x480).

The Device line indicates the name of the video card. Because that's the card I have, this line is set to "SPEA Mercury 64." The monitor indicates the monitor type. Note that in my case there was a specific entry for the SPEA Mercury card. However, there was no specific entry for my monitor, though one was close. The system uses this information to choose the best driver for you. However, you can still choose another driver.

As I mentioned previously, the Display subsection determines what is displayed on your screen. In this case, we have three different Display subsections, which are distinguished by the Depth line, which defines the color depth, or number of colors, that can be displayed. This indicates the number of bytes that are used to describe the colors. Therefore, in the first entry, we have 8 bits, or a total of 256 possible colors.

The Modes line defines the possible resolutions that your monitor can support. Normally, the lower the depth, the more modes the server can handle. In my case, the system did not configure this. Each of the modes has an entry for 640x480. Because I never wanted my server coming up in that mode, I was able to remove the modes. (Note that this is one option in the xf86config program.)

When it starts up, the X server will take the first entry it finds. In my case, this is 800x600 and 256 colors. However, you can use options to startx, which then passes the first entry on to xinit. If I wanted to increase the color depth to 24 bits, I could start the server like this:

startx -- -bpp 24

The Device section describes the characteristics of your video card. On my machine, it looks like this:

```
Section "Device"
    Identifier  "SPEA Mercury 64"
    VendorName  "Unknown"
    BoardName   "Unknown"
    VideoRam    2048
EndSection
```

The Identifier entry is used in other sections to match displays with Devices. Although the VendorName in this case is SPEA and the BoardName is Mercury 64, it does not matter that these two fields are empty.

Last, we get to the Monitor section. An except from the monitor section on my system follows (with *a lot* of things removed to save spaces). Note that you could have multiple Monitor sections if you were going to connect different monitors.

```
Section "Monitor"
Identifier  "Sony17sf"
    VendorName  "Sony"
    ModelName   "17sfII"
HorizSync   31.5 - 57.0
VertRefresh 50-70
# 640x400 @ 70 Hz, 31.5 kHz hsync
Modeline "640x400"     25.175 640  664  760  800   400  409  411  450
# 640x480 @60 Hz, 31.5 kHz hsync
Modeline "640x480"     25.175 640  664  760  800   480  491  493   525
# 800x600 @ 56 Hz, 35.15 kHz hsync
ModeLine "800x600"     36     800  824  896 1024   600  601  603  625
# 1024x768 @ 87 Hz interlaced, 35.5 kHz hsync
Modeline "1024x768" 44.9  1024 1048 1208 1264   768  776  784  817
```

Like the Devices section, the Identifier is used to match monitors and displays. Here the physical characteristics of the monitor are described, including the vertical refresh rate (how many times per second the screen can be redrawn) and the horizontal synchronization (which is based on the resolution and vertical refresh rate).

The most important part of the Monitor section are the modeline entries. If you have a common video card and monitor, you don't have to worry about this because the xf86config utility will create them for you. If you do need to create them, you should check the latest Xfree86 HOWTO.

7.2 The Basics of X

An X session is usually composed of several windows, each running a separate program or client. Like programs on any other system, programs running under X vary in functionality. Some interact completely with the user, like the XTerm terminal emulator. Others simply display output on the screen, like the xload system monitor.

The background window is referred to as the root window. Application windows, or clients, are displayed on top of the root window. Like UNIX processes, these windows are grouped together, or related, in a family hierarchy. As init is the great-grandmother of all processes, the root window is the great-grandmother of all windows. Clients displayed on the root window are children of the root window, and the root window is their parent. This hierarchy actually extends to different parts of a window. For example, menus are often considered children of the parent window as they inherit characteristics, but also can be configured and react independently of the parent window.

X consists of two sides: a server side and a client side. The basic functionality is similar to the way all client-server models work in that the X server has certain resources that it provides to the client. It is a common misconception that the server and clients are on the same machine. Because X is integrated with the TCP/IP stacks, requests can come from any client and can be requested of any server. In addition, because X is not a program but more a protocol, machines can communicate with completely different architectures. For example, a Digital OSF/1 server can provide services to both a Linux and an AIX client, as well as either of the others providing services to the OSF/1 machine. Just like other network applications, a single machine can be both client and server.

The server acts as the interface between the client programs and the physical hardware. When you input data through either the keyboard or pointer, the server accepts that input and is responsible for passing it along to the client. This information is passed to the client an *event*. Pressing a key or moving the pointer causes an event, to which the client may react. Often that reaction is in the form of changing the display on the screen. For example, a client receives the event that a particular menu was clicked on. It responds by requesting the server to display the pull-down menu. The server then passes the information on to the hardware, which shows the pull-down menu as a screen. It gives it to the server, which then passes it to the hardware. As a result of this separation of functionality, one client could display information on more than one server.

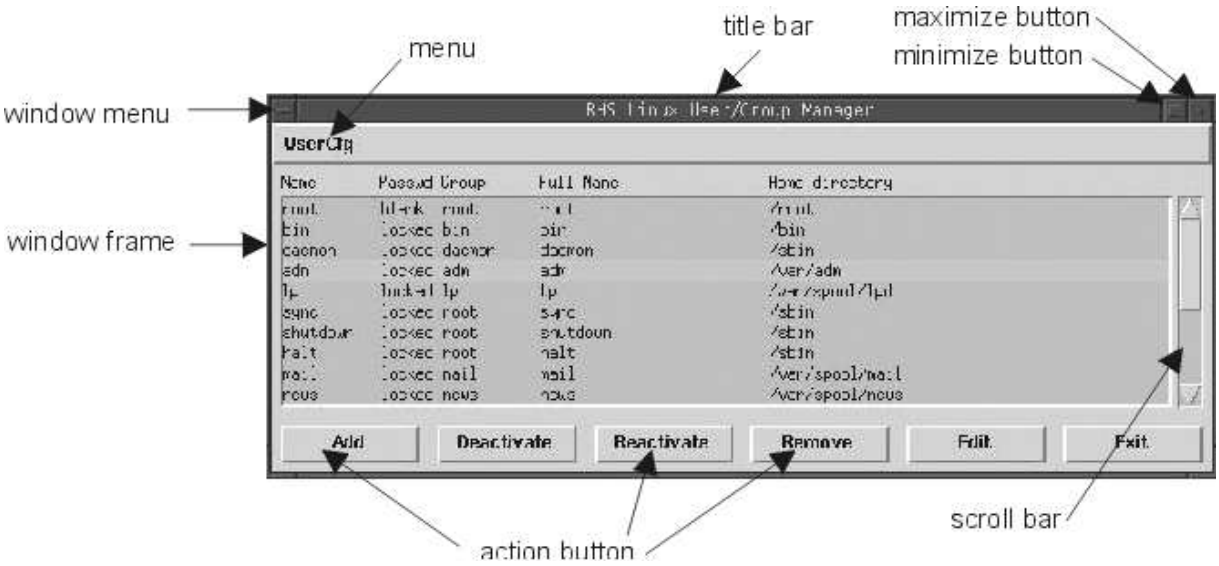
To start anything, an X server needs to be running somewhere. Despite that fact that you can access servers anywhere on the network, a common configuration is one in which the server is running on the same machine as the client.

Some systems have a graphic login that automatically starts when the system boots for example, the Common Desktop Environment, KDE, Gnome and so on. Another common way for the system to start is through the startx shell script, which reads the two files **`.xinitrc`** and **`.xserverrc`** file in your home directory and treats them in the same way as your shell would treat the .cshrc and .kshrc files. Here is where your initial clients are started, such as terminal emulator and the window manager. If you don't have a **`.xinitrc`** file in your home directory, then startx will read the system default file **`/etc/X11/xinit/xinitrc`**. In

reality, the X server is started by the **xinit** program. However, **startx** starts xinit for you.

Contrary to popular belief, neither the X server nor the clients are responsible for the appearance of the windows on the screen as we understand them. Instead, this falls to a window "manager." Most Linux distributions provide two window managers: The F? Virtual Window Manager **fvwm** and the Tab Windows Manager **twm**. In most cases that I have seen, the default window manager is **fvwm**.

What you can do to each part of a window is important in understanding the basic concepts of X. These parts are shown in Figure 0-1. A *click* is used to active a button. This is done by quickly pressing down and releasing one of the mouse buttons. Because there is only one mouse button on many systems, the mouse button used to click is usually button number one. On a right-handed mouse, this is the left button. A double-click is when the button is clicked twice in rapid succession.



Description of the Various Parts of a Window

To drag an object, select that object by placing the pointer somewhere on that object, then pressing down and holding the first mouse button. In many cases, such as in XTerm, you must click on the title bar. You then see the outline of that window, which you can move to a new location. You can also select the window by clicking Move in the Window Menu. To drop the object onto another, drag that object over another object and release the mouse button. This only works in appropriate circumstances, for example, dropping a document onto the printer icon to print it.

It can also be said that you, the user, manage the windows. You determine the size and location of the window, as well as determine which is the active window. You can change the size of the window in several different ways. By moving the pointer, you can *grab* any corner of the window by pressing and holding down the left mouse button. You can then move that corner in any direction, thus changing both the horizontal and vertical proportions of the window. You can also grab an edge and change the horizontal or vertical edge, depending on which edge you grab. In addition, you can choose the Size option from the window menu and

then move the pointer to the edge or corner with which you want to resize. This time, though, do not hold down the left mouse button.

There are also two buttons in the upper right hand corner of the window. The inner button is the maximize button. When you click it, the window will fill the screen it maximizes. When you click it again, it returns to its previous size not the default, but the size it was before you clicked the maximize button. The other button is the iconify button. This turns the window into a miniature version of its former self. This is a "representation" of that window. These little images are referred to as icons. Double-clicking the icon returns it to the size it was before you clicked it to iconify it.

When you choose which window is active, you *set the focus*. There are two types of focus policies used: explicit and pointer. In explicit focus, you must click somewhere within the window to set the focus. In pointer focus, the focus is set when the pointer enters a window.

If the default is explicit focus, I suggest you leave it as such until you are very familiar with moving around windows or have a compelling reason to change it. The problem with pointer focus is that you could be typing away in one window and accidentally push the mouse so the pointer is in another window allowing, all of a sudden, the new window to accept input. On slower machines, the opposite effect might happen. You may move the pointer intentionally to a new window and start typing. However, because the focus takes a moment to "catch up" with you, the input is sent to the previous window.

To change this, edit your .fvwmrc file and look for the entry that says AutoRaise. This item defines how long in milliseconds the system will wait until it automatically raises the window over which you have moved the cursor. This is pointer focus. Comment out this entry by placing a pound-sign # in front of the line. Just below it, is the entry ClickToFocus. This is the explicit mode. This means that you have to explicitely click on a window to change the focus.

In the **`.fvmwrc`** file, these focus modes are referred to as *auto-raise* mode and *focus-follows mouse* mode. Most of the other documentation refers to explicit and auto focus; use what you like.

Common to every windowing system at least every one I have ever seen is the concept of a menu. Like a menu in a restaurant, a menu in X presents a list of choices. Windows in X come in two types: pull-down and pop-up. Pull-down menus are almost universally associated with a particular location on the window. When you click on that location, a menu appears to drop down from that location. In a sense, you are pulling down that menu. By default, each window has Window Menu, which is a small square with a horizontal bar running through it, located in the upper left corner. Some people describe it as looking like a filing cabinet drawer with a handle. When you click on the Window Menu, you are give options that are related to the window itself. These include moving, resizing, or changing the windows position in the windows "stack" raising or lowering it.

Pop-up menus are usually not associated with any particular location on the window. These menus "pop-up" from the current cursor position. An example of a pop-up menu is the Root Menu that pops up anytime you click on an exposed area of the root window.

Earlier I mentioned that the window manager determines the "look and feel" of an application. This is not entirely true. Although what is presented is a function of the window manager, the underlying routines used to represent a button or a scrollbar can be different. Many of the Linux-provided clients use a set of routines called the X Toolkit Xt, which is actually two libraries the X Toolkit Intrinsics and the Athena Widget set [Xaw] used to create the interface components buttons, menus, etc., referred to as "widgets."

Keep in mind that X does not provide a graphical-user interface GUI. X is simply the windowing mechanism, but some other component provides the GUI. To produce such a GUI, the Open Software Foundation OSF developed the Motif Toolkit, which is based on the X Toolkit Intrinsics and a set of widgets developed by DEC and HP. This was originally designed to emulate the look and feel of the IBM/Microsoft Presentation Manager used in OS/2.

On Linux, you will find both Motif and standard X applications. Motif applications are those that use the Motif Toolkit and all have a common look and feel. One standard X application is the xclipboard. If you run it along with some other application such as xv a graphics viewer, you will notice some distinct differences, the most dramatic of which is the overall appearance. Motif-based applications appear three-dimensional, whereas standard X applications look "flat" two-dimensional.

7.3 Resources

If we wanted, we also could have included the geometry along with the colors, which would give us a command that is almost too long for the screen. Even now it is a long command that takes a long time to type in, and you can easily make mistakes. One solution would be to write everything in a shell script and start that script instead of typing everything on the command line.

The nice thing is we don’t have to do this. X provides a mechanism to change the appearance and sometimes the behavior of a client to fit our personal preferences. This is the concept of a resource. Up to now, we have specified the resource from the command line, such as foreground color and geometry. However, there are resource files that we can edit to change the default characteristics of a given client.

Resource files for most applications are found in **`/usr/lib/X11/app-defaults`**. The general form of the resource specification is

```
appname*subname*subsubname...:value
```

The application is the name of the program you are starting,usually with the first letter capitalized. Note the word "usually." I don’t know how many times I’ve tried to change a resource and not have it work, only to find out that this one applications name is written in lowercase. In the case of the files in **`/usr/lib/X11/app-defaults`**, no appname is necessary because there is one file for each client and X knows what client is meant when it reads these files. If set, X will search the path specified by the XFILESEARCHPATH variable for the resource information.

Unless you want to change the system defaults, I suggest that you leave these files alone. Instead, you can create a user- or machine-specific resource file. Normally, this is **$HOME/.Xdefaults--hostname**, where hostname is the name of the host to which these resource specifications apply. If the .Xdefaults file is to apply to the local host, you can omit the hostname. If you want to specify an alternative file, you can use the ENVIRONMENT variable.

These resources are organized into classes, which enables you to set groups of individual resources all at once. Individual resources are referred to as an instance. By convention,the class name begins with an uppercase letter and the instance begins with a lowercase letter. We can generally say that a resource (both class and instance)is named for the aspect of appearance that it controls. For example, the class called Foreground sets the foreground color. An instance of the Foreground class would be specified with a lowercase "F": foreground. Keep in mind that different parts of the clients are affected by the class Foreground, such as the text color, cursor color, and pointer color.

Basically all applications have resources. In each case, the class name has an initial capital. Examples of this are

background	window background color
border Width	width in pixels of the window border
border Color	windowborder color
foreground	window foreground color

The distinction between classes and instances is very useful if you want to set several resources at once. For example, if you define the foreground color for all aspects of the XTerm, the resource definition would look like this:

```
XTerm*Foreground: blue
```

This would be equivalent to

```
XTerm*foreground: blue
XTerm*cursorColor: blue
XTerm*pointerColor: blue
```

This means that the foreground color of text, cursor, and pointer are all blue. If we then defined the pointerColor instance to be something else, only it changes. For example, if we made the following definition

```
XTerm*pointerColor: red>
```

the color is now red, although all the others remain blue.

Although the asterisk is perhaps the most commonly used delimiter, it’s not the only one. The asterisk delimiter is used to indicate a *loose binding*, in which there can be several layers in

the object hierarchy. It's easy to think of the asterisk as having the same function as on the command line, that is, as a wild card. Here, the asterisk represents 0 or more intermediate layers between the root object and the resource we are defining.

If there are no intermediate layers between the objects, this referred to as a *tight binding*. If you wanted this, you *could* specify the binding with the asterisk because it means 0 or more intermediate levels. However, the symbol used to explicitly specify a tight binding is a dot (.). Because I know that the level just before the pointerColor in the hierarchy is "ansi," I can make the specification like this:

```
XTerm*.ansi.pointerColor: red
```

However, because the loose binding specifier (*) can be used any place though the tight binding specifier (.) can be used only when appropriate, it is easier always to use the loose binding specifier.

Both the resource specifications and binding can bring up some conflicts. In the example above, we said to use blue for *every* foreground color related to the client "XTerm." We also said to use red for the foreground color of pointer. Now this seems like a conflict, which it is. However, in this case, the instance of the pointerColor took precedence over the class of Foreground.

Consider these lines from an **`.Xdefaults`** file:

```
XTerm*Foreground: blue
XTerm*pointerColor: red
XTerm*ansi.pointerColor: green
```

We first defined the Foreground class to be blue. Next, we defined the instance of the pointerColor to be red. Both of these are done with loose bindings. We then defined the instance of the pointerColor for an ANSI terminal to be green. Because tight bindings have precedence over loose bindings, the pointer is green.

Taking this one step further, we change the class specification so it contains a tight binding. However, we leave the instance specification a loose binding. So, we end up with these two lines:

```
XTerm*ansi.Foreground: blue
XTerm*pointerColor: red
```

In this case, there is a tightly bound class specification that is followed by a loosely bound instance specification. When we start the XTerm, the pointer is blue, not red. In general, we can say that the more specific a specification is, the greater the precedence.

There are a limited number of options that we can use from the command line, although there are many more resources that we might want to change. To accommodate a large number of resource without increasing the number of options, we use the -xrm option. For example, if we wanted to change the tty modes of XTerm (what the characters are for erase, delete, quit, etc.), we could do this using the -xrm option and specifying an instance of the TtyModes class. For example, to change the interrupt key from the default of Del to Ctrl+C, the

command would look like this:

```
XTerm -xrm XTerm*ttyModes: intr ^C  &
```

Keep in mind that this resource specification is only valid for this one XTerm that we are starting here. If we wanted it to be valid for all XTerms, we would either change the default in **/usr/lib/X11/app-default**s or define the resource in the **.Xdefaults** file.

7.4 Colors

Although you may be satisfied with the default colors that X gives you, I am sure that eventually you will want to make some changes. In previous sections, I talked about how you can change the color of X clients either from the command line or by changing the appropriate resource. The only problem with that is you might not like the colors that Linux offers.

You might ask, "Why doesn't the system just give me a list with every possible color?" Well, you would need to have that list in a file somewhere. If you did, you would have a list that was more than 20Mb because of the way Linux stores colors.

Each color is represented by one byte for each of the three colors: red, green, and blue referred to as the RGB scheme. Each byte can have one of 256 values that represent the intensity of each color. In other words, the value represents how much of each color is included in a shade. If all three colors have the value 255, the shade is pure white. If each color has the value 0, the shade is black.

The **/usr/lib/X11/rgb.txt** file contains names of colors and, often, variations in that name. This is usually the case when the name of the color actually consists of two words, for example, antique white. In such a case, you would also find the color antique-white. Each entry contains the RGB values and the name of the color. For example, the antique white entry would look like this:

```
250 235 215 antique-white
```

This means that the intensity of red in this color is 250/255 of full intensity, the intensity of green is 235/255, and the intensity of blue is 215/255. What this really means is how much energy is sent to each phosphor. For details on what phosphors are and what part they play in displaying an image, see the section on monitors in the chapter on hardware.

If you specify the color as a resource either from the command line or a resource file, you specify the color as a hexadecimal value. The key thing to note is that you must specify the value for each color, even if it is 0. Because the hexadecimal values range from 0000 to FFFF, you have many more possible combinations of colors. When you specify colors in this way, the hex string you use must be preceded by a pound-sign #.

If you don't want to specify all four hexadecimal digits, you do not have to. However, all three colors need to be represented with the same number of digits because the system would not be able to tell what value goes with which settings. If we look at an example, this will be clearer.

Lets assume you want to set the intensity of red to F, the intensity of green to 4, and the intensity of blue to 2. You might then have a resource specification that looked like this:

*background: #F42

If we wanted the intensity of green to be 45 instead of 4, the resource specification might look like this:

```
*background: #F452
```

So what is it? Do we have red at F4, green at 5, and blue at 2? Or do we have red at F, green at 4, and blue at 52? The *only* way to keep things straight is if there are the same number of digits for each color.

Remember that not all video systems are created equal. You may not get the same color on your system as someone else does, even if you use the exact same hex values.

7.5 Displaying Clients

When the clients connect to the server, one key piece of information it needs is the display name. The display is of the form

hostname:displaynumber.screennumber

The hostname identifies the name of the machine to which the display is physically connected. The most common form of hostname is simply the node name, as more than likely the server is in the same network. However, it is possible to use a fully qualified domain or even an IP address for the hostname.

Unless you have some special hardware, you probably have only one physical display per server. However, each display is given a number starting at 0. If you only have one, then you will always access hostname:0. The screen number is only used in cases where a single keyboard and mouse are associated with multiple monitors. Like displays, screens are counted starting at 0. Because multiple screens are far less common than multiple displays, you can omit the screen number when specifying the display. Generally, the default display is stored in the DISPLAY variable, which is then used by default. However, many X clients have a -display option, with which you can specify the display.

The next important issue is the concept of *geometry*. One advantage of a system like X is the ability not only to move windows around the screen but also to change their size and shape as well. Rather than using the window manager to change the shape of the window, you can specify the shape and size when the application is started by specifying the clients geometry.

The geometry is represented by four characteristics: width, height, the distance from left or right, and the distance from the top or bottom. These are referenced by width, height, xoff, and yoff, respectively. Depending on the application, the height and width are measured in either pixels or characters, whereas the xoff and yoff values are measured only in pixels. Both xoff and yoff are measured in relationship to the screen. The general syntax of the geometry specification is

```
application -geometry widthxheight+xoff+yoff
```

Here the + (plus sign) before xoff and yoff indicate a distance from the left and top edges of the screen, respectively. By changing + to -, you change the offset to be from the right and bottom instead of left and top. For example, if you wanted to start the analog clock 30 pixels to the right of the upper left corner, the command would look like this:

```
oclock -geometry 90x90+30+0 &
```

(It's a good idea to run all clients in the background, otherwise you don't get your prompt back until the client terminates.) Now, if we wanted to start the clock 30 pixels to the left of the upper right corner, the command would look like this:

```
oclock -geometry 90x90-30+0 &
```

Now, if we wanted to start the clock 30 pixels to the left of the lower right corner, the command would look like this:

```
oclock -geometry 90x90-30-0 &
```

The four corners are thus mapped like this:

```
+0+0 -0+0

+0-0 -0-0
```

You can also specify negative offsets that would then start the client outside of the respective edge of the screen. For example, if we change the above command to look like this

```
oclock -geometry 90x90--30+0 &;
```

It will start the client so that the right edge of the clock is 30 pixels outside of the right edge of the screen. (Be careful not to have spaces in there.) This does not mean that the entire clock is outside of the right edge of the screen. This is a misconception that many people have (including me, at first). On many systems, there is something magical about the upper left corner of the client. Offsets from the edge of the screen are in relationship to this magical corner. This is not so with X.

A +xoff value is the distance of the *left* edge of the client from the left edge of the screen. A -xoff value is the distance of the *right* edge of the client from the right edge of the screen. This also means that a +yoff value is the distance of the top of the client to the top of the screen, and -yoff is the distance from the bottom of the client to the bottom of the screen.

Note that the geometry is specified in pairs. So, if you specify the height, you must also specify the width. Also, if you specify the x-offset, you must also specify the y-offset. However, you don't have to specify the offsets if you only want to specify the size. Therefore, you could start the clock like this:

```
oclock -geometry 90x90 &
```

This gives me a 90x90 clock at the default location. If you only want the offset to take the default size, it might look like this:

```
oclock –geometry +100+42 &
```

The thing that bothers me about this clock is that it is pretty boring. The colors are drab and it really doesn't have any life to it. The nice thing is that we can change the colors. With the analog clock, we can change several different things. If we wanted the background color to be cornflower blue, we would enter the command

```
oclock –bg cornflowerblue &
```

This creates an analog clock with the default size at the default location with a background of cornflower blue. However, it still looks boring. I want a foreground of red. So, lets run the command like this:

```
oclock –bg cornflowerblue –fg red &
```

Now it's beginning to have a little life to it. However, having both hands red is still not good enough. I want the hour hand red but the minute hand white, and I want the jewel at the top of the clock yellow. The command would then look like this:

```
oclock –bg cornflowerblue –hour red –minute white –jewel yellow
&
```

Thats not all. We can use a couple more options. However, these are listed in the oclock(X) man-page, so you can take a look there if you want. Other clients have different options because some of them don't make sense with an analog clock. For example, the digital clock (dclock) has an option to specify the font (-fn). Because there are no characters on the analog clock, an option to change the font wouldn't make sense.

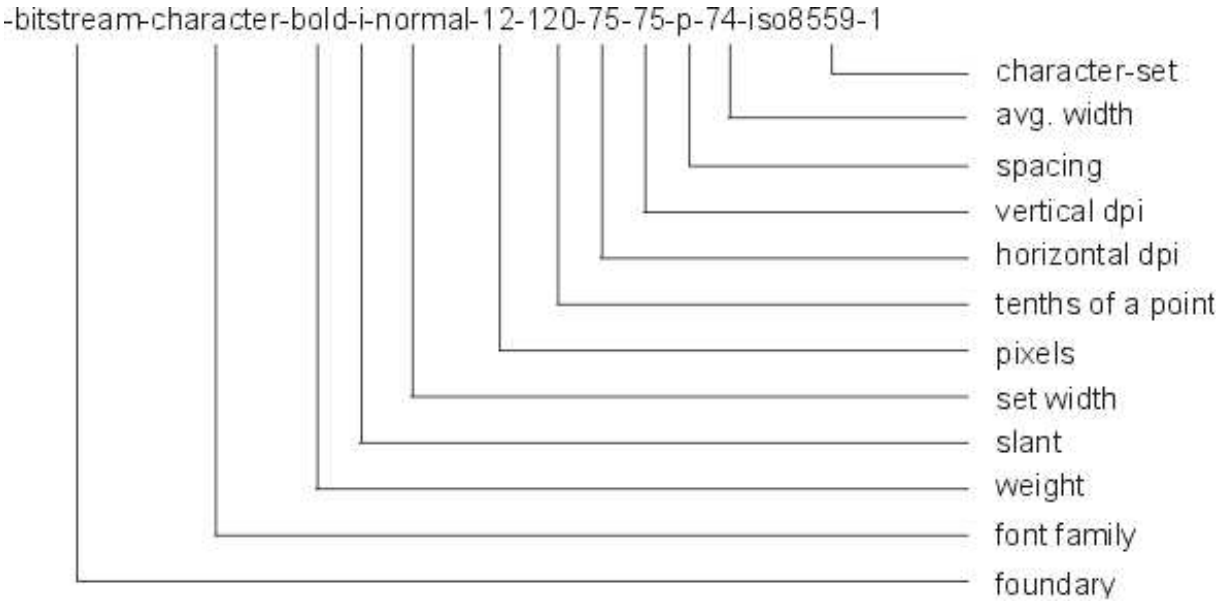
7.6 Fonts

Although not applicable to every client, fonts play a major role in many applications. Defined as a set of characters for displaying text and symbols, fonts share a common appearance in terms of size, boldness, and other physical characteristics. Fonts themselves can be grouped together into font families. Additionally, font families are grouped by resolutions (dots-per-inch, or DPI) into directories. Font families are so named because they were initially stored together in the same directory in the file system. Each directory contains a database that the server uses to translate font names into data that the server uses to display the characters on the screen. How the name of a font is translated, we see in Figure 0-2.

If the X client has a font menu like MS-Windows or Macintosh, life would be easy when it came to fonts. Instead, you need to choose the font as you start the application. If you were tired of the boring font used by default on XTerms, you could choose something a little more fancy, perhaps one that looked like cursive. For example, we could start xterm like this:

```
xterm -fn
-bitstream-charter-bold-i-normal--12-120-75-75-p-74-iso8859-1
```

At first, this appears rather intimidating. There are several hundred fonts on an Linux system and learning them all is a pain. More than that, it is a waste of time. Two utilities make like easier for you: xlsfont and xfontsel. The xlsfont utility simply lists all the available fonts with their complete name, as in the previous example. The **xfontsel** is a real X client that enables you to pick and choose a font based on different criteria. What those criteria are is helpful in understanding more about fonts.



Characteristics of the Font Name

The *foundry* is the fonts developer. Here we have bitstream, which, as one might guess, is from the company Bitstream, the same people who develop so many fonts for MS-Windows. The font family (here, charter) is a convenient way of organizing the fonts by certain appearance characteristics.

The weight of a font can be thought of as its thickness. Common weights are medium and bold. The slant, as you might guess, is the change in orientation of the character from the vertical. A Roman slant is upright, italic is tilted, but the characters are given a slightly different shape to make them more esthetically pleasing, and oblique is just tilted with no changes to the characters shape. In the example in Figure 0-2 we have an italic slant.

The set width is a general description of the average width. Common set widths are normal, condensed, and double-width. The size of the font on the screen is determined by several aspects of the font name. These characteristics are the pixels, points, and both vertical and horizontal DPI. Because the appearance on your screen depends on your monitor as well as what fonts you choose, it's safe to gauge the font size by the points. A point is an old printers measurement that represents 1/72 of an inch. In the example in Figure 0-2 we have 120 tenths of a pitch, therefore the size of each character is 1/6 of an inch.

Another important characteristic is the font spacing, which determines whether the font is *proportional* or *monospaced*. A proportional font is one in which the width of each character is different. Although this looks good on paper or on a word processor, it is not really suited for applications like terminal emulators. The monospaced font, in which every character takes up the same space, is better for such applications.

The character set is basically another way of saying what letters are represented. In this example and most others in Linux, this field will be iso8859-1, which represents the ISO Latin 1 character set, which is a superset of the standard ASCII character set. In addition to American English characters, iso8859-1 contains the special characters used in most European languages.

So now that we know what goes into font name, we can easily come up with the right font. Well, maybe. Fortunately, we don't have to. We can use a wild card for the parts of the font that we either don't think are important or don't want to guess at. Any one of the specifications can be wild carded and the system will do it's best to find a match. By "do its best," I mean that there can and will be cases in which multiple fonts match the specification. A rather simple example would be:

```
XTerm –fn –bitstream-charter*
```

On most systems there are 60 matches. So, which one does the system choose? Easy enough: the first one it finds. Unless you are more specific or know that the first one the system will find is the font you want, you might not get the font you want. Fonts are sorted in alphabetical order, and because bold comes before medium, we get the bold version of this font instead of the medium.

Pop quiz: Why did we enclose the font name in this example inside single quotes though we didn't in the first example? Remember that the shell expands everything into tokens before it passes things off to the command. If we didn't use the single quotes, the shell would try to expand the font name and we would get a message indicating that the system cannot find that font. Details on using quotes can be found here.

Life is even simpler than that. We don't need remember any long, drawn-out font names or try 20 different combinations to find the font we want. We can take advantage of the fact that the system understands font aliases, which are stored in the font directories in the file fonts.alias. These files are ASCII files with the alias in the first column and the font name in the second column.

There are two things to keep mind. First, although you can edit these files and make changes, the system will not recognize the new alias unless you reset the font path with the **xset** command, which is simply done as;

xset fp

Next, unless you are absolutely positive about what fonts each user is using and how they are being referenced, it is not a good idea to remove aliases. If you remove an alias that some client is expecting, the results are unpredictable.

If you always want to use a particular font with a particular application, you don't need to always specify the -fn option when starting the client. Because the font is just another resource, you can instead make the change to your resource definition file. An example in our .Xdefaults file might look like this:

```
XTerm*font: ibm10x20
```

If you looked though the directory files, you wouldn't find a font simply named ibm10x20; this is actually an alias. However, you could specify the full name of the font. Also, just like specifying the font from the command line, we can use wild cards to specify the "we-don't-cares."

The directories in which the fonts are found by the server are referred to by the font path. This is the XFONTS variable, which defaults to /usr/lib/X11/fonts but can be changed using the xset command. Because the server displays the information on the screen, the font path is on the server, not on the client machine. Therefore it is important to ensure that the server is able to display a particular font before changing it for any given client. There are five subdirectories under /usr/lib/X11/fonts varying in size and appearance.

The font database is contained in the fonts.dir file in each directory. You use this database when you select a font. The system knows what file to read to be able to show the proper characters on the screen. The fonts.dir files are ASCII files with the name of the font file in the first column and the font name in the second column. When the system is installed, the mkfontdir reads the font files found in the font path and creates the fonts.dir files. You can use mkfontdir yourself, but the details of fonts creation and management goes beyond the scope of this book.

Rather than requiring every machine on the network to have a full compliment of fonts, the system has something called a *font server.* Like a file server that provides files across the network, a font server provides fonts across the network. Just like files, if the font you want is not available by the server but is available locally, there is no problem. You can access it as well.

The font server program, **xfs**, is not started by default but can be started in one of several different ways. Although starting it manually might be good for testing purposes, it is more efficient to have the font server start up every time the system goes into multiuser mode. As with many of the differenromant aspects of the system, this is accomplished through a script in the **/etc/rc.d** directory. However, there is no script there by default. Obviously, you could write the script yourself, but you ought to let the system do it for you.

Starting the font server from the command line is not recommended for everyday use. To be able to use the fonts provided by the font server, you need to tell your X session about it. The best place to do this is inside your **$HOME/.xinitrc** file. Although the system administrator (you?) can change the **/etc/xinitrc** file, everyone using the default gets to use it, so you need to remember those people who already have their own .startxrc file. Before any clients are started, use the **xset** command to specify where the font server is. The general syntax is

xset fp=tcp/server:port

where the server is the name of the machine on which the font server is running, and port is the TCP broadcast port, which is 7000 by default. For example, to access the font server on the machine siemau, the command would be

xset fp=tcp/siemau:7000

Or, if you want to use local fonts as well, the line might look like this:

xset fp=tcp/boston:7000,/usr/X11/fonts/100dpi

The font servers configuration file is **/usr/lib/X11/fs/config**. Here you can limit what fonts will be made available through the font server by changing the catalog entry and specifying the full paths to the directories with the appropriate fonts. For example, if you only wanted to have access to the 100 dpi fonts, the line might look like this:

```
catalogue = /usr/lib/X11/fonts/100dpi
```

To make the changes take effect, either stop and restart the font server or use the **/etc/fontserv** command to re-read the configuration file and flush the cached entries:

fontserv re-read

fontserv flush

Like name servers in TCP/IP, which get name information from other name servers, you can have font server get fonts from other font servers. This is also done with the catalogue entry. For example, if I wanted to make the local 100 dpi fonts available, as well as the from the remote host scoburg, the entry might look like this

```
catalogue = /usr/lib/X11/fonts/100dpi,tcp/scoburg:7000
```

assuming that scoburg has its font server configured to use port 7000. Changing the port is also accomplished by changing the **/usr/lib/X11/fs/config** file. On a line by itself, add a line specifying the appropriate port. For example, if you wanted to change it to 7042, the entry would look like this:

```
port=7042
```

Once the change is made, the font server needs to be stopped and restarted. If any other machines were using this server, they need to be told that it is now set for port 7042.

You can use the -cf option when starting the font server to specify an alternate configuration file. Reconfigure any X servers that use the font server. Note: Use care when you reference other font servers. Font server connections place a heavy demand on network resources and bandwidth. Also, be careful not to let the number of references to other font servers become so large that your system font server becomes unmanageable.

7.7 The Window Manager

Because the window manager is an important aspect of X, I need to cover it in more detail. As I mentioned earlier, in most versions of Linux, you will find a couple of window managers. The fvwm seems to be the most common, although I do see twm quite often.

The following sections describe the basic default behavior of windows, icons, the icon box, input focus, and window stacking. The appearance and behavior of the window manager can be altered by changing the configuration of specific resources.

One way to control your windows is though *accelerator keys* also called hotkeys. By default, several accelerator keys perform various functions. Its quite possible that on your system these bindings have been commenting out of your .fvwmrc file. I'll get to how to change entries in a moment.

These functions probably can be reached through the window menu as well. It all depends on what is configured in your .fvwmrc file. Any windows managed by fvwm will have these keys, which are explicitly defined and can be changed by modifying the appropriate resource which I'll get to in a moment. These keys are

Alt+F1	Run popup "Utilities"
Alt+F2	Run popup "Window Ops"
Alt+F3	Run FvwmWinList Module
Alt+F4	Iconify the window
Alt+F5	Move the window
Alt+F6	Resize the window
Alt+F7	Circulate window up
Alt+F8	Circulate window down
Alt+F9	Iconify the window

The window manager is also responsible for managing icons. As I mentioned earlier, icons are small graphic representations of clients. Iconifying turning into an icon a client is a good way to reduce the space taken by clients that are currently not being used. Although the pager enables you to move clients from one virtual screen to another, icons are a good way of being able to instantly access a particular client. This is done with an "icon box." The icon box that you will find on your system is called FvwmIconBox and, in most cases, you will have to configure the system to start it. I'll discuss this shortly.

A nice thing about the icon box is that it represents all running clients, not just those that have been iconified. This makes finding the client you want easy, because by double-clicking its icon in the icon box, it immediately is made the active window and is brought to the front.

If you have a lot of icons, using the icon box is a good way to manage them. The author of the FvwmIconBox describes it as a "clutter-reduction program." The icon box is simply another client that manages the icon images. The icon box can be moved and resized, even iconified. If there are more icons in the icon box that can be shown in the box, scroll bars will appear.

One strong point of both window managers is the ability to configure them to our tastes. Up to now we have basically been talking about the appearance of fvwm. What is really fun is to

modify its behavior. One way to accomplish this is through the resource description file, which contains descriptions of resources that are easily defined in the resource files. The descriptions of resources include such things as the behavior when buttons are pressed and the individual entries in each menu.

The default resource description file is **/etc/X11/fvwm/system.fvwmrc** or **/etc/X11/fvwm/system.twmrc**. Here again, unless you want to institute systemwide changes, I recommend that you copy the appropriate file into the users home directory. This copy then has the name **$HOME/.fvwmrc** or **$HOME/.twmrc**. Here is where your default menus and default bindings are defined. When you click the root window, the root menu pops up, as defined in the **.fvwmrc** or **.twmrc** file, which is very easy to modify to your personal tastes and preferences.

Three types of resources can be described here: buttons, keys, and menus. It is said that window manager functions are *bound* to button or key-press events. The relationship between the button or key press is called a *binding*.

Because the resource description file is an ASCII text file, it is easy to edit to make the changes you want. The format of each type of resource is slightly different, but in each case, the fields are separated by white spaces. Any text from an unquoted pound sign # to the end of the line is considered a comment. Therefore, if any description must be contained the #, it must be quoted. Single characters can be "quoted" by escaping them using the back-slash. Any line containing a ! exclamation mark as the first character is also treated as a comment.

When an event occurs button or key is pressed or menu item is selected, a particular window manager function is called. In general, we can say that the functions have the following syntax:

```
function function_arguments
```

You can call dozens of functions that relate to everything from resizing the particular window or icon to shuffling the order, moving, and all the other functions we talked about. All of these are detailed in the fvwm man-page, so I don't feel the need to cover them all. However, I will discuss some of the more common functions as I describe the syntax of the resource descriptions.

The first thing we'll talk about is the idea of a popup menu. This is not like the menus in most programs where you see a list of options at the top of the screen and when you click the list, a menu suddenly appears. These are called pull-down menus because you pull them down from the menu list. Popup menus seem to popup out of nowhere. With other window managers, pop-ups are referred to as menus.

You will probably see on your system that when you click an open area of your desktop with the left mouse button, a menu called "Program Menu" will pop up. This menu is defined in your .fvwmrc file. Each popup menu is defined with the following syntax:

```
Popup "Popup Name"
functions to call
EndPopup
```

In the case of our "Program Menu," the pop-ups name is actually "Applications," so look for the line

```
Popup "Applications"
```

When you find it, you will see all the same entries that would appear when you click the desktop. When you click one entry in the list, that particular program or module will start. A module is a program used by the window manager and cannot be started in any other way.

You'll see that some of the entries have an arrow on the right side. When you click the arrow, it brings up another popup that is defined somewhere else in the .fvwmrc file. These pop-ups could then have entries that go to other pop-ups. Although I have tested this to five levels deep, I have never really had a need to go beyond three levels of popup menus.

To start a particular program or module from within the menu, use the Exec function. The syntax for the Exec definition is

```
Exec name  exec <arguments>>
```

An example would be

```
Exec "k3b" exec k3b &
```

The name in each line is the symbol that will appear in the menu pane for that entry, here, "K3B." Labels containing spaces or tabs must be enclosed within double quotes, as in our example. To start another popup, the syntax would be

```
Popup "Menu Entry"  Popup_Name
```

Here the "Menu Entry" is just what appears in the menu. The Popup_Name is what it actually will be called.

The title function within a popup is what appears at the top of the menu when it's started. For example, our applications popup was called "Applications" and that is how the system referred to it. However, when we started it, the words "Program Menu" appeared at the top. This is the title of the menu and has no other function.

If the function called is the no-operation function Nop, the function is invalid and/or is the equivalent of adding a separator in other window manager. If you use just two double quotes " " as the name, you will see a line between entries. However, you could include a line of something like an equal sign = to give your menu a different effect.

The accelerator keys syntax is

```
Key <keyname> <context> <modifiers> <function>
```

Here <keyname> is the name of a key like F1 for the F1 function key or Tab for the Tab key. The <context> is when this key press should be valid. Unlike mouse actions, a key press should be valid in all cases, so you can use an A. The <modifiers> is used for an addition key that should be pressed as well, such as "C" for the Ctrl key, "A" for the Alt key, "M" for the Meta key, and "S" for the Shift key. As I mentioned earlier, your accelerator keys may have

been commented out, so look for the section starting "Keyboard Accelerators."

One example looks like this:

```
Key F1 A M Popup "Utilities"
```

Note that in contrast to key presses, key bindings to window manager functions are just for the key *presses*. Key releases have no meaning in this context. Also, the modifiers are exclusive, which means that no other modifier key can be pressed. Because I specified just Meta+F1 and not SHIFT+Meta+F1, pressing SHIFT+Meta+F1 would have no effect unless I had defined it to something already. If you want, you could use something like "SM" to indicate pressing both the Shift and Meta keys.

Each button binding has the following syntax:

```
Mouse button context modifier function
```

The button field defines what button to which this function should apply 1, 2, 3. The context is when the particular function should be in effect. Valid contexts are:

A	A -Any context except for the title bar
R	Root window
W	Application window
T	Window title bar
S	Window side, top, or bottom
F	Window frame corners
I	Icon window

You can define certain characteristics of the windows. You can define certain characteristics, such as the foreground and background color, geometry, etc., when the client is started or by defining that particular resource, using the Style command within .fvwmrc. In many cases, resources define the same characteristics. See the fvwm man-page for details.

The syntax of the Style command is

```
Style <windowname> <options>
```

The <windowname> can be something other than the windows name, for example, a class or resource string. The <options> are a common separated list of values. Some options require arguments like Icon, which requires the name of the icon that you want to use when the window is iconified.

7.8 Remote Access

One of the powers of Linux is the wide range of features you can use to remotely access systems. Using **`telnet`** (or better yet, **`ssh`**) you have the full command line features you would if you were logged in locally.

One disadvantage of this is when running graphical applications that share data. If you are accessing a database, network protocols allow you to share a common data source. Problems arise when using applications that do not have the built in features. You could save your data

locally and copy it to the remote machine, or you could mount a remote filesystem. Both are possible and even useful, from time to time. The X Windowing system allows you to go one step further by running the application on the remote machine and have it *appear* as if it is running locally. The keyword is "appear" as only the display (that is, the appearance) is local.

For those of you who are familiar with the Microsoft Windows Terminal Server or products like Cirtix' Metaframe, the X Windowing protocol is similar in functionality. One key difference is X is much smaller allows you to work on slower connections. The X Windowing protocol is also an open standard and not propriatary, unlike the Windows Terminal Server or Metaframe.

Another key difference is the ability to redirect when the application is displayed. For example, I can tell the X application to start on a compeltely different machine. That is, not the machine where my X Windows sessions is running or where the application is running.

There are two basic ways of using X Windows to start applications on remote machines.

7.8.1 XDMCP

As its name implies, the X Display Manager Control Protocol (XDMCP) is used to manage your display. One thing this means is that XDMCP is capable of provide an X server with a *graphic* login from a remote machine, and behave as if you had actually logged in locally to that machine.

In practice, this has many applications. For one, several years ago it was common to find X-terminals, which were little more than a monitor, keyboard and mouse. Like the serial terminals you had a remote connection to a server. Instead of a simple character-based session, X-terminals provided a graphical sessions. Although the use of X-terminals has declined in recent years, the same basic principle can be used with older PCs. As long as the PC is capable of running an X-server, it can run XDMCP. This is is useful in many cases where you cannot afford new computers for each user, but still want to give the access to new applications

Although my laptop is capable of running a full version of Linux (and it does), I take advantage of this same capability at home. Sometimes when my family is watching TV, I would like to work on things (for example this site) in their company. I could copy the site on to the laptop or telnet to my server and change the display on my web development environment (Quanta) to the laptop. However, when I am working on my laptop like that, *all* of the work is being done on the server. That would mean that for every application I have to redirect the display to the laptop. Instead, I use XDMCP to provide me a graphical login on the server and then the entire behaves as if I were logged in locally to the server.

In this situation, all that is really necessary on the laptop (or PC) is the X server. All of the other applications, be it my web development environment, KMail or anything else is running on the remote machine. Since everything is running on the server, the local machine can run with far fewer resources than if all the applications were running locally. (Note in this discussion I have been referring to the server as the machine on which the applications run. However, when talking about the X-Windowing System, the server is the machine with the

display.)

As far as X is concerned, the X terminal will be running nothing but an X server. This X server will be configured to talk to a remote host using XDMCP (the X Display Manager Control Protocol). It will ask the remote host for an X session. The remote host will put up a login window on the X terminal, and after login it will run an X session with all bells and whistles, including the window manager, all using remote X to display on the X terminal.

You will probably notice that the remote host is acting like a server, though not an X server. The remote host is providing X sessions to X servers that ask for one. So, with respect to XDMCP, the remote host is actually a server, providing X sessions, also known as an XDMCP server. The X server is playing the role of an XDMCP client! Are you still with me?

The program that provides the XDMCP service on the XDMCP server is xdm. So, in order to get an X terminal up and running, you must configure two programs: X (the XDMCP client) on the X terminal, and xdm (the XDMCP server) on the remote host.

You must always remember that the X protocol (and the XDMCP protocol) are not encrypted. If you use remote X, everything that goes over the network can be sniffed by other hosts on the network. This is especially bad with remote X sessions, since the first thing that happens is logging in by giving a username and password. So, you must run remote X over a trusted network only!

Chapter 8 The Computer Itself

During the several years I spent on the phone in tech support, it was common for people to call in with no idea of what kind of computer they had. I remember one conversation with a customer in which he answered "I don't know" to every questioned I asked about his hardware. Finally, he got so frustrated and said, "Look! Im not a computer person. I just want you to tell me what's wrong with my system."

Imagine calling your mechanic to say there is something wrong with your car. He asks you whether the car has four or eight cylinders, whether it has fuel injection, whether it is automatic or manual, and whether it uses unleaded or leaded gas. You finally get frustrated and say, "Look. I'm not a engine person. I just want you to tell me what's wrong with my car."

The solution is to drive your car to the mechanic to have it checked. However, you can't always do that with your computer system. Dozens of people rely on it to do their work. Without it, the business stops. To better track down and diagnose hardware problems, you need to know what to look for.

This section should serve as a background for many issues I've covered elsewhere. This chapter is designed to familiarize you with the concepts rather than make you an expert on any aspect of the hardware. If you want to read more about PC hardware, an excellent book is the *Winn Rosch Hardware Bible* from Brady Books (its more than 1000 pages and, as of this writing, it's in it's third edition).

In the following sections, I will be talking primarily about PC hardware. Many of the concepts are the same as on Alpha machines or Macs, but when I talk about specific interactions with the hardware, they probably only apply to the PC, for two reasons. Despite the fact that Linux runs on several platforms, it was first developed on the PC and only recently successfully ported to the other architectures. The second reason is that my expertise is in PCs. I have several of them myself and have worked with them for years, so I have the experience to know what I am talking about.

In addition, the first commercial port to the Alpha is fairly recent. Therefore, there are not as many people using them. However, keep in mind that although the DEC Alpha is a different processor, the rest of the hardware is usually the same.

8.1 Basic Input-Output Services and the System Bus

A key concept for this discussion is the bus. So, just what is a bus? In computer terms, it has a similar meaning as your local county public transit, as it is used to move something from one place to another. The county transit bus moves people; a computer bus moves information.

The information is transmitted along the bus as electric signals. If you have ever opened up a computer, you probably saw that there is one central printed circuit board with the CPU, the expansion cards, and several chips sticking out of it. The electronic connections between these parts is referred to as a bus.

The signals that move along a computer bus comes in two basic forms: control and data. Control signals do just that: they control things. Data signals are just that: data. I will get to how this happens and what each part does as we move along.

In todays PC computer market, there are several buses, many of which have the same functions but approach things quite differently. In this section, I am going to talk about the different bus types, what goes on between the different devices on the bus, and what the main components are that communicate along the bus.

Despite differences in bus types, certain aspects of the hardware are common among all PCs. The Basic Input Output System BIOS, interrupts, Direct Memory Access channels, and base addresses are just a few. Although once the kernel is loaded, Linux almost never needs the system BIOS. However, understanding the function and purpose of the BIOS is useful in understanding the process that the computer goes through when booting. That is, from the time you hit the power switch to when Linux has full control of the hardware.

The BIOS is the mechanism DOS uses to access the hardware. DOS or a DOS application makes BIOS calls that then transfer the data to and from the devices. Except for the first few moments of the boot process and the last moment of a shutdown, Linux may never use it again.

The standard BIOS for PCs is the IBM BIOS, but that’s simply because "PC" is an IBM standard. However, "standard" does not mean "most common," as there are several other BIOS vendors, such as Phoenix and AMI.

DOS or a DOS application makes device *independent* calls to the BIOS to transfer data. The BIOS then translates this into device *dependent* instructions. For example, DOS or the application requests that the hard disk read a certain block of data. The application does not care what kind of hard disk hardware there is, nor should it. It is the BIOS’s job to make that translation to something the specific hard disk can understand.

In Linux, on the other hand, a special program called a device driver handles the functions of the BIOS. As we talked about in the section on the kernel, device drivers are sets of routines that directly access the hardware, just as the BIOS does.

The fact that Linux by-passes the BIOS and goes directly to the hardware is one reason why some hardware will work under DOS but not under Linux. In some instances, the BIOS has been specially designed for the machine on which it runs. Because of this, it can speak the same dialect of "machine language" that the rest of the hardware speaks. However, because UNIX does not speak the same dialect, things get lost in the translation.

The Intel 80x86 family of processors has an I/O space that is distinct from memory space. What this means is that memory or RAM is treated differently than I/O. Other machine architectures, such as the Motorola 68000 family, see accessing memory and I/O as the same thing. Although the addresses for I/O devices appears as "normal" memory addresses and the CPU is performing a read or write as it would to RAM, the result is completely different.

When accessing memory, either for a read or write, the CPU utilizes the same address and data lines as it does when accessing RAM. The difference lies in the M/IO# line on the CPU. For those not familiar with digital electronics, this can also be described as the "Memory/Not IO" line. That is, if the line is high there is a signal on the line, the CPU addresses memory. If it is low no signal, it addresses an I/O device.

Although the Linux operating system is much different from DOS, it still must access the hardware in the same fashion. There are assembly language instructions that allow an operating system or any program for that matter to access the hardware correctly. By passing these commands the base address of the I/O device, the CPU knows to keep the M/IO# line low and therefore access the device and not memory.

You can often see the base address of each device on the system when you boot. The hardware screen shows you the devices it recognizes along with certain values such as the base address, the interrupt vector, and the DMA channel. You can also see this same information by looking in the /var/log/messages and several files in the /proc file system.

If your motherboard only uses 10 address lines, devices on the motherboard that have an I/O address such as the DMA controller and PIC will appear at their normal address, as well as at "image" addresses. This is because the higher 6 bits are ignored, so any 16-bit address in which the lower 10 bits match will show up as an "image" address. Because 6 bits are ignored, there are 63 possible "image" addresses 64 minus the one for the "real" address.

These image addresses may cause conflicts with hardware that have I/O addresses higher than 0x3FF 1023, which is the highest possible with only 10 address lines. Therefore, if your motherboard only has 10 bits of I/O addresses, you shouldn’t put devices at addresses higher than 0x3FF.

When you install, it is vital that no two devices have overlapping or identical base addresses. Though you can share interrupts and DMA channels on some machines, you can never share base addresses. If you attempt to read a device that has an overlapping base address, you may get information from both devices.

If you are installing a board whose default base address is the same as the one already on the system, you must change one address before they both will work. Additionally, you are almost always asked for the base address of a card when you install it. Therefore, you will need to keep track of address. See the section on troubleshooting for tips on maintaining a notebook with this kind of information.

The table below contains a list of the more common devices and the base address ranges that they use.

Table - Common Hex Addresses

HexRange	Device
000-0ff	Motherboard devices DMA Controller, PIC, timer chip, etc.
1f0-1f8	Fixed disk controller WD10xx
278-27f	Parallel port 2
2f8-2ff	Serial port 2
378-37f	Parallel port 1
3bc-3bf	Monochrome display and parallel port 2
3c0-3cf	EGA or VGA adapter
3d0-3df	CGA, EGA, or VGA adapter
3f0-3f7	Floppy disk controller
3f8-3ff	Serial port 1

8.2 The Expansion Bus

It is generally understood that the speed and capabilities of the CPU is directly related to the performance of the system as a whole. In fact, the CPU is a major selling point of PCs, especially among less-experienced users. One aspect of the machine that is less understood and therefore less likely to be an issue is the expansion bus.

The expansion bus, simply put, is the set of connections and slots that enable users to add to, or expand, their system. Although it’s not really an "expansion" of the system, you often find video cards and hard disk controllers attached to the "expansion" bus.

Anyone who has opened his or her machine has seen parts of the expansion bus. The slots used to connect cards to the system are part of this bus. Note that people will often refer to this bus as *"the bus*." Though it will be understood what is meant, there are other buses on the system. Just keep this in mind as you go through this chapter.

Most people are aware of the differences in CPUs, whether the CPU is 16, 32 or 64-bit, what the speed of the processor is, whether there is a math co-processor, and so on. The concepts of BIOS and interrupts are also commonly understood.

One part of the machines hardware that is somewhat less known and often causes confusion is the bus architecture. This is the basic way in which the hardware components (usually on the motherboard) all fit together. Linux will run on several different kinds of buses. The most common are those in PCs, which I will talk about first. (Note: Here I am referring to the *main* system bus, although Linux can access devices on other buses.)

The three major types of bus architectures used are the Industry Standard Architecture (ISA), the Extended Industry Standard Architecture (EISA), and the Micro-Channel Architecture (MCA). Both ISA and EISA machines are manufactured by a wide range of companies, but only a few (primarily IBM) manufacture MCA machines. As of this writing, no commercial distributions are available for MCA, but a development project is underway.

In addition to these three architectures, a few other bus types can be used in conjunction with or to supplement the three, including the Small Computer System Interface (SCSI), Peripheral Component Interconnect (PCI), and the Video Electronics Standards Association Local Bus (VLB or VL-Bus).

Both PCI and VLB exist as separate buses on the computer motherboard. Expansion cards exist for both these types of buses. You will usually find either PCI or VLB in addition to either ISA or EISA. Sometimes, however, you can also find *both* PCI and VLB in addition to the primary bus. In addition, it is possible to have machines that only have PCI because it is a true system bus and not an expansion bus like VLB. Because of the advantages of the PCI-Bus, some manufacturers are beginning to manufacture machines with only the PCI-Bus. However, as of this writing, only a few machines provide PCI-only expansion buses.

SCSI, on the other hand, complements the existing bus architecture by adding an additional hardware controller to the system. There are SCSI controllers (more commonly referred to as host adapters) that fit in ISA, EISA, MCA, PCI, or VLB slots.

8.2.1 Industry Standard Architecture ISA

As I mentioned before, most people are generally aware of the relationship between CPU performance and system performance. However, every system is only as strong as its weakest component. Therefore, the expansion bus also sets limits on the system performance.

There were several drawbacks with the original expansion bus in the original IBM PC. First, it was limited to only 8 data lines, which meant that only 8 bits could be transferred at a time. Second, the expansion bus was, in a way, directly connected to the CPU. Therefore, it operated at the same speed as the CPU, which meant that to improve performance with the CPU, the expansion bus had to be altered as well. The result would have been that existing expansion cards would be obsolete.

In the early days of PC computing, IBM was not known to want to cut its own throat. It has already developed quite a following with the IBM PC among users and developers. If it decided to change the design of the expansion bus, developers would have to re-invent the wheel and users would have to buy all new equipment. There was the risk that users and developers would switch to another platform instead of sticking with IBM.

Rather than risk that, IBM decided that backward compatibility was a paramount issue. One key change was severing the direct connection between the expansion bus and CPU. As a result, expansion boards could operate at a different speed than the CPU, enabling users to keep existing hardware and enabling manufacturers to keep producing their expansion cards. As a result, the IBM standard became the industry standard, and the bus architecture became known as the Industry Standard Architecture, or ISA.

In addition to this change, IBM added more address and data lines. They doubled the data lines to 16 and increased the address lines to 24. This meant that the system could address up to 16 megabytes of memory, the maximum that the 80286 CPU Intel's newest central processor at the time could handle.

When the 80386 came out, the connection between the CPU and bus clocks were severed completely because no expansion board could operate at the 16MHz or more that the 80386 could. The bus speed does not need to be an exact fraction of the CPU speed, but an attempt has been made to keep it there because by keeping the bus and CPU synchronized, it is easier to transfer data. The CPU will only accept data when it coincides with its own clock. If an attempt is made to speed the bus a little, the data must wait until the right moment in the CPUs clock cycle to pass the data. Therefore, nothing has been gained by making it faster.

One method used to speed up the transfer of data is Direct Memory Access, or DMA. Although DMA existed in the IBM XT, the ISA-Bus provided some extra lines. DMA enables the system to move data from place to place without the intervention of the CPU. In that way, data can be transferred from, lets say, the hard disk to memory while the CPU is working on something else. Keep in mind that to make the transfer, the DMA controller must have complete control of both the data and the address lines, so the CPU itself *cannot* access memory at this time. What DMA access looks like graphically we see in Figure 0-1.

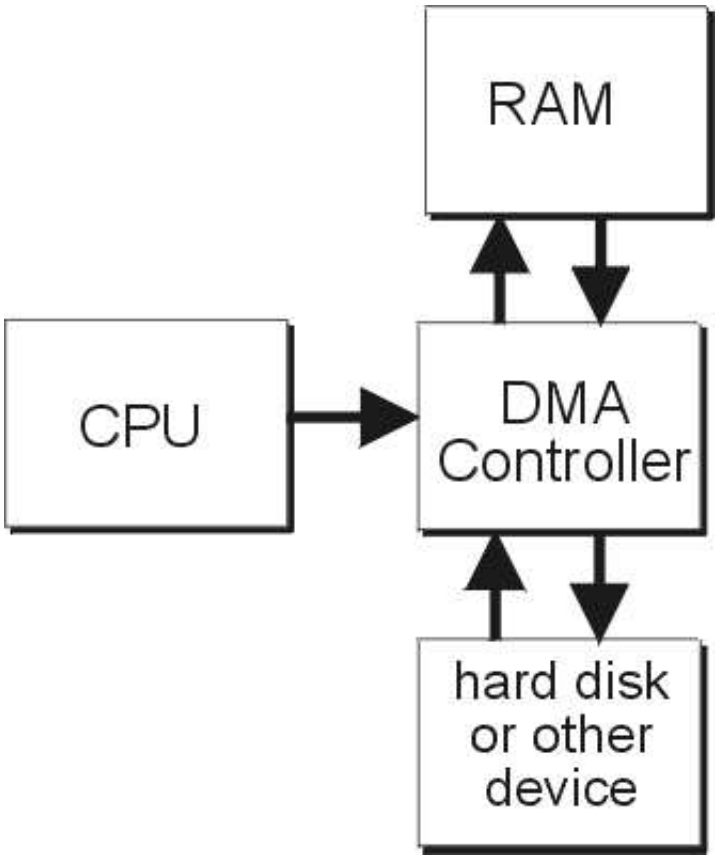


Image - Direct Memory Access

Lets step back here a minute. It is somewhat incorrect to say that a DMA transfer occurs without intervention from the CPU, as it is the CPU that must initiate the transfer. Once the transfer is started, however, the CPU is free to continue with other activities. DMA controllers on ISA-Bus machines use "pass-through" or "fly-by" transfers. That is, the data is not latched or held internally but rather is simply passed through the controller. If it were latched, two cycles would be needed: one to latch into the DMA controller and another to pass it to the device or memory depending on which way it was headed.

Devices tell the DMA controller that they wish to make DMA transfers through one of three "DMA Request" lines, numbered 13. Each of these lines is given a priority based on its number, 1 being the highest. The ISA-Bus includes two sets of DMA controllers: four 8-bit channels and four 16-bit channels. The channels are labeled 07, 0 having the highest priority.

Each device on the system capable of doing DMA transfers is given its own DMA channel. The channel is set on the expansion board usually by means of jumpers. The pins to which these jumpers are connected are usually labeled DRQ, for DMA Request.

The two DMA controllers both Intel 8237, each with four DMA channels, are cascaded together. The master DMA controller is the one connected directly to the CPU. One of its DMA channels is used to connect to the slave controller. Because of this, there are actually only seven channels available.

Everyone who has had a baby knows what an interrupt-driven operating system like Linux goes through on a regular basis. Just like a baby when it needs its diaper changed, when a device on the expansion bus needs servicing, it tells the system by generating an interrupt the baby cries. For example, when the hard disk has transferred the requested data to or from memory, it signals the CPU by means of an interrupt. When keys are pressed on the keyboard, the keyboard interface also generates an interrupt.

On receiving such an interrupt, the system executes a set of functions commonly referred to as an Interrupt Service Routine, or ISR. Because the reaction to a key being pressed on the keyboard is different from the reaction when data is transferred from the hard disk, there needs to be different ISRs for each device. Although the behavior of ISRs is different under DOS than UNIX, their functionality is basically the same. For details on how this works under Linux, see the chapter on the kernel.

On the CPU, there is a single interrupt request line. This does not mean that every device on the system is connected to the CPU via this single line, however. Just as a DMA controller handles DMA requests, an interrupt controller handles interrupt requests. This is the Intel 8259 Programmable Interrupt Controller, or PIC.

On the original IBM PC, there were five "Interrupt Request" lines, numbered 27. Here again, the higher the number, the lower the priority. Interrupts 0 and 1 are used internally and are not available for expansion cards.

The ISA-Bus also added an additional PIC, which is "cascaded" off the first PIC. With this addition, there were now 1615 interrupt values on the system 2x8-1 because the second is cascaded of the first. However, not all of these were available to devices. Interrupts 0 and 1 were still used internally, but so were interrupts 8 and 13. Interrupt 2 was something special.

It, too, was reserved for system use, but instead of being a device of some kind, an interrupt on line 2 actually meant that an interrupt was coming from the second PIC, similar to the way cascading works on the DMA controller.

A question I brought up when I first started learning about interrupts was "What happens when the system is servicing an interrupt and another one comes in?" Two mechanism can help in this situation .

Remember that the 8259 is a "programmable" interrupt controller. There is a machine instruction called Clear Interrupt Enable, or CLI. If a program is executing what is called a *critical section* of code a section that should not be stopped in the middle, the programmer can call the CLI instruction and disable acknowledgment of all incoming interrupts. As soon as the critical section is finished and closed, the program should execute a Set Interrupt Enable, or STI, instruction somewhat shortly afterward.

I say "should" because the programmer doesn't have to do this. A CLI instruction could be in the middle of a program somewhere and if the STI is never called, no more interrupts will be serviced. Nothing, aside from common sense, prevents the programmer from doing this. Should the program take too long before it calls the STI, interrupts could get lost. This is common on busy systems when characters from the keyboard "disappear."

The second mechanism is that the interrupts are priority based. The lower the interrupt request level, or IRQ, the higher the priority. This has an interesting side effect because the second PIC or slave is bridged off the first PIC or master at IRQ2. The interrupts on the first PIC are numbered 07, and on the second PIC the interrupts are numbered 8-15. However, the slave PIC is attached to the master at interrupt 2. Therefore, the actual priority is 0, 1, 8-15, 3-7.

Table 0-2 contains a list of the standard interrupts.

Table -2 Default Interrupts

IRQ	Device
0	System timer
1	Keyboard
2	Second level interrupt
3	COM 2
4	COM 1
5	Printer 2
6	Floppy
7	Printer 1
8	Clock
9	Not assigned
10	Not assigned
11	Not assigned
12	Not assigned
13	Math coprocessor
14	Hard Disk
15	Hard Disk

There's one thing you should consider when dealing with interrupts. On XT machines, IRQ 2 was a valid interrupt. Now on AT machines, IRQ 2 is bridged to the second PIC. So, to ensure that devices configured to IRQ 2 work properly, the IRQ 2 pin on the all the expansion slots are connected to the IRQ 9 input of the second PIC. In addition, all the devices attached to the second PIC are associated with an IRQ value where they are attached to the PIC, and they generate an IRQ 2 on the first PIC.

The PICs on an ISA machine are *edge-triggered*, which means that they react only when the interrupt signal is making the transition from low to high, that is, when it is on a transition *edge*. This becomes an issue when you attempt to share interrupts, that is, where two devices use the same interrupt.

Assume you have both a serial port and floppy controller at interrupt 6. If the serial port generates an interrupt, the system will "service" it. If the floppy controller generates an interrupt before the system has finished servicing the interrupt for the serial port, the interrupt from the floppy is lost. There is another way to react to interrupts called "level triggered," which I will get to shortly.

As I mentioned earlier, a primary consideration in the design of the AT Bus as the changed PC-Bus came to be called was that its maintain compatibility with it predecessors. It maintains compatibility with the PC expansion cards but takes advantage of 16-bit technology. To do this, connectors were not changed, only added. Therefore, you could slide cards designed for the 8-bit PC-Bus right into a 16-bit slot on the ISA-Bus, and no one would know the difference.

8.2.2 MCA

The introduction of IBM's Micro-Channel Architecture (MCA) was a redesign of the *entire* bus architecture. Although IBM developed the original AT architecture, which later became ISA, many companies produced machines that followed this standard. The introduction of MCA meant that IBM could produce machines to which it alone had the patent rights.

One of the most obvious differences is the smaller slots required for MCA cards. ISA cards are 4.75 x 13.5 inches, compared with the 3.5 x 11.5-inch MCA cards. As a result, the same number of cards can fit into a smaller area. The drawback was that ISA cards could not fit into MCA slots, and MCA cards could not fit into ISA slots. Although this might seem as though IBM had decided to cut its own throat, the changes they made in creating MCA made it very appealing.

Part of the decrease in size was a result of surface mount components, or surface mount technology (SMT). Previously, cards used "through-hole" mounting, in which holes were drilled through the system board (hence the name). Chips were mounted in these holes or into holders that were mounted in the holes. Surface mount does not use this and as a result, looks "flattened" by comparison. This saves not only space but also time and money, as SMT cards are easier to produce. In addition, the spacing between the pins on the card (0.050") corresponds to the spacing on the chips, which makes designing the boards much easier.

Micro-Channel also increases speed because there is a ground on every fourth pin, which reduces interference, and as a result, the MCA-Bus can operate at ten times the speed of non-MCA machines and still comply with FCC regulations in terms of radio frequency interference.

Another major improvement was the expansion of the data bus to 32 bits. This meant that machines were no longer limited to 16 megabytes of memory, but could now access 4 gigabytes.

One key change in the MCA architecture was the concept of *hardware-mediated bus arbitration*. With ISA machines, devices could share the bus, and the OS was required to arbitrate who got a turn. With MCA, that arbitration is done at the hardware level, freeing the OS to work on other things. This also enables multiple processors to use the bus. To

implement this, the bus has several new lines. Four lines determine the a*rbitration bus priority level*, which represents the 16 different priority levels that a device could have. Who gets the bus depends on the priority.

From the users perspective, the installation of MCA cards is much easier than that of ISA cards due to the introduction of the Programmable Option Select, or POS. With POS, the entire hardware configuration is stored in the CMOS. When new cards are added, you are required to run the machines *reference disk*. In addition, each card comes with an *options disk* that contains configuration information for the card. With the combination of reference disk and options disk, conflicts are all but eliminated.

Part of the MCA spec is that each card has its own unique identifying number encoded into the firmware. When the system boots, the settings in the CMOS are compared to the cards that are found on the bus. If one has been added or removed, the system requires you to boot using the reference disk to ensure that things are set up correctly.

As I mentioned, on each options disk is the necessary configuration information. This information is contained within the Adapter Description File (ADF). The ADF contains all the necessary information for your system to recognize the expansion card. Because it is only a few kilobytes big, many ADF files can be stored on a floppy. This is useful in situations like those we had in tech support. There were several MCA machines in the department with dozens of expansion cards, each with its own ADF file. Rather than having copies of each disk, the analysts who supported MCA machines (myself included) each had a single disk with all the ADF files. (Eventually that, too, became burdensome, so we copied the ADF files into a central directory where we could copy them as needed.) Any time we needed to add a new card to our machines for testing, we didn't need to worry about the ADF files because they were all in one place.

Because each device has its own identification number and this number is stored in the ADF, the reference diskette can find the appropriate number with no problem. All ADF files have names such as @BFDF.ADF, so it isn't obvious what kind of card the ADF file is for just by looking at the name. However, because the ADF files are simply text files, you can easily figure out which file is which by looking at the contents.

Unlike ISA machines, the MCA architecture enables *interrupt sharing*. Because many expansion boards are limited to a small range of interrupts, it is often difficult, if not impossible, to configure every combination on your system. Interrupt sharing is possible on MCA machines because they use something called *level-triggered interrupts,* or *level-sensitive interrupts*.

With e*dge-triggered interrupts,* or edge-sensitive interrupts, (the standard on ISA buses), an interrupt is generated and then is dropped. This sets a flag in the PIC, which figures out which device generated the interrupt and services it. If interrupts were shared with edge-triggered interrupts, any interrupt that arrived between the time the first interrupt is generated and serviced would be lost because the PIC has no means of knowing that a second interrupt occurred. All the PIC sees is that an interrupt occurred. Figure 0-2 shows how each of these elements relate to each other in time.

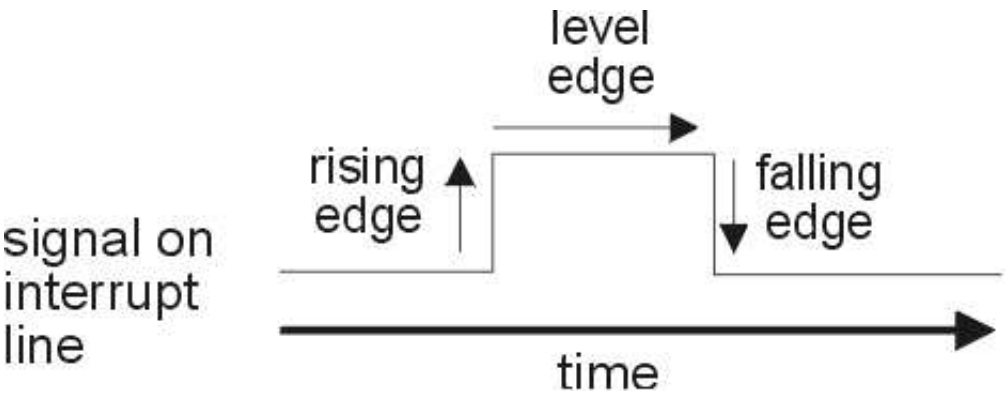


Figure - Interrupt Signal(Interactive)

With level-triggered interrupts, when an interrupt is generated, it is held high until the PIC forces it low after the interrupt has been serviced. If another device were on the same interrupt, the PIC would *try* to pull down the interrupt line; however, the second device would keep it high. The PIC would then see that it was high and would be able to service the second device.

Despite the many obvious advantages of the MCA, there are a few drawbacks. One primary drawback is the interchangeability of expansion cards between architectures. MCA cards can only fit in MCA machines. However, it is possible to use an ISA card in an EISA machine, and EISA machines is what I will talk about next.

8.2.3 Extended Industry Standard Architecture EISA

To break the hold that IBM had on the 32-bit bus market with the Micro-Channel Architecture, a consortium of computer companies, lead by Compaq, issued their own standard in September 1988. This new standard was an extension of the ISA-Bus architecture and was logically called the Extended Industry Standard Architecture EISA. EISA offered many of the same features as MCA but with a different approach.

Although EISA provides some major improvements, it has maintained backward compatibility with ISA boards. Therefore, existing ISA boards can be used in EISA machines. In some cases, such boards can even take advantage of the features that EISA offers.

To maintain this compatibility, EISA boards are the same size as their ISA counterparts and provide connections to the bus in the same locations. The original designe called for an extension of the bus slot, similar to the way the AT slots were an extension on the XT slots. However, this was deemed impractical because some hardware vendors had additional contacts that extended beyond the ends of the slots. There was also the issue that in most cases, the slots would extend the entire length of the motherboard, which meant that the motherboard would need to be either longer or wider to handle the longer slots.

Instead, the current spec calls for the additional connections to be intertwined with the old ones and extend lower. In what used to be gaps between the connectors are now leads to the new connectors. Therefore, EISA slots are deeper than those for ISA machines. Looking at EISA cards, you can easily tell them from ISA cards by the two rows of connectors.

Figure 0-3 shows what the ISA and EISA connections look like. Note that the adapters are not to scale.

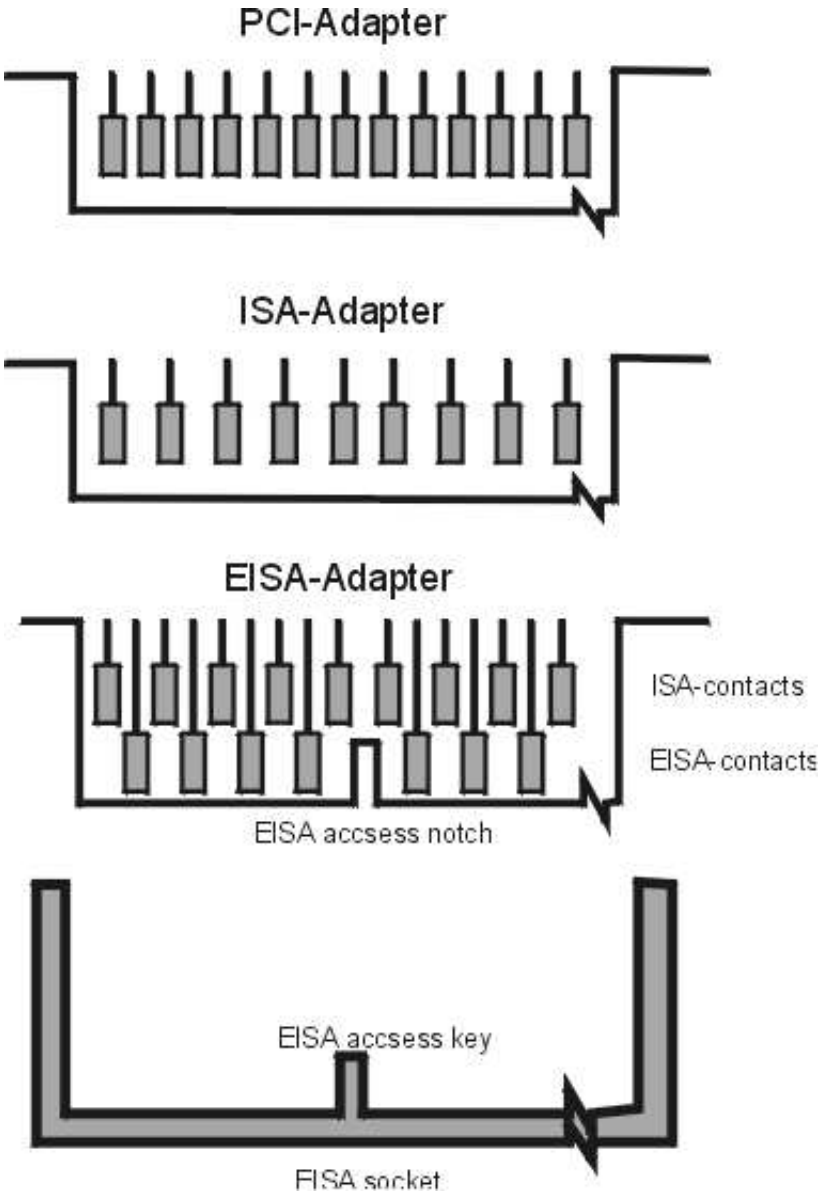


Image - Comparison of ISA, EISA and PCI, Connections

Another major improvement of EISA over ISA is the issue of bus arbitration. Bus arbitration is the process by which devices "discuss" whose turn it is on the bus and then let one of them go. In XT and AT class machines, the CPU completely managed control of the bus. EISA includes additional control hardware to take this job away from the CPU, which does two important things. First, the CPU is now "free" to carry on more important work, and second, the CPU gets to use the bus only when its turn comes around.

Hmmm. Does that sound right? Because the CPU is the single most important piece of hardware on the system, shouldn’t it get the bus whenever it needs it? Well, yes and no. The key issue of contention is the use of the word "single." EISA was designed with

multiprocessing in mind; that is, computers with more than one CPU. If there is more than one CPU, which *one*is more important?

The term used here is *bus arbitration*. Each of the six devices that EISA allows to take control of the bus has its own priority level. A device signals its desire for the bus by sending a signal to the Centralized Arbitration Control CAC unit. If conflicts arise e.g., multiple requests, the CAC unit resolves them according to the priority of the requesting devices. Certain activity such as DMA and memory refresh have the highest priority, with the CPU following close behind. Such devices are called "bus mastering devices" or "bus masters" because they become the master of the bus.

The EISA DMA controller was designed for devices that cannot take advantage of the bus mastering capabilities of EISA. The DMA controller supports ISA, with ISA timing and 24-bit addressing as the default mode. However, it can be configured by EISA devices to take full advantage of the 32-bit capabilities.

Another advantage that EISA has is the concept of dual buses. Because cache memory is considered a basic part of the EISA specification, the CPU can often continue working for some time even if it does not have access to the bus.

A major drawback of EISA as compared with MCA is that to maintain the compatibility to ISA, EISA speed improvements cannot extend into memory. This is because the ISA-Bus cannot handle the speed requirements of the high-speed CPUs. Therefore, EISA requires separate memory buses. This results in every manufacturer having its own memory expansion cards.

In the discussion on ISA, I talked about the problems with sharing level-triggered interrupts. MCA, on the other hand, uses edge-triggered interrupts, which enables interrupt sharing. EISA uses a combination of the two. Obviously, EISA needs to support edge-triggered interrupts to maintain compatibility with ISA cards. However, it enables EISA boards to configure that particular interrupt as either edge- or level-triggered.

As with MCA, EISA enables each board to be identified at boot up. Each manufacturer is assigned a prefix code to make the identification of the board easier. EISA also provides a configuration utility similar to the MCA reference disk to enable configuration of the cards. In addition, EISA supports automatic configuration, which enables the system to recognize the hardware at boot-up and configure itself accordingly. This can present problems for a Linux system because drivers in the kernel rely on the configuration to remain constant. Because each slot on an EISA machine is given a particular range of base addresses, it is necessary to modify your kernel before making such changes. This is often referred to as the EISA-config, EISA Configuration Utility, or ECU.

8.2.4 The Small Computer Systems Interface SCSI

The SCSI-Bus is an extension of your existing bus. A controller card, called a host adapter, is placed into one of your expansion slots. A ribbon cable that contains both data and control signals then connects the host adapter to your peripheral devices.

There are several advantages to having SCSI in your system. If you have a limited number of bus slots, adding a single SCSI host adapter enables you to add up to seven more devices by taking up only one slot with older SCSI systems and up to 15 devices with Wide-SCSI. SCSI has higher throughput than either IDE or ESDI. SCSI also supports many more different types of devices.

There are several different types of SCSI devices. The original SCSI specification is commonly referred to as SCSI-1. The newer specification, SCSI-2, offers increased speed and performance, as well as new commands. Fast SCSI increases throughput to more than 10MB per second. Fast-Wide SCSI provides a wider data path and throughput of up to 40MB per second and up to 15 devices. There there are Ultra-SCSI and Ultra-Wide-SCSI

The last type, SCSI-3, is still being developed as of this writing and will provide the same functionality as Fast-Wide SCSI as well as support longer cables and more devices.

Each SCSI device has its own controller and can send, receive, and execute SCSI commands. As long as it communicates with the host adapter using proper SCSI commands, internal data manipulation is not an issue. In fact, most SCSI hard disks have an IDE controller with a SCSI interface built onto them.

Because there is a standard set of SCSI commands, new and different kinds of devices can be added to the SCSI family with little trouble. However, IDE and ESDI are limited to disk-type devices. Because the SCSI commands need to be "translated" by the device, there is a slight overhead, which is compensated for by the fact that SCSI devices are intrinsically faster than non-SCSI devices. SCSI devices also have higher data integrity than non-SCSI devices. The SCSI cable consists of 50 pins, half of which are ground. Because every pin has its own ground, it is less prone to interference and therefore it has higher data integrity.

On each SCSI host adapter are two connectors. One connector is at the top of the card opposite the bus connectors and is used for internal devices. A flat ribbon cable is used to connect each device to the host adapter. On internal SCSI devices, only one connector is on the device itself. Should you have external SCSI devices, there is a second connector on the end of the card where it attaches to the chassis. Here SCSI devices are "daisy chained" together.

The SCSI-Bus must be *closed* to work correctly. By this I mean that each end of the bus must be terminated. There is usually a set of resistors or slots for resistors on each device. The devices at either end of the SCSI-Bus must have such resistors. This process is referred to as terminating the bus and the resistors are called terminating resistors.

Its fine to say that the SCSI-Bus needs to be terminated. However, that doesn’t help your understanding of the issue. As with other kinds of devices, SCSI devices react to the commands sent along the cable to them. Unless otherwise impeded, the signals reach the end of the cable and bounce back. In such cases, there are two outcomes, both of which are undesirable: either the bounced signal interferes with the valid one, or the devices react to a second unique in its mind command. By placing a terminator at the end of the bus, the signals are "absorbed" and, therefore, don’t bounce back.

The following two figures show examples of how the SCSI-Bus should be terminated. Note that Figure 0-6 says that it is an example of "all external devices." Keep in mind that the principle is still the same for internal devices. If all the devices are internal, then the host adapter would be still be terminated, as would the last device in the chain.

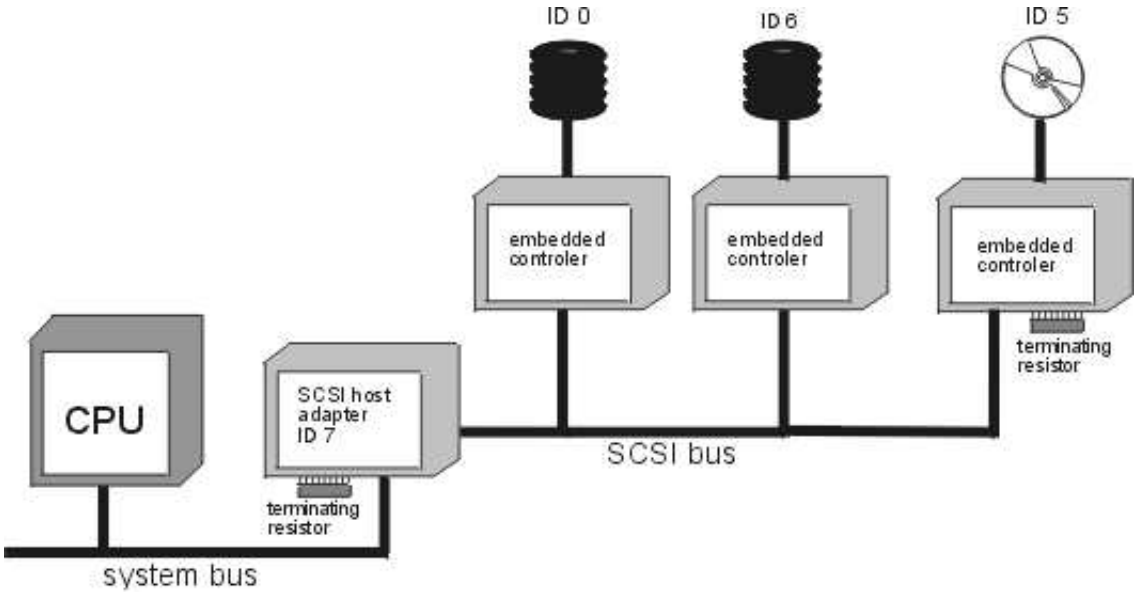
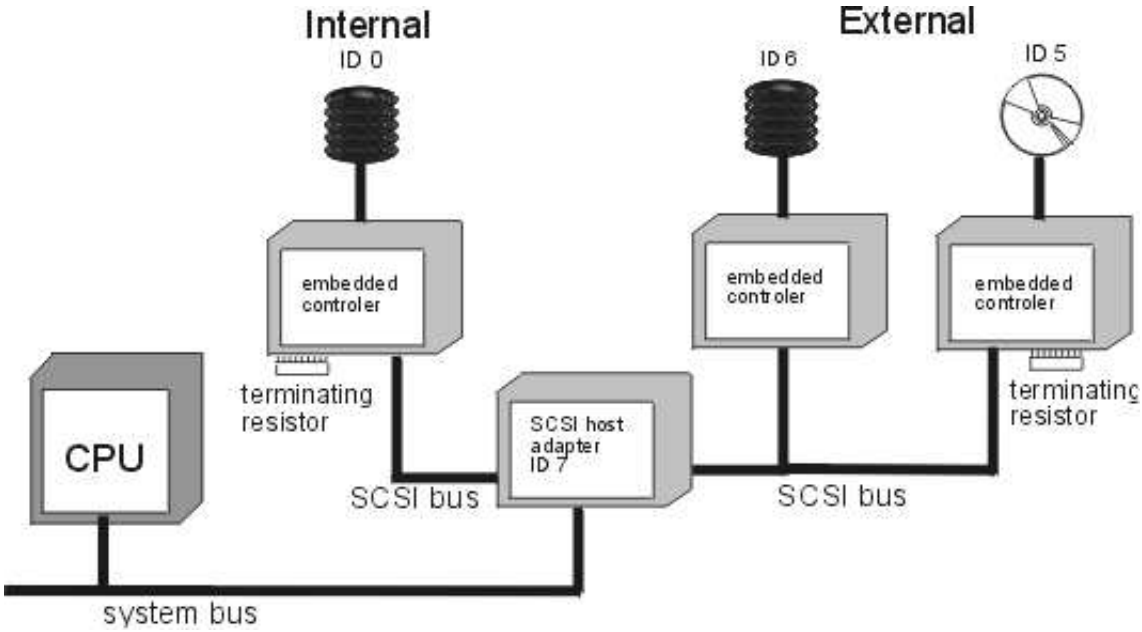


Image - Example of SCSI Bus with All External Devices



Example of SCSI Bus with Both External and Internal Devices

If you don’t have any external devices or only have external devices, the host adapter is at one end of the bus. Therefore, it too must be terminated. Many host adapters today have the ability to be terminated in software, so there is no need for terminating resistors also known as resistor packs.

Each SCSI device is "identified" by a unique pair of addresses, which are the controller addresses that are also referred to as the SCSI ID. This pair of addresses is usually set by jumpers or dip switches on the device itself. Keep in mind that the ID is something that is set on the device itself and is *not* related to location on the bus. Note that in the figures above, the SCSI ID of the devices are ordered ID 0, 6, and 5. Also the SCSI ID is often set using a combination of jumpers with no 1:1 relationship. That is a pair of pins labeled ID 0 through ID 7 Therefore, you should always read the hardware documentation to determine how to set the ID.

This sounds pretty obvious, but some people don't make sure. They make assumptions about what they see on the device regarding how the ID is set and do not fully understand what it means. For example, I have an Archive 5150 SCSI tape drive. On the back are three jumpers, labeled 0, 1, and 2. I have had customers call in with similar hardware with their SCSI tape drive set at 2. After configuring the tape drive and rebooting, they still couldn't access the tape drive. Nothing else was set at ID 2, so there were no conflicts. The system could access other devices on the SCSI-Bus, so the host adapter was probably okay. Different SCSI devices can be plugged into the same spot on the SCSI cable, so it wasn't the cable. The SCSI-Bus was terminated correctly, so that wasn't the problem.

Rather than simply giving up and saying that it was a hardware problem, I suggested that the customer change the SCSI ID to 3 or 4 to see if that worked. Well, the customer couldn't, because the jumpers on the back only allowed him to change the SCSI ID to 0, 1, or 2. It then dawned on me what the problem was: the jumpers in the back are in binary! To set the ID to 2, the jumper needs to be on jumper 1, *not* jumper 2. Once the customer switched it to jumper 1 and rebooted, all was well. Note: I helped this customer *before* I bought the Archive tape drive. When I got my drive home and wanted to check the SCSI ID, I saw only three jumpers. I then did something that would appall most users: I read the manual! Sure enough, it explained that the jumpers for the SCSI ID were binary.

An additional problem to this whole SCSI ID business is that manufacturers are not consistent among each other. Some might label the jumpers or switches 0, 1, and 2. Others label them 1, 2, and 4. Still others label them ID0, ID1, and ID2. I have even seen some with a dial on them with 8 settings, which makes configuration a lot easier. The key is that no matter how they are labeled, the three pins or switches are binary and their values are added to give you the SCSI ID.

Lets look at Figure 0-8, which represents the jumper settings on a SCSI device. In the first example, none of the jumpers is set, so the SCSI ID is 0. In the second example, the jumper labeled 1 is set. This is 2^1 or 2, so the ID here is 2. In the last example, the jumpers labeled 2 and 0 are set, which is $2^2 + 2^0 = 4 + 1$, or 5.

On an AT-Bus, the number of devices added is limited only by the number of slots granted, the AT-Bus is limited in how far away the slot can be from the CPU and therefore is limited in the number of slots. On a SCSI-Bus, however, there can be only seven devices in addition to the host adapter. Whereas devices on the AT-Bus are distinguished by their base addresses, devices on the SCSI-Bus are distinguished by their ID number.

ID numbers range from 07 and, unlike base addresses, the higher the ID, the higher the priority. Therefore, the ID of the host adapter should always be a 7. Because it manages all the other devices, it should have the highest priority. On the newer Wide SCSI-Buses, there can be up to 15 devices, plus the host adapter, with SCSI IDs ranging from 0 to 15.

Now back to our story...

The device address is known as the logical unit number LUN. On devices with embedded controllers, such has hard disks, the LUN is always 0. All the SCSI devices directly supported by Linux have embedded controllers. Therefore, you are not likely to see devices set at LUNs other than 0.

In theory, a single-channel SCSI host adapter can support 56 devices. Devices called bridge adapters connect devices without embedded controllers to the SCSI-Bus. Devices attached to the bridge adapter have LUNs between 0 and 7. If there are seven bridge adapters, each with eight LUNs relating to eight devices, 56 total devices are therefore possible.

The original SCSI-1 spec only defined the connection to hard disks. The SCSI-2 spec has extended this connection to such devices as CD-ROMS, tape drives, scanners, and printers. Provided these devices all adhere to the SCSI-2 standard, they can be mixed and matched even with older SCSI-1 hard disks.

One common problem with external SCSI devices is that the power supply is external as well. If you are booting your system with the power to that external device turned off, once the kernel gets past the initialization routines for that device the hardware screen, it can no longer recognize that device. The only solution is to reboot. To prevent this problem, it is a good idea to keep all your SCSI devices internally. This doesn't help for scanners and printer, but because Linux doesn't yet have drivers for them, it's a moot point.

Although the number of host adapter manufacturers has steadily decreased in the past few years, Adaptec, the premier name in host adapters, has bought up both Trantor and Future Domain. Adaptecs biggest competitor for years, Buslogic, was no longer able to compete and was taken over by Mylex a motherboard manufacturer, among other things. Despite the decrease in number of manufacturers, the number of models is still overwhelming.

Most host adapter manufacturers provide more than just a single model. Many provide models for the entire spectrum of buses and SCSI types. ISA, EISA, PCI, Fast SCSI, Wide SCSI, and Ultra-Wide SCSI are part of the alphabet soup of SCSI devices. You can connect Wide SCSI disks onto a Fast SCSI adapter, although it will still only get 8 bits instead of the Wide SCSIs 16 bits, so it therefore only gets 10Mbytes per second compared to 20Mbytes per second of Wide SCSI.

Ultra SCSI disks can also be connected with the same limitations it is an 8-bit bus. It can also handle Ultra-Wide SCSI and get 40Mbps. This is not too big of an issue, as most of the devices available today can only handle 10Mbps.

When looking at the performance of a SCSI device, you need to be careful of the manufacturers test results. They can be deceiving. If a test reads 200MB from the disk in 10 seconds, you get an average of 20MB per second. What if those 100MB are all from the same

track? The disk hardware reads the track and keeps it in its own cache. When the host adapter requests a new block from that track, the hard disk doesn't need to find the block on the disk,it delivers it from the cache. This decreases the access time and increases the *apparent* transfer rate of the drive dramatically. The manufacturer can say, in all honesty, that the host adapter has a transfer rate of 20Mbps, though the drive can only do half of this at most. Again, the chain is only as strong as its weakest link.

This does not mean that Wide SCSI or Ultra SCSI are only useful for the companies marketing departments. SCSI has the advantage of being able to talk to multiple devices. For example, it can request data from one drive and, rather than waiting for the data, free the SCSI-Bus disconnect. When the drive or other device is ready, it requests the bus again reconnect and the data is transferred. While the drive searches for the data, the host adapter can request data from another device. While this device is looking for the data, the first device can transfer the data to the host adapter. Being able to read or write devices like this means that a host adapter could get a *sustained* transfer rate of more that what *individual* devices can handle. Note that both the host adapter and device must support disconnect/reconnect.

Wide SCSI gets its performance gain by the fact it is wide 16 bits versus 8 bits. Ultra SCSI, on the other hand, gets the increase through a shorter cycle time. This is an important aspect because this makes for a steeper edge on the signal the time from a low to high signal is much shorter, and vice versa. This means that the SCSI-Bus has higher requirements regarding the cabling.

Internal devices usually are connected by flat cable ribbons and present few new problems with Fast SCSI. The maximum length of the cable is half of what it could be with older SCSI devices and you must follow the specs exactly. Round cables for external devices have to be created specifically for Ultra SCSI and are therefore more expensive. Although the actual data transfer rate between the host adapter and the device is only as high as the device can handle, the steepness of the edges is the same. This means that if you connect Fast SCSI devices to Ultra SCSI host adapters, you still need the special Ultra SCSI cables.

Another consideration is that Ultra SCSI requires *active* termination. On the host adapter side, this isn't a problem because the host adapters are designed to give active termination. However, many older devices support only passive termination and therefore cant work on Ultra SCSI host adapters. This really comes into play when larger amounts of data are being transferred.

PCI devices can generally behave as either masters or slaves. For slave devices, the CPU is responsible for all the activity. This is a disadvantage for slave devices because the CPU is often busy transferring data and issuing commands instead of doing other work. This is really an issue in multitasking operating systems like Linux that have "better" things to do. Master devices, on the other hand, have an advantage here. The CPU only needs to tell them where to transfer the data, and they do the work themselves.

Regardless of whether a device acts as a master or slave, it will take up an interrupt line. Single function devices, such as host adapters, are given the interrupt INT-A. This means that the actual IRQ between 5 and 15 will be determined by the system BIOS.

Generally, you can say that a higher throughput is required on a file server as compared to a workstation. Although there are applications like CAD or video processing, which require more throughput on a workstation than other kinds of applications, the major of the work is done by the server. Therefore, it is extremely important to consider the performance of your hard disk and similar devices on the server.

Despite reaching comparable prices and sizes, ATA harddisks as suited for work in a server because they do not have the throughput of SCSI. As we discussed previously, SCSI has the advantage of being able to have two devices communicate with each other directly with the need to go through the CPU. In addition, while waiting for one device to find the data, it can "disconnect" itself and you can make a request of another device on the same SCSI bus. This is especially important on servers as they usually have multiple disks, as well as other kinds of devices.

Here again, the chain to your harddisk is only as weak as the weakest link. Therefore, you need to have a SCSI host adapter that can keep up with the hard disk.

Let’s take my Adaptec 2940U2W host adapter which I have in my primary workstation as an example. This is an Ultra2 SCSI device, which gives me a maximum transfer rate of 80 Mbyte/second. One neat aspect of this host adapter, is that it uses a technology Adaptec calls "SpeedFlex.". Internally, there are three connectors. One for Ultra2 SCSI 80Mb/s, one for Wide Ultra SCSI 40Mb/s and one for Ultra SCSI 20Mb/s. Externally, you have two connectors. One 68-pin for Ultra2 SCSI and one 50-pin for Ultra SCSI. Therefore, you get the maximum throughput no matter what kind of device is connect to the host adapter. In addition, one aspect of the SpeedFlex technology is that it can operate the different buses at the different speeds simultaneously, so there no performance lost on one bus because of a slow device on another bus.

The "W" at the end of the host adapter name means that it has the Wide Ultra SCSI connector. Adaptec also produces the 2940U2, which has the same specifications, but without the internal Wide Ultra SCSI connector and without the external Ultra2 connector. Note that in each case, devices supporting older SCSI standards can still be connected.

In one one of my machines, I have an Adaptec 3940U2, which is also an Ultra2 SCSI adapter. One key difference is that this adapter is twin-channel. That means I have two SCSI buses running off of the same host adapter. This is extremely useful when you have more device that will fit on a single bus, plus added speed. Both of this aspects make this the perfect adapter for a Linux server although I use it in my Workstation. Another important difference is that the Adaptec 3940U2 supports 64-bit PCI, although it is still capable with 32-bit PCI.

One thing to note is that all of the devices support up to 15 devices plus the host adapter per channel. This means that the twin-channel 3940U2 can connect up to 30 devices. In addition, all devices support cables up to 12 meters long.

On one machine I have an Adaptec 1542 CF host adapter. The Adaptec 1540 a SCSI-1 and SCSI-2 compliant ISA host adapter, which has an 8-bit data bus and a maximum transfer rate of 10 Mbyte/sec. This is a perfect host adapter for workstations which require less performance.

Initially, I wanted to say "low end", but that tends to create false impressions of something of lesser quality. This definitely does not apply to the Adaptec 1540 family or any of the Adaptec products for that matter. A few months ago, I replaced the 1542CF I had after six years because the built-in floppy controller was no longer working correctly. The host adapter was working fine. I just couldn't access my floppy drive.

A few weeks later, I put it back into the machine as it as the only one with the external connector my CD-ROM changer had. Rather than buying an adapter for the connector, I put in my old Adaptec 1542 CF and it has run perfectly ever since.

All of the Adaptec host adapters support what is called "scatter gather." Here, requests for data on the harddisk which are "scattered" all over the drive are "gathered" together, in order that they be more efficiently process.

This is similar to way an elevator works. Image that four people get into an elevator. The first one presses the button for floor 12, the next one for floor 16 the next one for floor 3, and the last one wants floor 8. Although the person wanting to go to floor 12 was there first, the elevator stops at floor 3 first, then floor 8 before continuing to floor 12. This is much more efficient than going to each floor in the order the buttons were pressed.

Accessing a hard disk is similar. On active systems, there will be several requests waiting to processed. Adaptec host adapters will sort the requests based on their physical location on the hard disk. Interestingly enough, Adaptec refers to this as an "elevator sort."

8.2.5 Termination

With Ultra SCSI, termination plays a more important role. A steeper edge means that the reflection has a stronger effect than with Fast SCSI. Moreover, a faster cycle means that the bus is more sensitive to interference. In principle, SCSI termination, even with Ultra SCSI, is simple: both ends of the bus (that is, the *physical* ends of the bus) must be terminated.

If you have fewer devices than connectors on your SCSI cable, I advise you to connect devices at both ends of the cable, terminating both of them. Loose ends can definitely lead to problems with reflection. By having devices at the physical ends of the cable, there is no question which device is at the end of the bus. Keep in mind that the order of the devices on the bus is independent of this.

You run into problems when the device has no possibility of being terminated or functions only with passive termination. Although no termination is rarely found, many (especially older) devices support only passive termination. Such devices include a lot of CD-ROMS and tape drives. Read the hardware documentation to find out what type of termination your drive supports or contact the manufacturer before you purchase the drive.

You need to be careful with some hard disks. There is often a jumper labeled TERM, which does not enable/disable the termination, but rather enables/disables the power for the active termination.

If you do have a device with active termination, this device belongs at one end of the SCSI cable. The other end is usually the host adapter. PCI host adapters are almost exclusively produced with active termination.

If both external and internal devices are present, the host adapter must not be terminated because it is now in the middle of the bus and no longer at the end. The termination is now on the device at the end of the other cable. Note that older, 50-pin Centronics connectors are almost exclusively passive terminators. Therefore, if you replace your existing host adapter with an Ultra-SCSI adaptor, you really should change the termination to active.

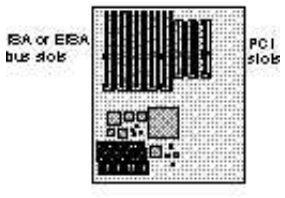
Wide SCSI presents its own termination and cabling problems. On most Wide-SCSI host adapters, you'll find an 8-bit and 16-bit connector, both of which you can use. However, keep in mind that *both* must be terminated.

8.2.6 PCI

More and more machines you find on the market today are being included with PCI local buses. One advantage that PCI offers over VL-Bus is the higher performance, automatic configuration of peripheral cards, and superior compatibility. A major drawback with the other bus types (ISA, EISA, MCA) is the I/O bottleneck. Local buses overcome this by accessing memory using the same signals lines as the CPU. As a result, they can operate at the full speed of the CPU as well as utilizing the 32-bit data path. Therefore, I/O performance is limited by the card and not the bus.

Although PCI is referred to as a local bus, it actually lies somewhere "above" the system bus. As a result it is often referred to as a "mezzanine bus" and has electronic "bridges" between the system bus and the expansion bus. As a result, the PCI bus can support up to 5 PCI devices, whereas the VL-BUS can only support two or three. In addition, the PCI bus can reach transfer speeds four times that of EISA or MCA.

Despite PCI being called a mezzanine bus, it could replace either ISA, EISA or MCA buses. Although in most cases, PCI is offered as a supplement to the existing bus type. If you look at a motherboard with PCI slots, you will see that they are completely separate from the other



slots. Whereas VLB slots are extensions of the existing slots.
Figure Comparison of ISA/EISA bus slots to PCI

PCI offers additional advantages over the VLB as the VLB cannot keep up with the speed of the faster CPUs, especially if there are multiple VLB devices on the system. Because PCI works together with the CPU it is much more suited to multi-tasking operating systems like UNIX. Whereas the CPU cannot work independently if a VLB device is running.

Like EISA and MCA, PCI boards have configuration information built into the card. As the computer is booting, the system can configure each card individually based on system resources. This configuration is done "around" existing ISA, EISA and MCA cards on your system.

To overcome a shortcoming PCI has when transferring data, Intel (designer and chief proponent of PCI) has come up with a PCI specific chip sets, which allows data to be stored on the PCI controller, freeing the CPU to do other work. Although this may delay the start of the transfer, however once the data flow starts, it should continue uninterrupted.

A shortcoming of PCI, is that ISA and EISA cards can be swapped for VLB cards, without any major problems. This is not so for the PCI cards. Significant changes need to be made to both the kernel and device drivers to account for the differences.

When it first came out, a shortcoming of PCI was the number of boards that are available. The number grew rapidly, and there are now more PCI boards than ISA boards available. It is on its way to becoming a de facto standard if not de jure.

On the other hand, the PCI bus is not processor dependent. This allows PCI to be installed in Pentium machines, as well as Alpha machines and other architectures. You can therefore run the same SCSI host adapter (or whatever). Since it is 64-bits wide, both the Pentium and Alphas are not slowed down (too much) by the bus.

8.2.7 AGP

One of latest entrants to the alphabet soup of bus types is the Accelerated Graphics Port (AGP). As its name implies, AGP is intended as a bus for graphics cards. Although PCI is more than sufficient for most users needs, there are some shortcoming with high-end applications. The need for something like AGP arises from problems when creating 3D graphics, which use a "texture map" to get the necessary effects. The problem lies in the fact that most video cards deal with at most 8MB, but texture maps are reaching 10 or even 20 MB (probably more by the time you read this.)

Added to this memory limitation is the bandwidth required to transfer that amount of data quickly. To get real-time 3D graphics, you need far better throughput than loading a document into WordPerfect. Added to this fact is that there is just s single PCI bus, which makes it even more of a bottleneck.

In essence the AGP is high-speed path between system memory and the video controller. It is no longer necessary to store things like the texture maps in the system memory rather than in the limited space of the video card. Because data transfer across the AGP bus can be up to fourth times that on the PCI bus, AGP provides even better performance.

8.3 Memory

Memory is the part of the computer where your program and data are while they are being used by the CPU. Contrast this to a hard disk or floppy, where the program is sitting on the disk and is not being used. (Of course with operating systems like Linux, parts of both the

program and the data can be stored on the disk, even as the program is running.) There are two types of memory that are most common talked about: RAM and cache.

In most memory today, an extra bit is added for each byte. This is a parity bit. Parity is a simple way of *detecting* errors within a memory chip (among other things). If an odd number of bits is set, the parity bit will be set to make the total number of bits set an even number (most memory uses even parity). For example, if three bits are set, the parity bit will also be set to make the total bits set four.

When data is written, the number of set bits is calculated and the parity bit is set accordingly. When the data is read, the parity bit is also read. If the total number of bits set is even, all is well. However, if an odd number of data bits is set and the parity bit is not set, or if an even number of data bits is set and the parity bit is set, a parity error has occurred.

When a parity error occurs in memory, the state of the system is uncertain. To prevent any further problems, the parity checking logic generates a Non-maskable Interrupt (NMI), and the CPU immediately jumps to special codes called NMI service routines.

When Linux is interrupted with an NMI as the result of a parity error, it too realizes things are not good, and the system panics. The panic causes the system to stop everything and shut down. Certain machines support ECC (Error Correcting Code) RAM, which corrects parity problems before it kills your system.

8.3.1 RAM

A computer stores the data it works with in three ways, often referred to as memory. Long-term memory, which remains in the system even when there is no power, is called non-volatile memory and exists in such places as hard disks or floppies, which are often referred to as secondary storage. Short-term memory, or volatile memory, is stored in memory chips called RAM (random-access memory). RAM is often referred to as primary storage. The third class of memory is often ignored, or at least not often thought of. This type of memory exists in hardware on the system but does *not* disappear when power is turned off. This is called ROM, or read-only memory.

Typically ROM is set by the motherboard manufacturer and contains the most essential information to start your computer and provide very basic access to your hardware like your floppy or IDE hard disk (This is your system BIOS) . Normally there is no need to change it (therefore, "read-only) but when problems are discovered the manufacturer might provide you a disk with a ROM update, which actually rewrites portions of your ROM .

I need to clarify one thing before we go on. Read-only memory is, as it says, read-only. For the most part, you cannot write to it. However, like random-access memory, the locations within it can be accessed in a "random" order, that is, at the discretion of the programmer. Also, read-only memory isn’t always read-only, but that’s a different story that goes beyond the scope of this book.

The best way to refer to memory to keep things clear (at least the best way in my opinion) is to refer to the memory we traditional call RAM as "main" memory. This is where our programs and the operating system actually reside.

There are two broad classes of memory: Dynamic RAM, or DRAM (pronounced dee-ram), and Static RAM, or SRAM (pronounced es-ram). DRAM is composed of tiny capacitors that can hold their charge only a short while before they require a "boost." SRAM is static because it does not require an extra power supply to keep its charge. As a result of the way it works internally, SRAM is faster and more expensive than DRAM. Because of the cost, the RAM that composes main memory is typically DRAM.

DRAM chips hold memory in ranges of 64KB to 16MB and more. In older systems, individual DRAM chips were laid out in parallel rows called banks. The chips themselves were called DIPPs, for Dual In-Line Pin Package. These look like the average, run-of-the-mill computer chip, with two rows of parallel pins, one row on each side of the chip. If memory ever went bad in one of these banks, it was usually necessary to replace (or test) dozens of individual chips. Because the maximum for most of these chips was 256 kilobits (32KB), it took 32 of them for each megabyte!

On newer systems, the DIPP chips have been replaced by Single In-Line Memory Modules, or SIMMs. Technological advances have decreased the size considerably. Whereas a few years ago, you needed an area the size of standard piece of binder paper to hold just a few megabytes, todays SIMMs can squeeze twice as much into an area the size of a stick of gum.

SIMMs come in powers of 2 megabytes (1, 2, 4, 8, etc.,) and are generally arranged in banks of four or eight. Because of the way the memory is accessed, you sometimes cannot mix sizes. That is, if you have four 2Mb SIMMs, you cannot simply add an 8Mb SIMM to get 16Mb. Bear this in mind when you order your system or order more memory. You should first check the documentation that came with the motherboard or the manufacturer.

Many hardware salespeople are not aware of this distinction. Therefore, if you order a system with 8MB that’s "expandable" to 128Mb, you may be in for a big surprise. True, there are eight slots that can contain 16Mb each. However, if the vendor fills all eight slots with 1Mb SIMMs to give you your 8MB, you may have to throw *everything* out if you ever want to increase your RAM.

However, this is not always the case. My motherboard has some strange configurations. The memory slots on my motherboard consist of two banks of four slots each (which is typical of many machines). Originally, I had one bank completely full with four 4Mb SIMMs. When I installed Open Server, this was barely enough. When I decided to start X-Windows and Wabi, this was much too little. I could have increased this by 1Mb by filling the first bank with four 256K SIMMs and moving the four 4Mb SIMMs to the second bank. However, if I wanted to move up to 20Mb, I could use 1Mb instead of 256K. So, here is one example where everything does *not* have to match. In the end, I added four 4MB SIMMs to bring my total up to 32MB. The moral of the story: Read the manual!

Another issue that you should consider with SIMMs is that the motherboard design may require you to put in memory in multiples of either two or four because this is the way the motherboard accesses that memory. Potentially, a 32-bit machine could read a byte from four SIMMs at once, essentially reading the full 32 bits in one read. Keep in mind that the 32 bits are probably not being read simultaneously. However, being able to read them in succession is faster that reading one bank and then waiting for it to reset.

Even so, this requires special circuitry for each of the slots, called address decode logic. The address decode logic receives a memory address from the CPU and determines which SIMM it's in and where it is on the SIMM. In other words, it decodes the address to determine which SIMM is needed for a particular physical address.

This extra circuitry makes the machine more expensive because this is not just an issue with the memory but with the motherboard design as well. Accessing memory in this fashion is called "page mode" because the memory is broken into sets of bytes, or pages. Because the address decode logic is designed to access memory in only one way, the memory that is installed must fit the way it is read. For example, my motherboard requires each bank to be either completely filled or completely empty. Now, this requires a little bit of explanation.

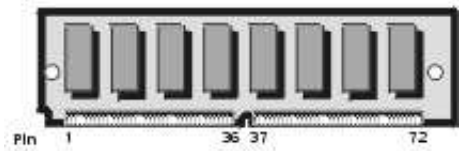
As I mentioned earlier, DRAM consists of little capacitors for each bit of information. If the capacitor is charged, then the bit is 1; if there is no charge, the bit is 0. Capacitors have a tendency to drain over time, and for capacitors this small, that time is *very* short. Therefore, they must be regularly (or dynamically) recharged.

When a memory location is read, there must be some way of determining whether there is a charge in the capacitor. The only way to do that is to discharge the capacitor. If the capacitor can be discharged, that means that there was a charge to begin with and the system knows the bit was 1. Once discharged, internal circuitry recharges the capacitor.

Now, assume that the system wanted to read two consecutive bytes from a single SIMM. Because there is no practical way for the address decode logic to tell that the second read is not just a re-read of the first byte, the system must wait until the first byte has recharged itself. Only then can the second byte be read.

By taking advantage of the fact that programs run sequential and rarely read the same byte more than once at any given time, the memory subsystem can interleave its reads. That is, while the first bank is recharging, it can be reading from the second, and while the second is recharging, it can be reading from the third, and so on. Because subsequent reads must wait until the previous read has completed, this method is obviously not as fast as simultaneous reads. This is referred to as "interleaved" or "banked" memory.

Standard 30-pin SIMM



72-pin PS/2 SIMM

Figure -9 Comparison of 30-pin and 72-pin SIMMs

Because all of these issues are motherboard-dependent, it best to check the hardware documentation when you change or add memory. Additionally, you may need to adjust settings, or jumpers, on the motherboard to tell it how much RAM you have and in what configuration.

Another issue that addresses speed is the physical layout of the SIMM. SIMMs are often described as being arranged in a "by-9" or "by-36" configuration, which refers to the n umber of bits that are immediately accessible. So, in a "by-9" configuration, 9 bits are immediately accessible, with 1 bit used or parity. In a "by-36" configuration, 36 bits are available with 4 bits for parity (one for each 8 bits). The "by-9" configuration comes on SIMMs with 30 pins, where the "by-36" configuration comes on SIMMs with 72 pins. The 72-pin SIMMs can read 32 bits *simultaneously*, so they are even faster than 30-pin SIMMs at the same speed. Figure 0-9 shows give you a comparison of the older SIMMs and PS/2 SIMMs.

There are also different physical sizes for the SIMM. The 30-pin SIMMs are slightly smaller than 72-pin SIMMs. The larger, 72-pin variety are called PS/2 SIMMs because they are used in IBMs PS/2 machines. As well as being slightly larger, the PS/2 SIMM has a notch in the center so it is impossible to mix up the two. In both cases, there is a notch on one end that fits into a key in the slot on the motherboard, which makes putting the SIMM in backward almost impossible.

SIMMs come in several different speeds. The most common today are between 60 and 80 nanoseconds. Although there is usually no harm in mixing speeds, there is little to be gained. However, I want to emphasize the word *usually*. Mixing speeds has been known to cause panics. Therefore, if you mix speeds, it is best keep all the SIMMS within a single bank at a single speed. If your machine does not have multiple banks, then it is best not to mix speeds. Even if you do, remember that the system is only as fast as its slowest component.

Recently, the computer industry has begun to shift away from the old SIMMs toward extended data out RAM or EDORAM. Although as of this writing, EDORAM is still more expensive than SIMM, it is expected that by early 1997, the demand for EDORAM will be such that the price difference will disappear.

The principle behind EDORAM is an extension of the fast-page-mode (FPM) RAM. With FPM RAM, you rely on the fact that memory is generally read sequentially. Because you don’t really need to wait for each memory location to recharge itself, you can read the next location without waiting. Because you have to wait until the signal is stabilized, though, there is still some wait, though it is much less of a wait than waiting for the memory to recharge. At higher CPU speeds, the CPU requests memory faster than memory can deliver it, and the CPU needs to wait.

EDORAM works by "latching" the memory, which means adding secondary memory cells. These detect the data being read from memory and store the signals so the CPU can retrieve it. This works at bus speeds of 66Mhz. This process can be made even faster by including "burst" EDORAM, which extends the locality principle even further. Because the system is going to read sequentially, why doesn’t it anticipate the processor and read more than just that

single location? In some cases, the system will read 128 bits at once.

Part of the reason why EDORAM hasn't simply taken over the market is the similar to the reason why PS/2 didn't take over standard SIMMs: the hardware needed to support them is different. You cannot just install EDORAM in your machine and expect it to work. You need a special chip set on your motherboard. One such chip set is the Intel Triton chip set.

A newer memory type are the dual in-line memory modules or DIMMs. These look similar to the SIMM, but are generally larger. One key difference is that although both SIMMs and DIMMs have contacts on both sides of the board, the pins on opposite sides of the DIMMs are electrically isolated from each other, which creates two separate contacts. In addition, unlike SIMMs, you do not need to have pairs of DIMMS in your machine.

8.3.2 Cache Memory

Based on the principle of spatial locality, a program is more likely to spend its time executing code around the same set of instructions. This is demonstrated by the tests that have shown that most programs spend 80 percent of their time executing 20 percent of their code. Cache memory takes advantage of that.

Cache memory, or sometimes just cache, is a small set of very high-speed memory. Typically, it uses SRAM, which can be up to ten times more expensive than DRAM, which usually makes it prohibitive for anything other than cache.

When the IBM PC first came out, DRAM was fast enough to keep up with even the fastest processor. However, as CPU technology increased, so did its speed. Soon, the CPU began to outrun its memory. The advances in CPU technology could not be used unless the system was filled with the more expensive, faster SRAM.

The solution to this was a compromise. Using the locality principle, manufacturers of fast 386 and 486 machines began to include a set of cache memory consisting of SRAM but still populated main memory with the slower, less expensive DRAM.

To better understand the advantages of this scheme, lets cover the principle of locality in a little more detail. For a computer program, we deal with two types of locality: temporal time and spatial space. Because programs tend to run in loops repeating the same instructions, the same set of instructions must be read over and over. The longer a set of instructions is in memory without being used, the less likely it is to be used again. This is the principle of temporal locality. What cache memory does is enable us to keep those regularly used instructions "closer" to the CPU, making access to them much faster. This is shown graphically in Figure 0-10.

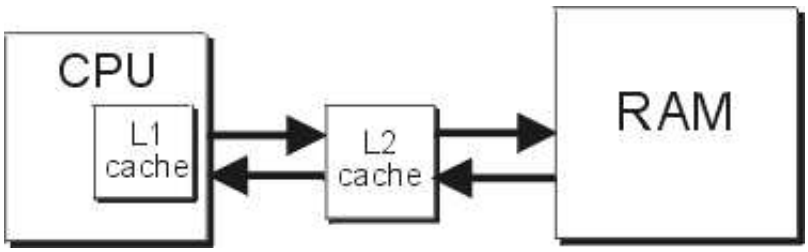


Image - Level 1 and Level 2 Caches

Spatial locality is the relationship between consecutively executed instructions. I just said that a program spends more of its time executing the same set of instructions. Therefore, in all likelihood, the next instruction the program will execute lies in the next memory location. By filling cache with more than just one instruction at a time, the principle of spatial locality can be used.

Is there really such a major advantage to cache memory? Cache performance is evaluated in terms of *cache hits.* A hit occurs when the CPU requests a memory location that is already in cache that is, it does not have to go to main memory to get it. Because most programs run in loops including the OS, the principle of locality results in a hit ratio of 85 to 95 percent. Not bad!

On most 486 machines, two levels of cache are used: level 1 cache and level 2 cache. Level 1 cache is internal to the CPU. Although nothing other than cost prevents it from being any larger, Intel has limited the level 1 cache in the 486 to 8k.

The level 2 cache is the kind that you buy separately from your machine. It is often part of the advertisement you see in the paper and is usually what people are talking about when they say how much cache is in their systems. Level 2 cache is external to the CPU and can be increased at any time, whereas level 1 cache is an integral part of the CPU and the only way to get more is to buy a different CPU. Typical sizes of level 2 cache range from 64K to 256K, usually in increments of 64K.

There is one major problem with dealing with cache memory: the issue of consistency. What happens when main memory is updated and cache is not? What happens when cache is updated and main memory is not? This is where the caches *write policy* comes in.

The write policy determines if and when the contents of the cache are written back to memory. The write-through cache simply writes the data through the cache directly into memory. This slows writes, but the data is consistent. Buffered write-through is a slight modification of this, in which data are collected and everything is written at once. Write-back improves cache performance by writing to main memory only when necessary. Write-dirty is when it writes to main memory only when it has been modified.

Cache or main memory, for that matter is referred to as "dirty" when it is written to. Unfortunately, the system has no way of telling whether anything has changed, just that it is being written to. Therefore it is possible, but not likely, that a block of cache is written back to memory even if it is not actually dirty.

Another aspect of cache is its organization. Without going into detail that would take most of a chapter itself, I can generalize by saying there are four different types of cache organization.

The first kind is fully associative, which means that every entry in the cache has a slot in the "cache directory" to indicate where it came from in memory. Usually these are not individual bytes, but chunks of four bytes or more. Because each slot in the cache has a separate directory slot, any location in RAM can be placed anywhere in the cache. This is the simplest scheme but also the slowest because each cache directory entry must be searched until a

match if any is found. Therefore, this kind of cache is often limited to just 4Kb.

The second type of cache organization is *direct-mapped* or *one-way set associative cache, which* requires that only a single directory entry be searched. This speeds up access time considerably. The location in the cache is related on the location in memory and is usually based on blocks of memory equal to the size of the cache. For example, if the cache could hold 4K 32-bit 4-byte entries, then the block with which each entry is associated is also 4K x 32 bits. The first 32 bits in each block are read into the first slot of the cache, the second 32 bits in each block are read into the second slot, and so on. The size of each entry, or line, usually ranges from 4 to 16 bytes.

There is a mechanism called a tag, which tells us which block this came from. Also, because of the very nature of this method, the cache cannot hold data from multiple blocks for the same offset. If, for example, slot 1 was already filled with the data from block 1 and a program wanted to read the data at the same location from block 2, the data in the cache would be overwritten. Therefore, the shortcoming in this scheme is that when data is read at intervals that are the size of these blocks, the cache is constantly overwritten. Keep in mind that this does not occur too often due to the principle of spatial locality.

The third type of cache organization is an extension of the one-way set associative cache, called the *two-way set associative*. Here, there are two entries per slot. Again, data can end up in only a particular slot, but there are two places to go within that slot. Granted, the system is slowed a little because it has to look at the tags for both slots, but this scheme allows data at the same offset from multiple blocks to be in the cache at the same time. This is also extended to four-way set associative cache. In fact, the cache internal to 486 and Pentium has a four-way set associate cache.

Although this is interesting at least to me, you may be asking yourself, "Why is this memory stuff important to me as a system administrator?" First, knowing about the differences in RAM main memory can aide you in making decisions about your upgrade. Also, as I mentioned earlier, it may be necessary to set switches on the motherboard if you change memory configuration.

Knowledge about cache memory is important for the same reason because you may be the one who will adjust it. On many machines, the write policy can be adjusted through the CMOS. For example, on my machine, I have a choice of write-back, write-through, and write-dirty. Depending on the applications you are running, you may want to change the write policy to improve performance.

8.4 The Central Processing Unit

Sometimes people just don’t understand. At first, I thought that they "didn’t have a clue," but that was really the problem. They had a clue, but a single clue doesn’t solve a crime, nor does it help you run a Linux system. You can easily copy a program from a DOS disk onto an Linux system, particularly if Linux is running on your DOS partition. In all likelihood, the permissions are already set to be executable. So you type in the name of the program and press Enter. Nothing happens, or you get an error about incorrect format. Hmmm. The software manual says that it runs on a 386 or higher (which you have), a VGA monitor (which

you have), and at least 2Mb of hard disk space (which you have). Why doesn't it work?

This is a true story. A customer called in saying that the system I was supporting at the time (not Linux) was broken. This customer had a program that worked fine on his DOS PC at home. It, too, was a 386, so there shouldn't be a problem, right? Unfortunately, wrong. Granted, in both cases, the CPU is reading machine instructions and executing them, but in fact, they are the same machine instructions. They have to be. The same also applies to a Linux system.

The problem is comparable to German and English. Although both use (basically) the same alphabet, words (sets of characters) written in German are not understandable by someone reading them as English, and vice versa. Sets of machine instructions that are designed to be interpreted under DOS will not be understood under Linux. (Actually, the problem is a little more complicated, but you get the basic idea.)

Just as your brain has to be told (taught) the difference between German and English, a computer needs to be told the difference between DOS and UNIX programs.

In this section, I will talk about the CPU, the brains of the outfit. It is perfectly reasonable for users and administrators alike to have no understanding of what the CPU does internally. However, a basic knowledge of some of the key issues is important so you can completely understand some of the issues I'll get into elsewhere.

Its like trying to tune-up your car. You don't really need to know how oxygen mixes with gasoline to be able to adjust the carburetor. However, knowing that it happens makes adjusting the carburetor that much easier.

I won't go into detail about the CPUs instruction cycle, that is, how it receives and executes instructions. Though I'm interested in things like that and would love to talk about them, it isn't really necessary to understand what we need to talk about here. Instead, I am going to talk mostly about how the CPU enables the operating system to create a scheme whereby many programs can be in memory simultaneously. These are the concepts of paging and multitasking.

Orignally, the only commercial distributions of Linux available were for Intel processors. RedHat released a version for the Digital Electronics Corporation (DEC) Alpha processor, and others have since followed.

In the next section, I will go into a little depth about the Intel process and how Linux interacts with it. Afterwards, I will talk briefly about the DEC Alpha to give you an idea of what it is about. Because of the number of Intel distributions and Intel-based machines, I won't go into the same depth for the Alpha. The concepts a basically the same, though the names of registers, etc., are different.

8.4.1 Intel Processors

Although it is an interesting subject, the ancient history of microprocessors is not really important to the issues at hand. It might be nice to learn how the young PC grew from a small, budding 4-bit system to the gigantic, strapping 64-bit Pentium. However, there are many

books that have covered this subject and unfortunately, I don't have the space. Besides, the Intel chips on which Linux runs are only the 80386 or 100-percent compatible clones and higher processors.

So, instead of setting the way-back machine to Charles Babbage and his Analytic Engine, we leap ahead to 1985 and the introduction of the Intel 80386. Even compared to its immediate predecessor, the 80286, the 80386 386 for short was a powerhouse. Not only could it handle twice the amount of data at once now 32 bits, but its speed rapidly increased far beyond that of the 286.

New advances were added to increase the 386s power. Internal registers were added and their size was increased. Built into the 386 was the concept of virtual memory, which was a way to make it appear as though there was much more memory on system than there actually was. This substantially increased the system efficiency. Another major advance was the inclusion of a 16-byte, pre-fetch cache. With this, the CPU could load instructions before it actually processed them, thereby speeding things up even more. Then the most obvious speed increase came when the speed of the processor was increased from 8Mhz to 16Mhz.

Although the 386 had major advantages over its predecessors, at first its cost seemed relatively prohibitive. To allow users access to the multitasking capability and still make the chip fit within their customers budgets, Intel made an interesting compromise: By making a new chip in which the interface to the bus was 16-bits instead of 32-bits, Intel made their chip a fair bit cheaper.

Internally, this new chip, designated the 80386SX, is identical to the standard 386. All the registers are there and it is fully 32 bits wide. However, data and instructions are accessed 16 bits at a time, therefore requiring two bus accesses to fill the registers. Despite this shortcoming, the 80386SX is still faster than the 286.

Perhaps the most significant advance of the 386 for Linux as well as other PC-based UNIX systems was its paging abilities. I talked a little about paging in the section on operating system basics, so you already have a general idea of what paging is about. I will also go into more detail about paging in the section on the kernel. However, I will talk about it a little here so you can fully understand the power that the 386 has given us and see how the CPU helps the OS.

There are UNIX-like products that run on a 80286, such as SCO XENIX. In fact, there was even a version of SCO XENIX that ran on the 8086. Because Linux was first released for the 386, I won't go into anymore detail about the 286 or the differences between the 286 and 386. Instead, I will just describe the CPU Linux used as sort of an abstract entity. In addition, because most of what I will be talking about is valid for the 486 and Pentium as well as the 386, I will simply call it "the CPU" instead of 386, 486, or Pentium.

Note: Linux will also run on non-Intel CPUs, such as those from AMD or Cyrix. However, the issues I am going to talk about are all common to Intel-based or Intel-derived CPUs.

I need to take a side-step here for a minute. On PC-Buses, multiple things are happening at once. The CPU is busily processing while much of the hardware is being access via DMA. Although these multiple tasks are occurring simultaneously on the system, this is not what is

referred to as multitasking.

When I talk about multitasking, I am referring to multiple processes being in memory at the same time. Because of the time the computer takes to switch between these processes, or tasks, is much shorter than the human brain can recognize, it appears as though the processes are running simultaneously. In reality, each process gets to use the CPU and other system resources for a brief time and then its another process's turn.

As it runs, the process could use any part of the system memory it needs. The problem with this is that a portion of RAM that one process wants may already contain code from another process. Rather than allowing each process to access any part of memory it wants, protections keep one program from overwriting another one. This protection is built in as part of the CPU and is called, quite logically, "protected mode." Without it, Linux could not function.

Note, however, that just because the CPU is in protected mode does not necessarily mean that the protections are being utilized. It simply means that the operating system can take advantage of the built-in abilities if it wants.

Although this capability is built into the CPU, it is not the default mode. Instead, the CPU starts in what I like to call "DOS-compatibility mode." However, the correct term is "real mode." Real mode is a real danger to an operating system like UNIX. In this mode, a there are no protections which makes sense because protections exist only in protected mode. A process running in real mode has complete control over the entire system and can do anything it wants. Therefore, trying to run a multiuser system on a real-mode system would be a nightmare. All the protections would have to be built into the process because the operating system wouldn't be able to prevent a process from doing what it wanted.

A third mode, called "virtual mode," is also built in. In virtual mode, the CPU behaves to a limited degree as though it is in real mode. However, when a process attempts to directly access registers or hardware, the instruction is caught, or trapped, and the operating system is allowed to take over.

Lets get back to protected mode because this is what makes multitasking possible. When in protected mode, the CPU can use virtual memory. As I mentioned, this is a way to trick the system into thinking that there is more memory than there really is. There are two ways of doing this. The first is called swapping, in which the entire process is loaded into memory. It is allowed to run its course for a certain amount of time. When its turn is over, another process is allowed to run. What happens when there is not enough room for both process to be in memory at the same time? The only solution is that the first process is copied out to a special part of the hard disk called the swap space, or swap device. Then, the next process is loaded into memory and allowed its turn. The second is called paging and we will get to it in a minute.

Because it takes such a large portion of the system resources to swap processes in and out of memory, virtual memory can be very inefficient, especially when you have a lot of processes running. So lets take this a step further. What happens if there are too many process and the system spends all of its time swapping? Not good.

To avoid this problem, a mechanism was devised whereby only those parts of the process that are needed are in memory. As it goes about its business, a program may only need to access a small portion of its code. As I mentioned earlier, empirical tests show that a program spends 80 percent of its time executing 20 percent of its code. So why bother bringing in those parts that aren't being used? Why not wait and see whether they are used?

To make things more efficient, only those parts of the program that are needed or expected to be needed are brought into memory. Rather than accessing memory in random units, the memory is divided into 4K chunks, called pages. Although there is nothing magic about 4K per se, this value is easily manipulated. In the CPU, data is referenced in 32-bit 4-byte chunks, and 1K 1,024 of each chunk is a page 4,096. Later you will see how this helps things work out.

As I mentioned, only that part of the process currently being used needs to be in memory. When the process wants to read something that is not currently in RAM, it needs to go out to the hard disk to pull in the other parts of the process; that is, it goes out and reads in new pages. This process is called paging. When the process attempts to read from a part of the process that is not in physical memory, a "page fault" occurs.

One thing you must bear in mind is that a process can jump around a lot. Functions are called, sending the process off somewhere completely different. It is possible, likely, for that matter, that the page containing the memory location to which the process needs to jump is not currently in memory. Because it is trying to read a part of the process not in physical memory, this, too, is called a page fault. As memory fills up, pages that haven't been used in some time are replaced by new ones. I'll talk much more about this whole business later.

Assume that a process has just made a call to a function somewhere else in the code and the page it needed is brought into memory. Now there are two pages of the process from completely different parts of the code. Should the process take another jump or return from the function, it needs to know whether it is going into memory. The operating system could keep track of this, but it doesn't need to the CPU will keep track for it.

Stop here for a minute! This is not entirely true. The OS must first set up the structures that the CPU uses. However, the CPU uses these structures to determine whether a section of a program is in memory. Although not part of the CPU, but rather RAM, the CPU administers the RAM utilization through page tables. As their names imply, page tables are simply tables of pages. In other words, they are memory locations in which other memory locations are stored.

Confused? I was at first, so lets look at this concept another way. Each running process has a certain part of its code currently in memory. The system uses these page tables to keep track of what is currently memory and where it is located. To limit the amount the CPU has to work, each of these page tables is only 4K, or one page, in size. Because each page contains a set of 32-bit addresses, a page table can contain only 1,024 entries.

Although this would imply that a process can only have 4K x 1,024, or 4Mb, loaded at a time, there is more to it. Page tables are grouped into page directories. Like the page table, the entries in a page directory point to memory locations. However, rather than pointing to a part

of the process, page directories point to page tables. Again, to reduce the CPUs work, a page directory is only one page. Because each entry in the page directory points to a page, this means that a process can only have 1,024 page tables.

Is this enough? Lets see. A page is 4K or 4,096 bytes, which is 2^{12}. Each page table can refer to 1,024 pages, which is 2^{10}. Each page directory can refer to 1,024 page tables, which is also 2^{10}. Multiplying this out, we have

page size x pages in page table x page tables in page directory

or

$2^{12} \times 2^{10} \times 2^{10} = 2^{32}$

Because the CPU is only capable of accessing 2^{32} bytes, this scheme allows access to every possible memory address that the system can generate.

Are you still with me?

Inside of the CPU is a register called the Control Register 0, or CR0 for short. In this register is a single bit that turns on this paging mechanism. If this paging mechanism is turned on, any memory reference that the CPU receives is interpreted as a combination of page directories, page tables, and offsets, rather than an absolute, linear address.

Built into the CPU is a special unit that is responsible for making the translation from the virtual address of the process to physical pages in memory. This special unit is called what else? the paging unit. To understand more about the work the paging unit saves the operating system or other parts of the CPU, lets see how the address is translated.

Translation of Virtual-to-Physical Address

When paging is turned on, the paging unit receives a 32-bit value that represents a virtual memory location within a process. The paging unit takes theses values and translates them, as shown in Figure 0-11. At the top of the figure, we see that the virtual address is handed to the paging unit, which converts it to a linear address. This is not the physical address in memory. As you see, the 32-bit linear address is broken down into three components. The first 10 bits 22-31 are offset into the page directory. The location in memory of the page directory is determined by the Page Directory Base Register PDBR.

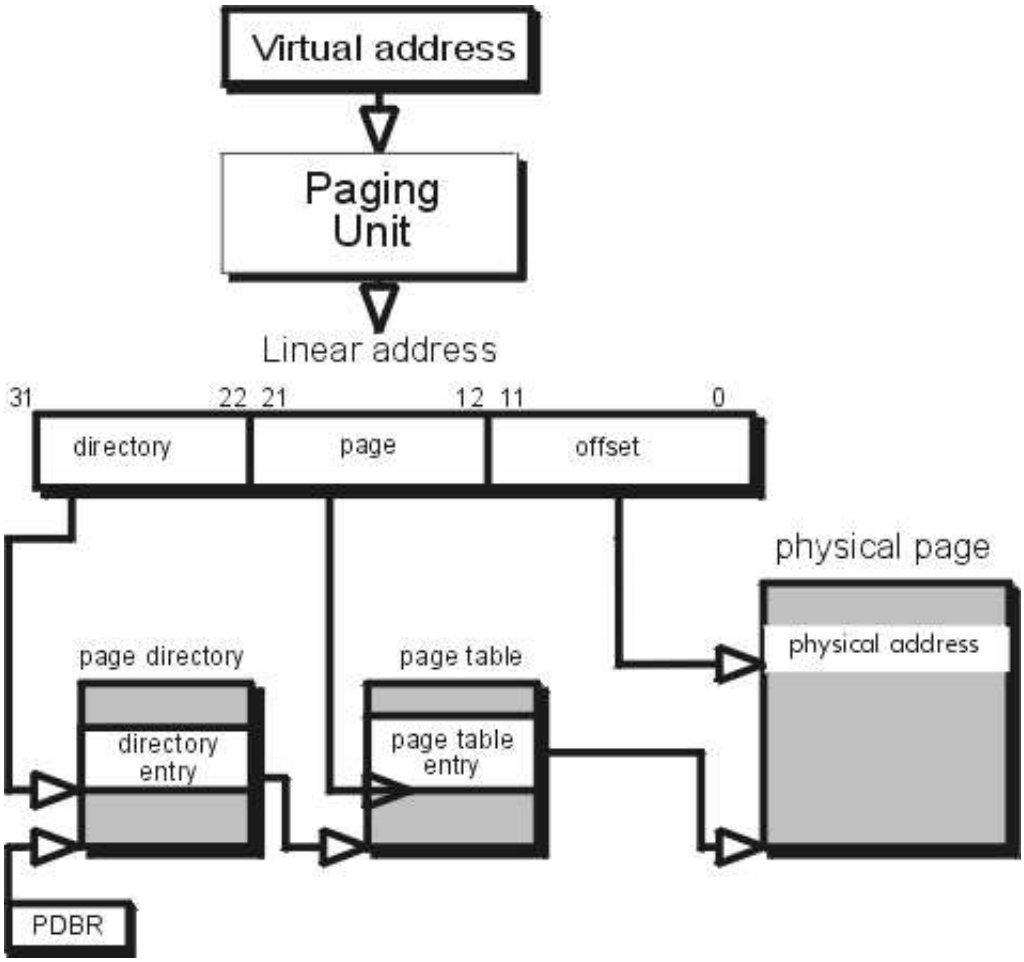


Image - Translation of virtual addresses into physical addresses by the paging unit.

The page directory entry contains 4 bits that point to a specific page table. The entry in the page table, as you see, is determined by bits 1221. Here again, we have 10 bits, which means each entry is 32 bits. These 32-bit entries point to a specific page in *physical* memory. Which byte is referenced in physical memory is determined by the offset portion of the linear address, which are bits 011. These 12 bits represent the 4,096 4K bytes in each physical page.

Keep in mind a couple of things. First, page tables and page directories are not part of the CPU. They can't be. If a page directory were full, it would contain 1,024 references to 4K chunks of memory. For the page tables alone, you would need 4Mb! Because this would create a CPU hundreds of times larger than it is now, page tables and directories are stored in RAM.

Next, page tables and page directories are abstract concepts that the CPU knows how to utilize. They occupy physical RAM, and operating systems such as Linux know how to switch this capability on within the CPU. All the CPU does is the "translation" work. When it starts, Linux turns this capability on and sets up all the structures. These structures are then handed off to the CPU, where the paging unit does the work.

As I said, a process with all of its page directory entries full would require 4Mb just for the page tables. This implies that the entire process is somewhere in memory. Because each of the

page table entries points to physical pages in RAM, you would need 16Gb of RAM. Not that I would mind having that much RAM, though it is a bit costly and even if you had 16Mb SIMMs, you would need 1000 of them.

Like pages of the process, it's possible that a linear address passed to the paging unit translates to a page table or even a page directory that was not in memory. Because the system is trying to access a page which contains a page table and not part of the process that is not in memory, a page fault occurs and the system must go get that page.

Because page tables and the page directory are not really part of the process but are important only to the operating system, a page fault causes these structures to be *created* rather than read in from the hard disk or elsewhere. In fact, as the process starts up, all is without form and is void: no pages, no page tables, and no page directory.

The system accesses a memory location as it starts the process. The system translates the address, as I described above, and tries to read the page directory. It's not there. A page fault occurs and the page directory must be created. Now that the directory is there, the system finds the entry that points to the page table. Because no page tables exist, the slot is empty and another page fault occurs. So, the system needs to create a page table. The entry in the page table for the physical page is found to be empty, and so yet another page fault occurs. Finally, the system can read in the page that was referenced in the first place.

This whole process sounds a bit cumbersome, but bear in mind that this amount of page faulting only occurs as the process is starting. Once the table is created for a given process, it won't page fault again on that table. Based on the principle of locality, the page tables will hold enough entries for a while, unless, of course, the process bounces around a lot.

The potential for bouncing around brings up an interesting aspect of page tables. Because page tables translate to physical RAM in the same way all the time, virtual addresses in the same area of the process end up in the same page tables. Therefore, page tables fill up because the process is more likely to execute code in the same part of a process rather than elsewhere this is spatial locality.

There is quite a lot there, yes? Well, don't get up yet because were not finished. There are a few more issues that I haven't addressed.

First, I have often referred to page tables and *the* page directory. Each process has a single page directory it doesn't need any more. Although the CPU supports multiple page directories, there is only one directory for the *entire* system. When a process needs to be switched out, the entries in the page directory for the old process are overwritten by those for the new process. The location of the page directory in memory is maintained in the Control Register 3 CR3 in the CPU.

There is something here that bothered me in the beginning and may still bother you. As I have described, each time a memory reference is made, the CPU has to look at the page directory, then a page table, then calculate the physical address. This means that for *every* memory reference, the CPU has to make two more references just to find out where the next instruction or data is coming from. I though that was pretty stupid.

Well, so did the designers of the CPU. They have included a functional unit called the Translation Lookaside Buffer, or TLB. The TLB contains 32 entries and, as the internal and external caches point to sets of instructions, points to pages. If a page that is being searched is in the TLB, a TLB hit occurs just like a cache hit. As a result of the principle of spatial locality, there is a 98-percent hit rate using the TLB.

When you think about it, this makes a lot of sense. The CPU does not just execute one instruction for a program then switch to something else, it executes hundreds or even thousands of instructions before another program gets its turn. If each page contains 1,024 instructions and the CPU executes 1000 before it's another programs turn, all 1000 will most likely be in the same page. Therefore, they are all TLB hits.

Now, lets take a closer look at the page table entries themselves. Each is a 32-bit value that points to a 4K location in RAM. Because it points to an area of memory larger than a byte, it does not need all 32 bits to do it. Therefore, some bits are left over. Because the page table entry points to an area that has 2^{20} bytes 4,096 bytes = 1 page, it doesn't need 12 bits. These are the low-order 12 bits and the CPU uses them for other purposes related to that page. A few of them are unused and the operating system can, and does, use them for its own purposes. Intel also reserves a couple, and they should not be used.

One bit, the 0th bit, is the present bit. If this bit is set, the CPU knows that the page being referenced is in memory. If it is not set, the page is not in memory and if the CPU tries to access it, a page fault occurs. Also, if this bit is not set, none of the other bits has any meaning. How can you talk about something that's not there?

Another important bit is the accessed bit. Should a page be accessed for either read or write, the CPU sets this bit. Because the page table entry is never filled in until the page is being accessed, this seems a bit redundant. If that was all there was to it, you'd be right. However, there's more.

At regular intervals, the operating system clears the access bit. If a particular page is never used again, the system is free to reuse that physical page if memory gets short. When that happens, all the OS needs to do is clear the present bit so the page is considered "invalid."

Another bit used to determine how a page is accessed is the dirty bit. If a page has been written to, it is considered dirty. Before the system can make a dirty page available, it must make sure that whatever was in that page is written to disk, otherwise the data is inconsistent.

Finally, we get to the point of what all this protected mode stuff is all about. The protection in protected mode essentially boils down to two bits in the page table entry. One bit, the user/supervisor bit, determines who has access to a particular page. If the CPU is running at user level, then it only has access to user-level pages. If the CPU is at supervisor level, it has access to all pages.

I need to say here that this is the maximum access a process can have. Other protections may prevent a user-level or even supervisor-level process from getting even this far. However, these are implemented at a higher level.

The other bit in this pair is the read/write bit. As the name implies, this bit determines whether a page can be written to. This single bit is really just an on-off switch. If the page is there, you have the right to read it if you can that is, either you are a supervisor-level process or the page is a user page. However, if the write ability is turned off, you cant write to it, even as a supervisor.

If you have a 386 CPU, all is well. If you have a 486 and decide to use one of those bits that I told you were reserved by Intel, you are now running into trouble. Two of these bits were not defined in the 386 but are now defined in the 486: page write-through PWT and page cache disable PCD.

PWT determines the write policy see the section on RAM for external cache regarding this page. If PWT is set, then this page has a write-through policy. If it is clear, a write-back policy is allowed.

PCD decides whether this page can be cached. If clear, this page cannot be cached. If set, then caching is allowed. Note that I said "allowed." Setting this bit does not mean that the page will be cached. Other factors that go beyond what I am trying to get across here are involved.

Well, I've talked about how the CPU helps the OS keep track of pages in memory. I also talked about how the CR3 register helps keep track of which page directory needs to be read. I also talked about how pages can be protected by using a few bits in the page table entry. However, one more thing is missing to complete the picture: keeping track of which process is currently running, which is done with the Task Register TR.

The TR is not where most of the work is done. The CPU simply uses it as a pointer to where the important information is kept. This pointer is the Task State Descriptor TSD. Like the other descriptors that I've talked about, the TSD points to a particular segment. This segment is the Task State Segment TSS. The TSD contains, among other things, the privilege level at which this task is operating. Using this information along with that in the page table entry, you get the protection that protected mode allows.

The TSS contains essentially a snapshot of the CPU. When a process's turn on the CPU is over, the state of the entire CPU needs to be saved so that the program can continue where it left off. This information is stored in the TSS. This functionality is built into the CPU. When the OS tells the CPU a task switch is occurring that is, a new process is getting its turn, the CPU knows to save this data *automatically*.

If we put all of these components together, we get an operating system that works together with the hardware to provide a multitasking, multiuser system. Unfortunately, what I talked about here are just the basics. I could spend a whole book just talking about the relationship between the operating system and the CPU and still not be done.

One thing I didn't talk about was the difference between the 80386, 80486, and Pentium. With each new processor comes new instructions. The 80486 added an instruction pipeline to improve the performance to the point where the CPU could average almost one instruction per cycle. The Pentium has dual instructions paths pipelines to increase the speed even more. It also contains *branch prediction logic*, which is used to "guess" where the next instruction should come from.

The Pentium as well as the later CPUs has a few new features that make for significantly more performance. This first feature is multiple instruction paths or pipelines, which allow the CPU to work on multiple instructions at the same time. In some cases, the CPU will have to wait to finish one before working on the other, though this is not always necessary.

The second improvement is called dynamic execution. Normally, instructions are executed one after other. If the execution order is changed, the whole program is changed. Well, not exactly. In some instances, upcoming instructions are not based on previous instructions, so the processor can "jump ahead" and start executing the executions before others are finished.

The next advance is branch prediction. Based on previous activity, the CPU can expect certain behavior to continue. For example, the odds are that once the CPU is in a loop, the loop will be repeated. With more than one pipeline executing instruction, multiple possibilities can be attempted. This is not always right, but is right more than 75 percent of the time!

The PentiumPro P6 introduced the concept of data flow analysis. Here, instructions are executed as they are ready, not necessarily in the order in which they appear in the program. Often, the result is available before it normally would be. The PentiumPro P6 also introduced speculative execution, in which the CPU takes a guess at or anticipates what is coming.

The P6 is also new in that it is actually two separate chips. However, the function of the second chip is the level 2 cache. Both an external bus and a "private" bus connect the CPU to the level 2 cache, and both of these are 64 bits.

Both the Socket and the CPU itself changed with the Pentium II processor. Instead of a processor with pins sticking out all over the bottom, the Pentium II uses a Single Edge Contact Cartridge SECC. This reportedly eliminates the need for resigning the socket with every new CPU generation. In addition, the CPU is encased in plastic, which protects the CPU during handling. Starting at "only", the Pentium II can reach speeds of up to 450 MHz.

Increasing performance even further, the Pentium II has increased the internal, level-one cache to 32Kb, with 16 Kb for data and 16Kb for instructions. Technically it may be appropriate to call the level-two cache internal, as the 512KB L2 cache is included within the SECC, making access faster than for a traditional L2 cache. The Dual Independent Bus DIB architecture provides for higher throughput as there are separate system and cache buses.

The Pentium II also increases performance internally through changes to the processor logic. Using Multiple Branch Prediction, the Pentium predicts the flow of instructions through several branches. Because computers usually process instructions in loops i.e. repeatedly it is generally easy to guess what the computer will do next. By predicting multiple branches, the processor reduces "wrong guesses."

Processor "management" has become an important part of the Pentium II. A Built-In Self-Test BIST is included, which is used to test things like the cache and the TLB. It also includes a diode within the case to monitor the processor’s temperature.

The Pentium II Xeon Processor added a "system bus management interface," which allows the CPU to communicate with other system management components hardware and software. The thermal sensor, which was already present in the Pentium II, as well as the new Processor

Information ROM PI ROM and the Scratch EEPROM use this bus.

The PI ROM contains various pieces of information about the CPU, like the CPU ID, voltage tolerances and other technical information. The Scratch EEPROM is shipped blank from Intel but is intended for system manufacturers to include whatever information they want to, such as an inventory of the other components, service information, system default and so forth.

Like the Pentium II, the latest as of this writing processor, the Pentium III also comes in the Single Edge Contact Cartridge. It has increased the number of transistors from the 7.5 million in the Pentium II to over 9.5 million. Currently, the Pentium III comes in 450Mhz and 500 MHz models, with a 550MHz model in the planning.

The Pentium II also includes the Internet Streaming SIMD Extensions, which consist of 70 new instructions which enhance imagining in general, as well as 3D graphics, streaming audio and video as well as speech recognition.

Intel also added a serial number to the Pentium II. This is extremely useful should the CPU itself get stolen or the computer get stolen, after the CPU has been installed. In addition, the CPU can be uniquely identified across the network, regardless of the network card or other components. This can be used in the future to prevent improper access to sensitive data, aid in asset management and help in remote management and configuration.

Even today in the people still think that the PC CPU is synonymous with Intel. That is if you are going to buy a CPU for your PC that it will be manufactured by Intel. This is not the case. Two manufactures Advanced Micro Devices ADM and Cyrix provide CPUs with comparable functionality. Like any other brand name, Intel CPUs are often more expensive than an equivalent from another company with the same performance.

8.4.2 AMD

More than likely, you have seen the stickers on the front of computers saying "Intel Inside." As you might guess, this computer has an Intel processor. This sticker and the associated aide campaign is important for name recognition. For many people, the name Intel has become synonymous with CPUs for PCs.

Many people may have heard the name of other CPU vendors, but often feel they are simply cheap "clones." This is unfortunate, because the performance of these CPUs is comparable to the Intel CPUs. Although these other vendors generally release chips with the same performance several months after the comparable one from Intel, they are typically less expensive and therefore have a better price-performance ratio.

This is where business buyers really look. If a product provides the necessary performance and reliability at a lower price, it does not make business sense to pay for something just because it has an expensive television add. Therefore, more and more business, including PC manufacturers are switching to AMD CPUs.

One of the first successful "clones" of the Intel CPUs was the AMD AM5x86. The first assumption is that the "5" in that its name indicates that it is a clone of the Intel Pentium. Instead, it is much better to think of the AM5x86 at a high-end version of the 486. Tests have

shown that a 133Mhz AM5x86 will not quite outperformed a 90MHz Pentium, but will outperform one at 70 MHz. Because of the reduced cost, you still get better performance dollar for dollar, despite the 48 faster processor.

The AMD K5 was the first Pentium-class CPU developed by AMD. One interesting aspect of the case five, is that it "translates" the Intel instructions into fixed length RISC instructions. This makes executing the instructions a lot faster, because the CPU does not need to waste time figuring out how on the instruction really is. In addition, since all instructions are the same length, they can be loaded more efficiently into the K5's six-stage instructions pipeline, which can process for instruction simultaneously.

Following the K5, AMD logically came out with the K6. One benefit of this CPU was the fact that and was the first on market which used Intel's own MMX technology. In addition, the K6 has instruction pipelines which are fed by a set of four instruction decoders. Like the K5, the K6 translates the Intel instructions into RISC instructions before executing them. Added to that the K6 has separate instruction and data caches like the Pentium, but those in the K6 or four times as large (32KB).

The successor to the successful AMD-K6 series is the AMD-K6-2, which is the first CPU to offers AMD's 3DNow! Technology. As you might guess from its name implies, 3DNow! improves system performance when displaying 3D graphics, something Intel's MMX technology was not designed to do. However, MMX does provide some performance improvements, so the AMDK6-2 includes MMX, as well. As of this writing, the AMD-K6-2 is available in speeds from 300MHz to 475MHz.

Next came the AMD-K6-III series. As with the AMD-K6-2, the AMD-K6-III series also provides AMD's 3DNow! technology. One of the most significant improvements is the addition of an addition CPU, which give it a "tri-level" cache. Thus providing a maximum cache of 2.3MB, which is more than four times as much as possible with the Intel Pentium III Processors. In addition, the L2 cache operates at the same speed as the CPU. This means the 450MHz version has the potential to outperform a 500MHz Pentium III.

The next step is the AMD-K7 processor. As of this writing, it has not yet been released, but the features announced by AMD are exciting. One important aspect is that it is expected to be the first CPU to support a 200 MHz system bus. This includes a nine stage, superscalar execution pipeline, with a 128KB L1 cache. This is twice what is currently available.

8.4.3 Alpha Processors

The Alpha processor from the Digital Equipment Corporation (DEC) is the first non-Intel-based processor on which Linux was commercially available and is substantially different from its Intel cousins. The most significant difference is that the Alpha AXP is a Reduced Instruction Set Computer (RISC), as compared to the Intel, which is a Complex Instruction Set Computer (CISC). Without turning this book into a textbook on microprocessor design, I can simply say that the difference is that RISC has fewer instructions (the instruction set is reduced) and therefore takes more instructions to do a specific job. A CISC processor has more instructions and takes fewer instructions to do the same job.

Imagine someone told you to stand up and open the door. The CISC instruction might simply say, "Go open the door." Using what's built into your CPU (your brain), you know to translate this to "stand up and open the door." On the other hand, the RISC instructions might be "Stand up. Turn left and take two paces forward. Turn right, and take three paces forward. Raise right hand, etc." There might then be a CISC instruction that says, "Go open the window." However, the RISC instructions might be "Stand up. Turn left, and take two paces forward. Turn left, and take three paces forward, etc."

Not only does the CISC give fewer instructions, it also requires less logic circuitry. As a result, an Alpha AXP processor can run at higher speeds than an Intel. This does *not* make the Alpha AXP intrinsically faster! Take our example. I simply tell you to open the window, and you do it. However, giving you each instruction individually takes more time. One significant difference is that when the PentiumPro just broke the 200Mhz barrier, the Alphas were more than twice that.

Even if the increase in clock speed is not considered, the design of the Alpha AXP enables it to do more work per cycle. Several issues were addressed to help eliminate any aspect of the processor that would hinder multiple instruction issues. For example, there are no branch delay or skip instructions. As a result of its design, the Alpha AXP (as of this writing) can get up to 10 new instructions per cycle.

In addition, the Alpha AXP was designed to run with multiple processors, though that's not to say that it can't run as a single processor. The Alpha AXP was designed with several instructions that simplify adding multiple processors. Unlike other processors, this functionality was designed from the beginning and didn't have to be built onto an existing system.

One advantage of Alpha AXP is that it doesn't have a lot of baggage to carry around. The Intel 80x86 family is based on the 8086 and is completely backward-compatible. If you have an 8086 program, it will run on an PentiumPro. The Alpha AXP was developed with a full 64-bit architecture, although it has a few 32-bit operations for backward compatibility.

Part of the 64-bit architecture is the Alphas 64-bit virtual address space. All values (registers, addresses, integers, etc.) are operated on as full 64-bit quantities. Unlike with the Intel processors, there are no segmented addresses. In some cases, the operating system may restrict the number of bits that is used to translate the virtual address; however, at least 43 bits are used.

Like the Intel, Alphas memory protection is done on a per-page basis. The design of the paging mechanism in Intel specifies a 4KB page, but the Alpha AXP can have 8KB, 16KB, 32KB, or even 64KB pages. In addition, the Alpha AXP also uses many-to-one page mapping, as does the Intel, so that multiple processors can have a virtual memory address that references the same page in physical memory.

The Alpha AXP architecture is a 64-bit load/store RISC architecture designed with speed in mind. All registers are 64 bits in length; 32 integer registers and 32 floating point registers. Integer register 31 and floating point register 31 are used for null operations. A read from them generates a zero value and a write to them has no effect. All instructions are 32 bits long

and memory operations are either reads or writes. The architecture allows different implementations so long as the implementations follow the architecture.

There are no instructions that operate directly on values stored in memory; all data manipulation is done between registers. So, if you want to increment a counter in memory, you first read it into a register, then modify it and write it out. The instructions only interact with each other by one instruction writing to a register or memory location and another register reading that register or memory location. One interesting feature of Alpha AXP is that there are instructions that can generate flags, such as testing if two registers are equal, the result is not stored in a processor status register, but is instead stored in a third register. This may seem strange at first, but removing this dependency from a status register means that it is much easier to build a CPU which can issue multiple instructions every cycle. Instructions on unrelated registers do not have to wait for each other to execute as they would if there were a single status register. The lack of direct operations on memory and the large number of registers also help issue multiple instructions.

The Alpha AXP architecture uses a set of subroutines, called privileged architecture library code (PALcode). PALcode is specific to the operating system, the CPU implementation of the Alpha AXP architecture and to the system hardware. These subroutines provide operating system primitives for context switching, interrupts, exceptions and memory management. These subroutines can be invoked by hardware or by CALL_PAL instructions. PALcode is written in standard Alpha AXP assembler with some implementation specific extensions to provide direct access to low level hardware functions, for example internal processor registers. PALcode is executed in PALmode, a privileged mode that stops some system events happening and allows the PALcode complete control of the physical system hardware.

8.4.4 Mips

There is nothing here, because I have no experience at all with MIPS processors. If anyone could write up a short page on them I would **really** appreciate it. Please let me know.

8.4.5 SPARC

There is nothing here, because I have no experience at all with SPARC processors. If anyone could write up a short page on them I would **really** appreciate it. Please let me know.

8.4.6 ARM Processors

The ARM processor implements a low power, high performance 32 bit RISC architecture. It is being widely used in embedded devices such as mobile phones and PDAs (Personal Data Assistants). It has 31 32 bit registers with 16 visible in any mode. Its instructions are simple load and store instructions (load a value from memory, perform an operation and store the result back into memory). One interesting feature it has is that every instruction is conditional. For example, you can test the value of a register and, until you next test for the same condition, you can conditionally execute instructions as and when you like. Another interesting feature is that you can perform arithmetic and shift operations on values as you load them. It operates in several modes, including a system mode that can be entered from user mode via a SWI (software interrupt).

It is a synthasisable core and ARM (the company) does not itself manufacture processors. Instead the ARM partners (companies such as Intel or LSI for example) implement the ARM architecture in silicon. It allows other processors to be tightly coupled via a co-processor interface and it has several memory management unit variations. These range from simple memory protection schemes to complex page hierarchies.

8.5 Motherboards

Even if you buy pre-configured computers, you should still consider what kind of motherboards you are getting. It is very common today to find a number of the devices, which were formally expansion car, are now integrated into the motherboard. In many cases, the motherboard is smaller and therefore the total cost is reduced. However, this means if you wish to use something other than what indeed motherboard manufacturer has provided for you, you'll need to spend additional money as well as needed away to disable the device on the motherboard.

The reason the motherboard becomes smaller is that it can "get away with" having fewer expansion bus slots. Since the hard disk controller, for example, is integrated into motherboard, you do not need to use an expansion slot for. If this were a controller for IDE drives, you are less likely to want to buy one your own. However, if the SCSI host adapter is built and, you want to use something more powerful than the one which is provided, have to take up of the main expansion slots with the additional SCSI host adapter.

Another problem motherboard design brings with it is the placement of the integrated controllers. In some cases, I have found the plugs for such devices stuck between the expansion slots. While this does a great job of saving space, it makes it extremely difficult to access the pins. The only way to connect the cable to the pins was to remove all of the cards. However, you had to be extremely careful when ypu put the cards back in so as not to pull a cable off of the pins. Although it is unlikely you will be changing expansion cards every day, the headache and wasted time often negates any benefit of having paid $10 less for the motherboard.

Most were motherboards which I have encountered, with three PCI slots and at least three ISA slots. Some come with either an addition PCI or ISA slot, while some have an AGP slot. However, you can expect to have at least six expansion slots makes between PCI and ISA.

One thing you need to be careful about when shopping for motherboards is whether or not they support your chosen CPU. People do not often have a CPU before they have the motherboard (although I did once), you may have decided on a particular CPU before you buy it and the motherboard.

The days are gone in which you could simply bought a "PC motherboard" and expected to work with your CPU. The technology is changing so fast and there are so many different kinds of CPUs on the market, you need to be absolutely sure the CPU is compatible with the motherboard. Most of the motherboard manufacturers have Web sites with a compatibility matrix. You can find out which CPUs are supported and which clock speeds.

Sockets

One thing to consider when buying in motherboard for your CPU is where you are going to plug in that CPU. Not all CPUs are alike and non-all sockets for the CPUs are alike. As of this writing, nine different socket types (0-8) have been defined.

Socket Designation	*Number of Pins*	*Pin Layout*	*Voltage*	*CPU*	*OverDrive Processors*
0	168	In line	5V;	486DX	DX2, DX4
1	169	In line	5V	486DX, 486SX	DX2, DX4
2	238	In line	5V	486DX, 486SX, DX2	DX2, DX4, Pentium
3	237	In line	3V or 5V	486DX, 486SX, DX2, DX4	DX2, DX4, Pentium
4	273	In line	5V	60 and 66 MHz Pentium	Pentium
5	320	Staggered	3V	Other Pentium	Pentium
6	235	In line	3V	DX4	Pentium
7	321	Staggered	3V	Other Pentium	Pentium
8	387	Staggered	3V	Pentium Pro	Pentium Pro

There area several things to note about this table. First, There was never an officially designated socket 0, but Intel made a line of socket for 486 OverDrive processors which followed the Socket 0 design.

Second, the difference between an in-line and staggered pin layout is simply whether or not the pins line up in different rows.

The inner 169 pins of Socket 2 match those of socket 1, so you can simply plug in a CPU that is normally intended for Socket 1. The only difference is that the outer pins are open.

You will see that Socket 3 is the same size as Socket 2, but has one pin missing and the "keying pins" are in different place. It supports CPUs with either 3V or 5V and the rearranged keying pins help prevent someone from accidentally putting a 3V CPU into a 5V machine.

Socket 4 was for the first Pentiums, but is no longer used. It was followed by the Socket 5, which had a staggered pin layout. Socket 6 had a similar layout to sockets 2 and 3, but was only able to handle the 486DX4.

Finally, we get to Socket 7, which is currently the most common for Pentium based machines. The Pentium Pro CPUs fit into Socket 8.

8.6 Hard Disks

You've got to have a hard disk. You could actually run Linux from a floppy even a high-density 5.25" floppy, but life is so much easier when you run from the hard disk. Not only do you have space to save your files, you have access to all the wonderful tools that Linux provides. Disk drives provide a more permanent method for storing data, keeping it on spinning disk platters. To write data, a tiny head magnetizes minute particles on the platter's surface. The data is read by a head, which can detect whether a particular minute particle is magnetized. Note that running Linux off of a floppy, such as as a router or fire wall is not a bad idea. You still have the basic functionality you need and there is very little to hack.

A hard disk is composed of several physical disks, called platters, which are made of aluminum, glass or ceramic-composites and coated with either an "oxide" media the stuff on the disks or "thin film" media. Because "thin film" is thinner than oxide, the denser that is, larger hard disks are more likely to have thin film. The more platters you have, the more data you can store.

Platters are usually the same size as floppies. Older platters were 5.25" round, and the newer ones are 3.5" round. If someone knows the reason for this, I would love to hear it. In the center of each platter is a hole though which the spindle sticks. In other words, the platters rotate around the spindle. The functionality is the same as with a phonograph record. Remember those? The platters spin at a constant speed that can vary between 3000 and 15,000 RPM depending on the model. Compare this to a floppy disk which only spins at 360 RPM. The disk's read/write heads are responsible for reading and writing data and there is a pair for each platter, one head for each surface. The read/write heads do not physically touch the surface of the platters, instead they float on a very thin 10 millionths of an inch cushion of air. The read/write heads are moved across the surface of the platters by an actuator. All of the read/write heads are attached together, they all move across the surfaces of the platters together.

The media that coats the platters is very thin. about 30 millionths of an inch. The media has magnetic properties that can change its alignment when it is exposed to a magnetic field. That magnetic field comes in the form of the hard disks read/write heads. It is the change in alignment of this magnetic media that enables data to be stored on the hard disk.

As I said earlier, a read/write head does just that: it reads and writes. There is usually one head per surface of the platters top and bottom. That means that there are usually twice as many heads as platters. However, this is not always the case. Sometimes the top- and bottom-most surfaces do not have heads.

The head moves across the platters that are spinning several thousand times a minute at least 60 times a second!. The gap between the head and platter is smaller that a human hair, smaller than a particle of smoke. For this reason, hard disks are manufactured and repaired in rooms where the number of particles in the air is fewer than 100 particles per cubic meter.

Because of this very small gap and the high speeds in which the platters are rotating, if the head comes into contact with the surface of a platter, the result is aptly named a head crash. More than likely this will cause some physical damage to your hard disk. Imagine burying

your face into an asphalt street going even only 20 MPH!

The heads move in and out across the platters by the older stepping motor or the new, more efficient voice-coil motor. Stepping motors rotate and monitor their movement based on notches or indentations. Voice-coil motors operate on the same principle as a stereo speaker. A magnet inside the speaker causes the speaker cone to move in time to the music or with the voice. Because there are no notches to determine movement, one surface of the platters is marked with special signals. Because the head above this surface has no write capabilities, this surface cannot be used for any other purpose.

The voice-coil motor enables finer control and is not subject to the problems of heat expanding the disk because the marks are expanded as well. Another fringe benefit is that because the voice-coil operates on electricity, once power is removed, the disk moves back to its starting position because it no longer is resisting a "retaining" spring. This is called "automatic head parking."

Each surface of the platter is divided into narrow, concentric circles called *tracks*. Track 0 is the outermost track and the highest numbered track is the track closest to the central spindle. A *cylinder* is the set of all tracks with the same number. So all of the 5th tracks from each side of every platter in the disk is known as cylinder 5. As the number of cylinders is the same as the number of tracks, you often see disk geometries described in terms of cylinders. Each track is divided into *sectors*. A sector is the smallest unit of data that can be written to or read from a hard disk and it is also the disk's block size. A common sector size is 512 bytes and the sector size was set when the disk was formatted, usually when the disk is manufactured.

Since data is stored physically on the disk in concentric rings, the head does not spiral in like a phonograph record but rather moves in and out across the rings the tracks. Because the heads move in unison across the surface of their respective platters, data are usually stored not in consecutive tracks but rather in the tracks that are positioned directly above or below them. Therefore, hard disks read from successive tracks on the same cylinder and not the same surface.

Think of it this way. As the disk is spinning under the head, it is busy reading data. If it needs to read more data than what fits on a single track, it has to get it from a different track. Assume data were read from consecutive tracks. When the disk finished reading from one track, it would have to move in or out to the next track before it could continue. Because tracks are rings and the end is the beginning, the delay in moving out or in one track causes the beginning of the next track to spin past the position of the head before the disk can start reading it. Therefore, the disk must wait until the beginning comes around again. Granted, you could stager the start of each track, but this makes seeking a particular spot much more difficult.

Let's now look at when data are read from consecutive tracks that is, one complete cylinder is read before it goes on. Once the disk has read the entire contents of a track and has reached the end, the beginning of the track just below it is just now spinning under the head. Therefore, by switching the head it is reading from, the disk can begin to read or write as though nothing was different. No movement must take place and the reads occur much faster.

Each track is broken down into smaller chunks called sectors. The number of sectors into which each track is divided is called sectors per track, or sectors/track. Although any value is possible, common values for sectors/track are 17, 24, 32, and 64. These are shown graphically in Figure 0-12.

Each sector contains 512 bytes of data. However, each sector can contain up to 571 bytes of information. Each sector contains information that indicates the start and end of the sector, which is only ever changed by a low-level format. In addition, space is reserved for a checksum contained in the data portion of the sector. If the calculated checksum does not match the checksum in this field, the disk will report an error.

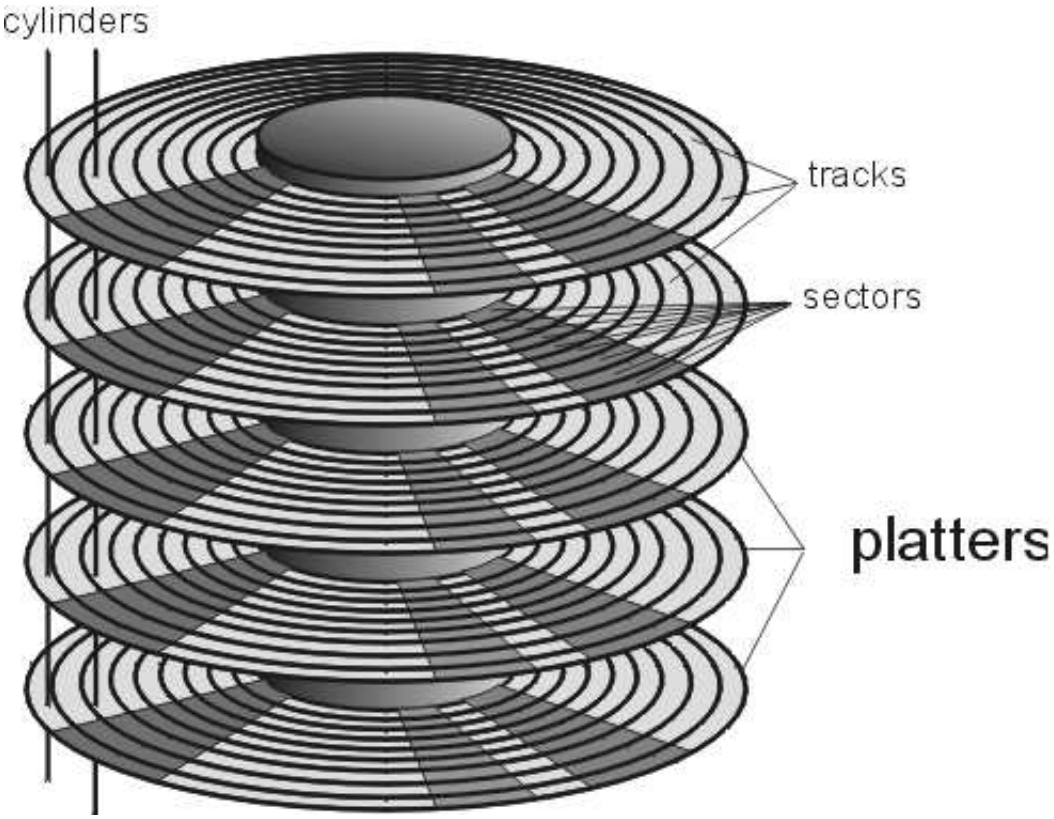


Image - Logical Components of a Hard Disk

This difference between the total number of bytes per sector and the actual amount of data has been cause for a fair amount of grief. For example, in trying to sell you a hard disk, the salesperson might praise the tremendous amount of space that the hard disk has. You might be amazed at the low cost of a 1G drive.

There are two things to watch out for. Computers count in twos, humans count in tens. Despite what the salesperson wants you to believe or believes himself, a hard disk with 1 billion bytes is *not* a 1 gigabyte drive it is only 10^9 bytes. One gigabyte means 2^{30} bytes. A hard disk with 10^9 1 billion is only about 950 megabytes. This is five percent smaller!

The next thing is that seller will often state is the *unformatted* storage capacity of a drive. This is the number that you would get if you multiplied all the sectors on the disk by 571 see the preceding discussion. Therefore, the unformatted size is irrelevant to almost all users. Typical

formatted MFM drives give the user 85 percent of the unformatted size, and RLL drives give the user about 89 percent. MFM and RLL are formatting standards, the specifics of which are beyond the scope of this book.

This brings up an interesting question. If the manufacturer is telling us the unformatted size and the formatted size is *about* 85 percent for MFM and 89 percent for SCSI /IDE using RLL, how can I figure out how much usable space there really is? Elementary, my dear Watson: Its a matter of multiplication.

Lets start at the beginning. Normally when you get a hard disk, it comes with reference material that indicates how many cylinders, heads, and sectors per track there are among other things. The set of all tracks at the same distance from the spindle is a cylinder. The number of cylinders is therefore simply the number of tracks because a track is on one surface and a cylinder is all tracks at the same distance. Because you can only use those surfaces that have a head associated with them, you can calculate the number of total tracks by multiplying the number of cylinders by the number of heads. In other words, take the number of tracks on a surface and multiply it by the number of surfaces. This gives you the total number of tracks.

From our discussion of tracks, you know that each track is divided into a specific number of sectors. To find the total number of sectors, simply multiply the number of total tracks that we calculated above by the sectors per track. Once you have the total number of sectors, multiply this by 512 the number of bytes *of data* in a sector. This give us the total number of *bytes* on the hard disk. To figure out how may megabytes this is, simply divide this number by 1,048,576 1024 x 1024 = 1MB.

Hard disks can be further subdivided into *partitions*. A partition is a large group of sectors allocated for a particular purpose. Partitioning a disk allows the disk to be used by several operating system or for several purposes. A lot of Linux systems have a single disk with three partitions; one containing a DOS filesystem, another an EXT2 filesystem and a third for the swap partition. The partitions of a hard disk are described by a partition table; each entry describing where the partition starts and ends in terms of heads, sectors and cylinder numbers. For DOS formatted disks, those formatted by fdisk, there are four primary disk partitions. Not all four entries in the partition table have to be used. There are three types of partition supported by fdisk, primary, extended and logical.Extended partitions are not real partitions at all, they contain any number of logical parititions. Extended and logical partitions were invented as a way around the limit of four primary partitions. The following is the output from fdisk for a disk containing two primary partitions:

```
Disk /dev/sda: 64 heads, 32 sectors, 510 cylinders
Units = cylinders of 2048 * 512 bytes

   Device Boot   Begin    Start      End   Blocks   Id  System
/dev/sda1            1        1      478   489456   83  Linux native
/dev/sda2          479      479      510    32768   82  Linux swap

Expert command m for help: p

Disk /dev/sda: 64 heads, 32 sectors, 510 cylinders
```

```
Nr AF  Hd Sec  Cyl  Hd Sec  Cyl   Start    Size ID
 1 00   1   1    0  63  32  477      32  978912 83
 2 00   0   1  478  63  32  509  978944   65536 82
 3 00   0   0    0   0   0    0       0       0 00
 4 00   0   0    0   0   0    0       0       0 00
```

This shows that the first partition starts at cylinder or track 0, head 1 and sector 1 and extends to include cylinder 477, sector 32 and head 63. As there are 32 sectors in a track and 64 read/write heads, this partition is a whole number of cylinders in size. fdisk alligns partitions on cylinder boundaries by default. It starts at the outermost cylinder 0 and extends inwards, towards the spindle, for 478 cylinders. The second partition, the swap partition, starts at the next cylinder 478 and extends to the innermost cylinder of the disk.

All PC-based operating systems need to break down the hard disk into partitions. In fact, all operating systems I can thing of that use the same kind of hard disks need to create partitions. A partition can be any size, from just a couple of megabytes to the entire disk. Each partition is defined in a *partition table* that appears at the very beginning of the disk. This partition table contains information about the kind of partition it is, where it starts, and where it ends. This table is the same whether you have a DOS-based PC, UNIX, or both.

Because the table is the same for DOS and UNIX, there can be only four partitions total because there are four entries in the table. DOS gets around this by creating *logical* partitions within one physical partition. This is a characteristic of DOS, *not* the partition table. Both DOS and UNIX must first partition the drive before installing the operating system and provide the mechanism during the installation process in the form of the fdisk program. Although their appearances are very different, the DOS and Linux fdisk commands perform the same function.

When you run the Linux fdisk utility, the values you see and input are all in tracks. To figure out how big each fdisk partition is, simply multiply that value by 512 and by the number of sectors per track. Remember that each sector holds 512 bytes of data.

A disk is usually described by its geometry, the number of cylinders, heads and sectors. For example, at boot time Linux describes one of my IDE disks as:

```
hdb: Conner Peripherals 540MB - CFS540A, 516MB w/64kB Cache, CHS=1050/16/63
```

This means that it has 1050 cylinders tracks, 16 heads 8 platters and 63 sectors per track. With a sector, or block, size of 512 bytes this gives the disk a storage capacity of 529200 bytes. This does not match the disk's stated capacity of 516 Mbytes as some of the sectors are used for disk partitioning information. Some disks automatically find bad sectors and re-index the disk to work around them.

To physically connect itself to the rest of the computer, the hard disk has five choices: ST506/412, , ESDI, SCSI, IDE, and the newer Enhanced IDE EIDE. However, the interface the operating system sees for ST506/412 and IDE are identical, and there is no special option for an IDE drive. At the hardware level, though, there are some differences that I need to cover for completeness.

To be quite honest, only ESDI and ST506/412 are really disk interfaces. SCSI and IDE are referred to as "system-level interfaces" and they incorporate ESDI into the circuitry physically located on the drive.

The ST506/412 was developed by Seagate Technologies hence the ST for its ST506 hard disk, which had a whopping 5Mb formatted capacity. This was in 1980 when 360K was a *big* floppy. Seagate later used the same interface in their ST412, which doubled the drive capacity which is still less hard disk space than most people RAM. Oh well. Other drive manufacturers decided to incorporate this technology, and over the years, it has become a standard. One of its major drawbacks is that is 15-year-old technology, which can no longer compete with the demands of todays hard disk users.

In 1983, the Maxtor Corporation established the Enhanced Small Device Interface ESDI standard. The enhancements ESDI provided offered higher reliability because Maxtor had built the encoder/decoder directly into the drive and therefore reduced the noise, high transfer rates, and the ability to get drive parameters directly from this disk. This meant that users no longer had to run the computer setup routines to tell the CMOS what kind of hard disk it had.

One drawback that I have found with ESDI drives is the physical connection between the controller and the drive itself. Two cables were needed: a 34-pin control cable and a 24-pin data cable. Although the cables are different sizes and can’t easily be confused, the separation of control and data is something of which I was never a big fan. The connectors on the drive itself were usually split into two unequal halves. In the connector on the cable, a small piece of plastic called a key prevented the connector from being inserted improperly. Even if the key was missing, you could still tell which end was which because the pins on the hard disk were labeled and the first line on the cable had a colored stripe down its side. This may not always be the case, but I have *never* seen a cable that wasn’t colored like this.

Another drawback that I have found is that the physical location on the cable determines which drive is which. The primary drive is located at the end of the cable, and the secondary drive is in the middle. The other issue is the number of cables: ESDI drives require three separate cables. Each drive has its own data cable and the drives share a common control cable.

Although originally introduced as the interface for hard cards hard disks directly attached to expansion cards, the IDE integrated drive electronics interface has grown in popularity to the point where it is perhaps the most commonly used hard-disk interface today rapidly being replaced by SCSI. As its name implies, the controller electronics are *integrated* onto the hard disk itself. The connection to the motherboard is made through a relatively small adapter, commonly referred to as a "paddle board." From here, a single cable attaches two hard disks in a daisy chain, which is similar to the way floppy drives are connected, and often, IDE controllers have connectors and control electronics for floppy drives as well.

IDE drives often play tricks on systems by presenting a different face to the outside world than is actually on the disk. For example, because IDE drives are already pre-formatted when they reach you, they can have more physical sectors in the outer tracks, thereby increasing the overall amount of space on the disk that can be used for storage. When a request is made to read a particular block of data on the drive, the IDE electronics translate this to the actual

physical location.

Because IDE drives come pre-formatted, you should never low-level format an IDE drive unless you are specifically permitted to do so by the manufacturer. You could potentially wipe out the entire drive to the point at which it must be returned to the factory for "repair." Certain drive manufacturers, such as Maxtor, provide low-level format routines that accurately and *safely* low-level format your drive. Most vendors that I am aware of today simply "zero" out the data blocks when doing a low-level format. However, *don't* take my word for it! Check the vendor.

Only two IDE drives can be connected with a single cable, but there is nothing special about the position of the drive on the cable. Therefore, the system needs some other way of determining which drive is which. This is done with the drives themselves. Most commonly, you will find a jumper or set up jumpers used to determine if the drive is the master, slave or master-only/single-drive. On much older system, you could only have a single IDE controller, which meant only two drive. Today, it is common to find two IDE controllers build onto the mother board and some system allow you to add extra IDE controllers, thereby increasing the number of IDE drives even further.

The next great advance in hard disk technology was SCSI. SCSI is not a disk interface, but rather a semi-independent bus. More than just hard disks can be attached to a SCSI-Bus. Because of its complex nature and the fact that it can support such a wide range of devices, I talked in more detail about SCSI earlier in this chapter. However, a few specific SCSI issues relate to hard disks in general and the interaction between SCSI and other types of drives.

The thing to note is that the BIOS inside the PC knows nothing about SCSI. Whether this is an oversight or intentional, I don't know. The SCSI spec is more than 10 years old, so there has been plenty of time to include it. Because the BIOS is fairly standard from machine to machine, including SCSI support might create problems for backward compatibility.

On the other hand, the BIOS is for DOS. DOS makes BIOS calls. To be able to access all the possible SCSI devices through the BIOS, it must be several times larger. Therefore, every PC-based operating system needs to have extra drivers to be able to access SCSI devices.

Because the BIOS does not understand about SCSI, you have to trick the PCs BIOS a little to boot from a SCSI device. By telling the PCs BIOS that no drives are installed as either C: or D:, you force it to quit before it looks for any of the other types. Once it quits, the BIOS on the SCSI host adapter has a chance to run.

The SCSI host adapter obviously knows how to boot from a SCSI hard disk and does so wonderfully. This is assuming that you enabled the BIOS on the host adapter. If not, you're hosed.

There is also the flip side of the coin. The official doctrine says that if you have a non-SCSI boot driver, you *have* to disable the SCSI BIOS because this can cause problems. However, I know people who have IDE boot drives and still leave the SCSI BIOS enabled. Linux normally reacts as though the SCSI BIOS were not enabled, so, what do to? I suggest that you see what works. I can only add that if you have more than one host adapter , only one should have the BIOS enabled.

Another thing is that once the kernel boots from a SCSI device, you loose access to other kinds of drives. Just because it doesn't boot from the IDE or whatever, does this mean you cannot access it at all? Unfortunately, yes. This is simply the way the kernel is designed. Once the kernel has determined that it has booted off a SCSI hard disk, it can no longer access a non-SCSI hard disk.

The newest member of the hard disk family is Enhanced IDE, or EIDE. The most important aspect of this new hard disk interface is its ability to access more than 504 megabytes. This limitation is because the IDE interface can access only 1,024 cylinders, 16 heads, and 63 sectors per track. If you multiply this out using the formula I gave you earlier, you get 504Mb.

EIDE also has other advantages such as higher transfer rates, ability to connect more than just two hard disks, and attach more than just hard disks. One drawback the EIDE had at the beginning was part of its very nature. To overcome the hard disk size limit that DOS had, EIDE drives employ a method called *logical block addressing* LBA.

The idea behind LBA is that is that the systems BIOS would "rearrange" the drive geometry so that drives larger than 528Mb could still boot. Because Linux does not use the BIOS to access the hard disk, the fact that the BIOS could handle the EIDE drivemeant nothing. New drivers needed to be added to accountfor this.

More and more you find the EIDE controllers built directly onto the motherboard. On the one hand this is a good thing, since you do not need to use a expansion card slot for the controller. However, you need to be careful where it is located. I have had a few motherboards, where the IDE controller was stuck between the PCI and ISA slots. This made it extremely difficult to access the pins without removing either of the cards in either of the PCI or ISA slot sometimes both.

Although this might not seem like a big deal, but it may become one the next time you do anything with that machine. It is not uncommon when adding a harddisk or something else to the system that you accidentally pull on the harddisk cable. All you need to do is pull it no more than quarter of an inch before some plugs are no longer connected to the pins. Because this connection is almost impossible to see, you don't notice that the cable has come loose. When you reboot the machine nothing works as the system is getting signals through only some of the lines. If the pins for the IDE controller are out in the open, you may still pull on the cable, but it is easier to see and far easier to fix.

There is much more to choosing the right harddisk that its size. Although size determines how much data you can store, it tells you nothing about the speed at which you can access that data. How quickly you can access your data is the true measure of performance.

Unfortunately, there is no one absolute measure of harddisk performance. The reason is simply that data access occurs in some many different ways, it is often difficult for even the experienced administrator to judge which drive is better. However, there are several different characteristics of harddisks, which, when viewed together give you a good idea of the overall performance of a drive.

One character that is often quoted is the seek time. This refers to the time need to move the read/write head between tracks. If the data is not on the same track, it could mean moving the head a couple thousand tracks in either direction. Movement from one track to the next adjacent one might take only 2 ms, whereas moving the entire diameter of the drive might take 20ms.

So which one do you use? Typically, neither. When access times are specified, you normally see the average seek time. This is measured as the average time between randomly located tracks on the disk. Typical rages at the time of this writing are between 8ms and 14ms. The problem is that disk access is often usually? not random. Depending on how you work, you read large a number of blocks at once, such as to load a WordPerfect file. Therefore, average seek time does not reflect access of large pieces of data.

Once the head has moved to the track you need, you are not necessarily read to work. You need to wait until the right block is under the read/write head. The time the drive takes to reach that point is called rotational latency. The faster the drive speeds, the more quickly the block is underneath the head. Therefore, rotational latency is directly related to the rotational speed rpm of your drive.

By increasing the rotational speed of a drive you obviously decrease the time the drive has to wait. The fastest drives as I am writing this spin at least 7200 times per minutes, which means that have an average rotational latency is about 4.2 ms.

You can also decrease the rotational latency by staggering the start of the start of each track. This is especially effective when doing sequential reads across tracks. If the start of all tracks were at the same place, the head would move to the new track and the start of the track would have already spin out from underneath. If the tracks are staggered, the head has to wait less time less rotational latency until the start of the track is underneath.

Think back to our discussion of harddisks and the concept of a cylinder. This is all of the tracks at the same distance from the spindle. To physically move heads from one track to another takes more time than simple switch which head you are using. However, because switch heads does not occur instantaneously, there is a certain amount of rotational latency. Therefore, the start of each track is staggered as one moves up and down the cylinder, as well as across the cylinders.

By decreasing the rotational latency, we increase the speed at which the head reaches the right position. Once we are there, we can begin reading, this is the average access time. This, too, is measured in milliseconds.

Still, this is not the complete measure of the performance of our drive. Although it is nice that the drive can quickly begin to read, this does not necessarily mean that it will read the data fast. The faster the harddisk can read the data, the faster your WordPerfect file is loaded. This is due to the transfer rate. This is normally measured in megabytes per second.

However, the actual transfer is not necessarily what the harddisk manufacturer says it is. They may have given the transfer rate in terms of the maximum or average sustained transfer rate. This is important to understand. If you have one huge 200Mb file that you are reading on a new drive, the entire drive might be contiguous. Therefore, there is very little movement of

the heads as the file is read. This would obviously increase the average transfer rate. However, if you have two hundred 1 Mb files spreads out all over the disk, you will definitely notice a lower transfer rate.

In addition, this is another case of the chain being as strong as its weakest link. The actual transfer rate dependant on other factors, as well. A slow harddisk controller or slow system bus can make a fast harddisk display bad performance.

Another aspect is how much of the date is being re-read. For example, if you read the same one Mb file two hundred times, the head won't move much. This is not a bad thing, as data is often read repeatedly. Harddisk manufacturers are aware of this and therefore will add caches to the harddisk to improve performance. Data that is read from the harddisk can be stored in the cache so if it is needed again, it can be accessed more quickly than if it must be first read from the drive. Data that is written, may also be needed again, so it too can be re-read from the cache.

This is also called a cache buffer, because it also serves to buffer the data. Sometimes the harddisk cannot keep up with the CPU. It may be the disk is writing someone as new data comes in. Rather than making the CPU wait, the data is written to the cache, which the harddisk can read when it can. Other times, the CPU is doing something else as the data is from the harddisk is ready. The harddisk can write it to the buffer and the CPU can take it when it can.

Finally, there is data throughput. This is a measure of the total amount of data the CPU can access in a given amount of time. Since the data is going through the harddisk controller and through the system bus, this may not be a good measure of performance of the drive itself. However, if the other components can process the data as quickly as the drive can provide it, it is a good measure of the complete system.

This is why you will see references to Seagate drives on the Adaptec web site. Adaptec understands the relationship between the components in your system. Therefore, they suggest drives that can keep up with the other components such as the appropriate ones from Seagate.

Another aspect of the administration costs that a lot of people do not think about is the drive designation. Although calling a harddisk "WhirlWind" or "Falcon" might be pleasing to the marketing people or the IT manager who has no clue about the technical details. However, the administrator is not interested in what name it has but rather its characteristics it has. If it takes a long time to figure out the characteristics, the total cost of owner ship has increased.

How often have you had to wade through pages and pages on a company's Internet site to figure out how big a particular model was?. Although many most? companies have a 1:1 relationship between the model designation and the characteristics, you have to first figure out the scheme, as often it is not posted anywhere on the site.

This is one reason why I keep coming back to Seagate. Without thinking I can come up with the model number or something very close. The general format is:

ST<F><MB><INT>

Where:
<F> = Form factor, such as 3", 3" half-high, 5", etc.
<B> = Approximate size in megabytes.
<INT> = Interface.

So, looking at my drive, which is a ST39140A, I can quickly tell that it is a form factor 3 3" drive and 1" high, it has approximately 9140 MB and an ATA interface. Granted some of the abbreviations used for the interface take a little to get used to. However, the naming scheme is consistent and very easy to figure out.

As with other hardware, your choice of harddisk is also guided by the reputation of the company. This applies not only to what you have heard, but also your own personal experiences. Often it is more than just having heard or reading that a particular manufacturer is bad, but rather an issue of being "sure." This is why I will never buy an IBM harddisk again. All three I have bought were defective. Although other people have claimed not to have problems with them, I do not want to risk my data on them. Three times is too much of a coincidence and I would not feel safe if I installed an IBM harddisk on any of my machines, nor would I have a clear conscience if I installed it in a customer's machine.

On the other hand, I have had a proportionally large number of Seagate drives since I first started working with computers. None of which have ever given me problems. So far, all of my Seagate drives have been replaced with larger drives, not because they have failed, but they have grown, too small. There are only so many bays in a computer case and filling them up with small drives is not worth it. Instead, I got larger drives.

In addition to the size and speed of your drive, one important consideration is the interface to the harddisk. Typically, SCSI harddisks are more expensive than ATA drives, even if you ignore the extra costs for the SCSI host adapter. Even if you want to ignore the extra costs to acquire the drive, you need to consider the costs to install and manage the host adapter, the performance increase you get with SCSI is negligible for work stations. Generally, you do not need the extra throughput that SCSI can provide.

In most cases, space will be an issue. Although you need just a few hundred megabytes for the operating system, you are getting larger and larger applications, with dozens of components which quickly fill up space on your harddisk. Buying and installing a new ATA harddisk is generally simpler than adding a SCSI harddisk particularly if your first harddisk is ATA. In addition, on newer system you can have up to four ATA devices, including CD-ROM drives, which is generally sufficient for a workstations, as well as mobile users.

On the other hand, if you are in an environment where you need more than four device or need devices that do not support ATA, then you will have to go with SCSI. In addition, SCSI is basically a must when talking about your server. Size isn't an issue as what is available is generally the same for ATA and SCSI. The key difference is performance. This is particularly important in a multi-user environment.

Let's take the Seagate Cheetah as an example. As of this writing it is the fastest available on the market with10,000 RPM. It has a maximum internal transfer rate of 306Mbits/s, which means it is even faster than the 80Mbits/s of the Ultra SCSI interface. This is a result of an average seek time of 6 milliseconds and 2.99 average latency. This means the average access time is under 9 milliseconds. To compensate, the Cheetah series has default buffer size of 1Mb. In addition, the throughput is too high to use anything other than SCSI or Fibre Channel, so it is not available with an ATA interface.

There are also a few other reasons why something like the Cheetah is the perfect solutions for a server. First, it supports up to 15 devices on a single wide SCSI bus. Using the Fibre Channel versions, you can get up to 126 devices, which are also hot swappable.

Another thing to consider is the maintenance and administration. Low-end Medalist drives have an expected mean-time between failures MTBF of 400,000 hours. Which is about 45 years. The MTBF for the Cheetah is approximately 1,000,000 hours or over 100 years. No wonder I haven't ever had a harddisk crash.

The Seagate drives also do something else to reduce maintenance and administration costs. First, there is something Seagate calls SeaShield and is something other harddisk manufacturers should adopt. This is simply a protective cover around the electronics that are exposed on other harddisks. This protects the electronics from static electrical discharge, as well as damage caused by bumping the drive against something. In addition, this cover provides the perfect space for installation instructions, like the jumper settings. There is no need to go hunting around for the data sheet, which often isn't supplied with the drive. Talk about saving administration costs!

Some of you might be saying that names like Cheetah go against my desire to have understandable model names. My answer is that the opposite is true. As of this writing Seagate has four primary series: Medalist, Medalist Pro, Barracuda and Cheetah. This simply tells the rotation rate, which is 5400, 7200, 7200 and 10,000 RPM respectively. The Medalist is Ultra ATA. The Medalist Pro is either ATA or SCSI. The Barracuda and Cheetah are either SCSI or Fibre Channel. Okay, this requires you to use your brain a little, but it is far easier than many other vendors.

8.6.1 RAID

RAID is an acronym for Redundant Array of Inexpensive Disks. Originally, the idea was that you would get better performance and reliability from several, less expensive drives linked together as you would from a single, more expensive drive. The key change in the entire concept is that hard disk prices have dropped so dramatically that RAID is no longer concerned with inexpensive drives. So much so, that the I in RAID is often interpreted as meaning "Intelligent", rather than "Inexpensive."

In the original paper that defined RAID, there were five levels. Since that paper was written, the concept has been expanded and revised. In some cases, characteristics of the original levels are combined to form new levels.

Two concepts are key to understanding RAID. These are redundancy and parity. The concept of parity is no different than that used in serial communication, except for the fact that the parity in a RAID system can be used to not only detect errors, but correct them. This is because more than just a single bit is used per byte of data. The parity information is stored on a drive separate from the data. When an error is detected, the information is used from the good drives, plus the parity information to correct the error. It is also possible to have an entire drive fail completely and still be able to continue working. Usually the drive can be replaced and the information on it rebuilt even while the system is running. Redundancy is the idea that all information is duplicated. If you have a system where one disks is an exact copy of another, one disk is redundant for the other.

A striped array is also referred to as RAID 0 or RAID Level 0. Here, portions of the data are written to and read from multiple disks in parallel. This greatly increases the speed at which data can be accessed. This is because half of the data is being read or written by each hard disk, which cuts the access time almost in half. The amount of data that is written to a single disk is referred to as the stripe width. For example, if single blocks are written to each disk, then the stripe width would be a block

This type of virtual disk provides increased performance since data is being read from multiple disks simultaneously. Since there is no parity to update when data is written, this is faster than system using parity. However, the drawback is that there is no redundancy. If one disk goes out, then data is probably lost. Such a system is more suited for organizations where speed is more important than reliability.

Keep in mind that data is written to all the physical drives each time data is written to the logical disk. Therefore, the pieces must all be the same size. For example, you could not have one piece that was 500 MB and a second piece that was only 400 Mb. (Where would the other 100 be written?) Here again, the total amount of space available is the sum of all the pieces.

Disk mirroring (also referred to as RAID 1) is where data from the first drive is duplicated on the second drive. When data is written to the primary drive, it is automatically written to the secondary drive as well. Although this slows things down a bit when data is written, when data is read it can be read from either disk, thus increasing performance. Mirrored systems are best employed where there is a large database application

Availability of the data (transaction speed and reliability) is more important than storage efficiency. Another consideration is the speed of the system. Since it takes longer than normal to write data, mirrored systems are better suited to database applications where queries are more common than updates.

The term used for RAID 4 is a block interleaved undistributed parity array. Like RAID 0, RAID 4 is also based on striping, but redundancy is built in with parity information written to a separate drive. The term "undistributed" is used since a single drive is used to store the parity information. If one drive fails (or even a portion of the drive), the missing data can be created using the information on the parity disk. It is possible to continue working even with one drive inoperable since the parity drive is used on the fly to recreate the data. Even data written to the disk is still valid since the parity information is updated as well. This is not intended as a means of running your system indefinitely with a drive missing, but rather it

gives you the chance to stop your system gracefully.

RAID 5 takes this one step further and distributes the parity information to all drives. For example, the parity drive for block 1 might be drive 5 but the parity drive for block 2 is drive 4. With RAID 4, the single parity drive was accessed on every single data write, which decreased overall performance. Since data and parity and interspersed on a RAID 5 system, no single drive is overburdened. In both cases, the parity information is generated during the write and should the drive go out, the missing data can be recreated. Here again, you can recreated the data while the system is running, if a hot spare is used. Figure - Raid 5

As I mentioned before, some of the characteristics can be combined. For example, it is not uncommon to have stripped arrays mirrored as well. This provides the speed of a striped array with redundancy of a mirrored array, without the expense necessary to implement RAID 5. Such a system would probably be referred to as RAID 10 (RAID 1 plus RAID 0).

Regardless of how long your drives are supposed to last, they will eventually fail. The question is when. On a server, a crashed harddisk means that many if not all of your employees are unable to work until the drive is replaced. However, there are ways of limiting the effects the crash has in a couple ways. First, you can keep the system from going down unexpectedly. Second, you can protect the data already on the drive.

The key issue with RAID is the mechanisms the system uses to portray the multiple drives as single one. The two solutions are quite simply hardware and software. With hardware RAID; the SCSI host adapter does all of the work. Basically, the operating system does not even see that there are multiple drives. Therefore, you can use hardware RAID with operating systems that do not have any support on their own.

On the other hand software RAID is less expensive. Linux comes included with software, so there is no additional cost. However, to me this is no real advantage as the initial hardware costs are a small fraction of the total cost of running the system. Maintenance and support play a much larger roll, so these ought to be considered before the cost of the actual hardware. In it's Annual Disaster Impact Research, Microsoft reports that on the average a downed server costs at least $10,000 per hour. Think about how many RAID controllers you can buy with that money.

In addition, the total cost of ownership also includes user productivity. Should a drive fail, performance degrades faster with a software solution than with a hardware solution.

Let's take an Adaptec AA-133SA RAID controller as an example. At the time of this writing it is one of the top end models and provides three Ultra SCSI channels, which means you could theoretically connect 45 devices to this single host adapter. Since each of the channels is Ultra SCSI, you have a maximum throughput of 120Mbit/s. At the other end of the spectrum is the Adaptec AAA-131CA, which is designed more for high-end workstations, as it only supports mirroring and striping.

One thing to note is that the Adaptec RAID host adapters do not just provide the interface, which makes multiple drives appear as one. Instead, they all include a coprocessor, which increases the performance of the drives considerably.

However, providing data faster and redundancy in not all of it, Adaptec RAID controllers also have the ability to detect errors and in some cases correct errors on the hard disk. Many SCSI systems can already detect single-bit errors. However, using the parity information from the drives, the Adaptec RAID controllers can correct these single-bit errors. In addition, the Adaptec RAID controllers can also detect 4-bit errors.

You need to also keep in mind the fact that maintenance and administration are more costly than the initial hardware. Even though you have a RAID 5 array, you still need to replace the drive should it fail. This brings up two important aspects.

First, how well can your system detect the fact that a drive has failed? Whatever mechanisms you chose must be in a position to immediately notify the administrators should a drive fail.

The second aspect returns to the fact that maintenance and administration costs are much higher than the cost of the initial hardware. If the hardware makes replacing the drive difficult, you increase your downtime and therefore the maintenance costs increase. Adaptec has addressed this issue by allowing you to "hot swap" your drives. This means you can replace the defective drive on a running system, without have to shutdown the operating system.

Note that this also requires that the case containing the RAID drive be accessible. If your drives are in the same case as the CPU (such as traditional tower cases), you often have difficulty getting to the drives. Removing one while the system is running is not practical. The solution is an external case, which is specifically designed for RAID.

Often you can configure the SCSI ID of the drive with dials on the cases itself and sometimes the position in the case determines the SCSI ID. Typically, the drives are mounted onto rails, which slide into the case. Should one fail, you simple slide it out and replace it with the new drive.

Protecting your data and being able to replace the drive is just a start. The next level up is what is referred to as "hot spares." Here, you have additional drives already installed that are simply waiting for another to break down. As soon as a failure is detected, the RAID card replaces the failed drive with a spare drive, simply reconfigures the array to reflect the new drive and the failure is reported to the administrator. Keep in mind that this must be completely supported in the hardware.

If you have an I/O-bound application, a failed drive decreases the performance. Instead of just delivering the data, your RAID array must calculate the missing data using the parity information, which means it has a slower response time in delivering the data. The degraded performance continues until you replace the drive. With a hot spare, the RAID array is rebuilding it self as it is delivering data. Although performance is obviously degraded, it is to a lesser extent than having to swap the drives manually.

If you have a CPU-bound application, you obtain substantial increases in performance over software solutions. If a drive fails, the operating system needs to perform the parity calculations in order to reconstruct the data. This keeps the CPU from doing the other tasks and performance is degraded. Because the Adaptec RAID controller does all of the work of reconstructing the data, the CPU doesn’t even notice it. In fact, even while the system is

running normally, the RAID controller is doing the appropriate calculations, so there is no performance lost here either.

In addition, the Adaptec RAID controllers can be configured to set the priority of performance versus availability. If performance is given a high priority, it will take longer to restore the data. If availability is given the higher priority, performance suffers. Either is valid, depending on your situation. It is also possible to give each the same priority.

Because the new drive contains no data, it must take the time to re-create the data using the parity information and the data from the other drives. During this time performance will suffer as the system is working to restore the data on the failed drive.

Redundancy like this can (and therefore the safety of your data) be increased further by having redundant RAID 5 arrays. For example, you could mirror the entire RAID set. This is often referred to as RAID 51, as it is a combination of RAID 5 and RAID 1, although RAID 51 was not defined in the originally RAID paper. Basically, this is a RAID array which is mirrored. Should a drive fail, not only can the data be recovered from the parity information, but it can also be copied from its mirror. You might also create a RAID 15 array. This is a RAID 5 array, which is made up of mirror sets.

8.6.2 Serial ATA

by James Pyles

Serial ATA Advanced Technology Attachment, drives began arriving on retail shelves in November of 2002. They use a completely new interface between the hard drive and the motherboard and are quite likely to replace the standard ATA interface in the next few years.

So just what is the big deal about Serial ATA drives?

To answer that, we need to delve into a bit of history. Since 1989, IDE drives have been the standard hard drive interface. Except for enlarging the pipeline from 33 to 100/133 MB/sec with the invention of ATA in 1998, nothing has changed about the interface between the HDD and the motherboard. For almost 15 years an eternity in the world of computer technology, the standard connection between the hard drive and the motherboard in PCs has been the same parallel interface.

So how is a Serial ATA drive different?

Serial ATA or SATA drives are a new standard in HDD/motherboard interfaces and are touted as the next mainstream storage interface. SATA drives are fast with the first generation having a volume of 150 MB/sec compared to Parallel ATA PATA drive's 100 MB/sec. Generation two coming out in 2005 is predicted to go up to 300 MB/sec. This is also better than USB interfaces at 60 MB/sec and generation two will be better than current firewire volume of 200 MB/sec.

The current ATA drive's parallel interface will eventually become a throughput bottleneck, especially as drive densities continue to increase. In 2001, major vendors were able to put 40 GB of storage on a hard drive *per platter*. By the end of 2003, it will likely be 80 GB and

future densities are predicted at 120 GB per platter. This impacts throughput because the more density, the more pressure to push data faster across the interface.

SATA interfaces can make use of cables twice as long as IDE cabling at a full one meter. The cable is thinner, .25 inches compared to a two inch IDE ribbon, and plugs into the SATA drive in a manner similar to a network cable.

SATA drives are also "hot swappable" meaning no more reboots, they are quiet thanks to fluid bearings, but not quite silent, use less power overall, and come with either a 2 or 8 MB cache.

Now for the downside

Currently, SATA drives have not yet entered the mainstream PC market. This is predicted to happen by 2004 when it is believed that major vendors like Dell and HP will start offering SATA drives in their upper end systems. If you want to use a SATA drive in your PC, you will not only have to purchase the drive, power supply and cable, but a PCI SATA adapter card. Your motherboard must also be SATA capable. For SATA to enter the mainstream market, Intel will need to integrate this standard into their chipsetâ¦planned by the middle or end of 2003.

The other downside: only one device can be used per port. This means that there is no RAID capacity for generation one SATA since you can't run multiple drives on the same channel.

SATA II

That's right, there's more. The SATA working group announced that SATA II will allow additional ports to support up to 15 drives. This will benefit mostly servers running multiple rack-mounted drives.

Other extensions announced by the group are native command queuing, giving hard drives the ability to prioritize data requests from multiple CPUs, doubling the bandwidth from 1.5 to 3 GB/sec, fallover switches for when high reliability is vital, and connections for external SATA II drives.

The speed and increased throughput of SATA drives are extremely important in the future of servers. While an end user is unlikely to notice the performance difference, the speed improvement will be important in the use of RAID to operate multiple drives.

Yeah, but how much does it cost?

I found a comparison of Maxtor SATA and PATA drives. ExtremeTech published a comparison between Maxtor's DiamondMax Plus 9 PATA and the Plus 9 SATA. Each drive has a formatted capacity of 200 GB, 66.67 per platter and a rotational speed of 7200 RPM. The PATA drive will set you back $210 USD while the equivalent SATA drive is a bit more at $275. Maxtor's SATA FAQ states that SATA drives will be typically a bit more than equivalent PATAs, throughout 2003 but as SATA drives become more common, the prices will reach parity. The SATA Plus 9 currently comes in capacities of 60GB, 80GB, 120GB, 160GB, and 200GB.

Not trying to favor Maxtor, I went shopping at Atacom.com for comparable Seagate drives and looked at the Seagate PATA 80GB ST380021A versus the SATA Barracuda V 80GB. The PATA drive costs $83.95 while the equivalent Barracuda ran $125.95.

What about drivers?

According to serialata.org, SATA supports legacy drivers for PATA drives. In other words, you can install a SATA drive on a motherboard with the appropriate PCI SATA adapter and use the OEM's existing parallel ATA drivers. Vendors will begin supplying bridges for parallel to serial conversion for legacy devices. This being said, Seagate's support page recommends that, if your operating system Windows 2000/XP in this case doesn't detect the SATA drive during installation, you may need to go to the vendor's website to acquire additional drivers.

Although many newer motherboards have SATA controllers, not all do and obviously older motherboards do not support SATA. If you have a motherboard that does not support SATA, all is not lost. You can buy extra SATA controller cards for between $20-$30 US.

Silicon Image is providing parallel to serial conversion for Linux. Their drivers are supplied by the Linux ATA development site.

SATA and Linux

In June 2003, Pogo Linux included three new Linux servers using Seagate SATA drives in its StorageWare line. They claim that these servers are the first to use SATA for Linux. The StorageWare S140, S212, and S316 are all powered by Intel Xeon processors with Hyper-Threading Technology.

The SATA drives are configured with a 3Ware 8500 Escalade SATA RAID controller in a RAID 5 array. According to Pogo Linux, this will enable the StorageWare servers to give high speed performance at a fraction of the cost of even the latest, high-end SCSI based servers. You might want to save up your pennies if you're considering a purchase as the prices are $3,499, $7,499, and 9,499 respectively for the above-listed servers. Pogo Linux also produces the Velocity-DXWorkstation which comes with SATA, IDE, or SCSI starting at $1,979.

None of this means you actually have to spend thousands of dollars to use SATA with Linux. You will, at this point, have to buy the drive, connectors, power supply and adapter and perhaps download the drivers. For a few hundred dollars, you'll have all the materials to enter the world of Serial ATA. Keep in mind that SATA drives are just now becoming available in mass quantities and you will still have to wait until PC OEMs come with SATA integrated. I mentioned above that this is predicted for 2004 but the Maxtor SATA FAQ page was particularly evasive. It's likely that negotiations between the major relevant vendors are, as of this writing, still in progress.

If you plan to use SATA and Linux on a home or small office system, you will likely have to play around with the installation and configuration. As with anything new, there is uncharted territory to explore.

8.7 Floppy Drives

A customer once called in to tech support about a system that would not boot. For some unknown reason, the system crashed and would no longer boot from the hard disk. It got to a particular point in the boot process and hung. Even an old copy of the kernel hung in the same way.

Fortunately, the customer had an emergency boot floppy that enabled him to boot and gain access to the hard disk. The customer stuck the floppy in the drive and pressed the reset button. After a few seconds, there were the normal messages the system presented at boot up. For the moment, things looked fine. Suddenly the messages stopped and I could hear over the phone how the floppy drive was straining. It finally came up with the dreaded "floppy read error." Rather than giving up, I decided to try it again. Same thing.

At that point I started to get concerned. The hard disk booted, but the kernel hung. The floppy booted, but somewhere in the middle of loading the kernel, there was a bad spot on the floppy. This was not a good thing.

The floppy disk was brand new and the customer had tested it out immediately after he had made it. The most logical thing that caused this problem was putting the floppy too close to a magnetic field. Nope! That wasn't it, either. The customer had been told to keep this floppy in a safe place, and that's what the customer did.

What was that safe place? The customer had tacked it to the bulletin board next to the monitor, not through the hub or at one of the corners, but right through the floppy itself. The customer had been careful not to stick the pin through the media access hole because he was told never to touch the floppy media itself.

In this section, I'm going to talk about floppy disks, lovingly referred to as floppies. They come in different sizes and shapes, but all floppies serve the same basic functions. Interaction with floppies can be a cause of great heartache for the unprepared, so, Im going to talk about what they are like physically, how they are accessed, and what kinds of problems you can have with them.

Although they hold substantially less data than hard disks, floppies appear and behave very much like hard disks. Like hard disks, floppies are broken down into sectors, tracks, and even cylinders. Like hard disks, the number of tracks tells us how many tracks are on a given surface. Therefore, a floppy described as 40 tracks (such as a 360Kb floppy) actually contains 80 tracks, or 40 cylinders.

Other common characteristics are the header and trailer of each sector, which results in 571 bytes per sector, 512 of those being data. Floppy disks almost universally use MFM data encoding.

Linux floppy drivers support a wide range of floppies: from the ancient 48 tracks per inch/8 sectors per track, 5.25" floppies to the newest 135 tracks per inch/36 sector per track, 3.5" floppies that can hold almost 3Mb of data. More commonly however, the floppy devices found on systems today are somewhere in between these two types.

Because they are as old as PCs themselves, floppies have changed little except for their size and the amount of data that can be stored on them. As a result, very few problems are encountered with floppies. One most common problem is that customers are unsure which floppy device goes to which type of drive. Sometimes customers do know the difference and try to save money by forcing the floppy to format in a density higher than it was designed for.

The truth of the matter is, you *can't* format floppies higher than you're supposed to, that is, higher than the manufacturer specifies. To some extent, you might get away with punching holes in single-sided floppies to make them double-sided. However, forcing a floppy to a format at a higher density (if it works) isn't worth risking your data.

To understand why this is so, I need to talk about the concept of coercivity, that is, how much energy (how strong the magnetic field) must be used to make a proper recording on a disk. Older floppies had a lower coercivity and therefore required a weaker magnetic field to hold the signal; that is, less energy was required to "coerce" them into a particular pattern.

This seems somewhat contradictory, but look at it another way. As densities increased, the magnetic particles got closer together and started to interfere with each other. The result was to make the particles weaker magnetically. The weaker the particles are magnetically, the stronger a force was needed to "coerce" them into the proper patterns to hold data. Therefore, high-density disks have a higher coercivity.

As the capacity of drives increased, the tracks became narrower. The low density 5.25" floppies had 48 tracks per inch and could hold 360K of data. The high density 5.25" floppies have twice as many tracks per inch and can hold 1.2Mb (the added increase is also due to the fact they have 15 sectors per track instead of nine). Because there are more tracks in a given space, they are thinner. Problems arise if you use a disk formatted at 360K in a 1.2Mb drive. Because the 1.2 Mb drive writes the thinner tracks, not all of the track of the 360K floppy is overwritten. This may not be a problem in the 1.2Mb drive, but if you ever try to read that floppy in a 360K drive, the data will run together. That is, the larger head will read data from more than one track.

Formatting a 360K floppy as a 1.2Mb usually fails miserably because of the different number of tracks, so you usually cant get yourself into trouble. However, with 3.5" floppies, the story is a little different. For both the 720Kb and 1.44Mb floppies, there are 80 tracks per side. The difference is that the 1.44Mb floppies are designed to handle 18 sectors per track instead of just nine. As a result, formatting *appears* to go well. It is only later that you discover that the data is not written correctly.

The reason for this is that the magnetic media for the lower-density 720Kb floppies is less sensitive. By formatting it as 1.44Mb, you subject it to a stronger magnetic field than you should. After a while, this "overdose" causes the individual magnetic fields to begin to interfere with one another. Because high-density, 1.44Mb floppies cost a few cents apiece, it's not worth risking data by trying to force low-density to high-density to save money.

While I'm on the subject of money, note that buying unformatted floppies to save money is becoming less and less the smart thing to do. If you figure that formatting floppies takes at least two minutes apiece and the cost difference between a package of ten formatted floppies

and ten unformatted is $2, then it would only make sense (or cents) to have someone format these if they were making only $6.00 an hour. Rarely does a company have someone whose sole job is to format floppies. That job usually falls on those people who use them and most of them get more than $6.00 an hour. Therefore, you may as well buy the formatted floppies.

(I actually did some consulting work for a company whose president insisted that they buy *unformatted* floppies. Because the only people who used the floppies were his programmers and system administrators, they earned more than $6.00 an hour. In one case, I calculated that turning a package of 10 unformatted floppies into formatted floppies worked out to costing twice as much for the unformatted as for the formatted ones. That didn't phase him a bit because the system administrators were on salary and were paid no matter what. By saving the few dollars by buying unformatted ones, his profit margin looked better, at least on paper.)

8.8 Tape Drives

For the longest time, tape drives were a block to me. Although I understood the basic concept (writing to a tape similar to a music cassette), it took me quite a bit of time before I felt comfortable with them.

Because this device has the potential for either saving your data or opening up career opportunities for you to flip burgers, knowing how to install and use them is an important part of your job as a system administrator. Because the tape device node is usually read/write, regular users can also back up their own data with it.

The first tape drives supported under Linux were quarter-inch cartridge tapes, or QIC tapes. QIC is not just an abbreviation for the size of the media; it is also a standard.

In principle, a QIC tape is like a music cassette. Both consist of a long, two-layer tape. The "backing" is usually made of cellulose acetate (photographic film) or polyester (1970s leisure suits), polyester being more common today. The "coating" is the actual media that holds the magnetic signals.

The difference is in the way the tapes are moved from the supply reel to the take-up reel. In cassette tapes, movement is accomplished by a capstan and the tape is pinched between two rollers. QIC tapes spread the driving pressure out over a larger area by means of a drive belt. Additionally, more care is taken to ensure that the coating touches only the read/write heads. Another major difference is the size. QIC tapes are much larger than cassette tapes (and a little bit smaller than a VHS video tapes).

Initially, the QIC tape was 300 feet long and held approximately 30Mb of data. This was a DC300 tape. The tape that next appeared was a DC600, which was 600 feet long and could hold about 60Mb. As with other technologies, tape drives got better and longer and were able to hold more data. The technology advanced to the point where the same tapes could be used in new drives and could store as much as twice as much as they could before.

There are currently several different QIC standards for writing to tape drives, depending on the tape and tape drive being used. Older, 60Mb drives used a QIC-24 format when writing to 60Mb tapes. Newer drives use the QIC-525 format to write to several different kinds of tapes.

As a result, different tapes yield different capacity depending on the drive on which they are written.

For example, I have an Archive 5150 tape drive that is "officially" designed to work with 150MB tapes (DC6150). However, I can get 120Mb from a DC600. Why? The DC600 is 600 feet long and the DC6150 is only 20 feet longer. A tape drive designed to use DC600 tapes only writes in 9 tracks, however, and a tape that uses DC6150s (like mine) writes in 15 tracks. In fact, you can use many different combinations of tapes and drives.

One thing I would like to point out from a technical standpoint is that there is no difference between 150Mb and 250Mb QIC drives. When the QIC standard was enhanced to include 1000-foot tapes, 150Mb drives automatically became 250Mb drives. (I wish I had known this before I bought so many DC6150 tapes. Oh, well, live and learn.)

A similar thing happened with 320Mb and 525Mb tapes. The QIC-320 standard was based on 600-foot tapes. However, the QIC committee decided to go with the QIC-525 standard based on 1000-foot tape. Thats why a 600-foot tape writing with the QIC-525 standard writes 320Mb.

Notice that this entire time, I never referred to QIC--02 tapes. Thats because QIC-02 is not a tape standard, but a controller standard.

An interesting side note is just how the data is actually written to the tape. QIC tape drives use a system called "serpentine recording." Like a serpent, it winds its way back and forth along the length of the tape. It starts at one end and writes until it reaches the other end. The tape drive then reverses direction and begins to write toward the other end.

Other common tape drives are QIC--40 and QIC-80 tape drives, which provide 40Mb and 80Mb, respectively. These provide an inexpensive backup solution. These tape drives are connected to standard floppy controllers and, in most cases, the standard floppy cables can be used. The size of the tapes used for this kind of drive is about the same as a pack of cigarettes.

Aside from using the same type of controller, QIC-40/80 tape drives are similar to with floppy drives in other ways as well. Both use *modified frequency modulation* (MFM) when writing to the device. Sectors are assigned in similar fashion and each tape has the equivalent of a file allocation table to keep track of where each file is on the media.

QIC-40/80 tapes must be formatted before they are used, just like floppies. Because the size of data storage is substantially greater than for a floppy, formatting takes substantially longer. Depending on the speed of the tape drive, formatting can take up to an hour. Pre-formatted tapes are also available and, like their floppy counterparts, the prices are only slightly higher than unformatted tapes.

Because these tape drives run off the floppy controller, it is often a choice between a second floppy drive and a tape drive. The deciding factor is the floppy controller. Normally, floppy controllers can only handle two drives, so this is usually the limit.

However, this limit can be circumvented if the tape drive supports *soft select* (sometimes called "phantom select"), whereby the software chooses the device number for the tape drive when it is using it. The ability to soft select depends on the drive. Though more floppy tape drives support this capability, many of the older drives do not. I will get into more detail about this in the second part of the book when I talk about installing and using tape drives.

On larger systems, neither QIC nor mini-tapes can really handle the volume of data being stored. While some QIC tapes can store up to 1.3Gb, they cannot compare to digital audio tape (DAT) devices. Such devices use Digital Data Storage (DDS) media. Rather than storing signals similar (or analogous) to those coming across the bus, DDS stores the data as a series of numbers or digits on the tape, hence, the name "digital." The result is much higher reliability.

Physically, DATs are the smallest tapes that Linux supports. The actual media is 4mm, so DATs are sometimes referred to as 4mm tapes.

Hewlett-Packard DATs can be divided into multiple logical tapes. This is useful when making backups if you want to store different file systems to different "tapes" and you don’t want to use any extra physical tapes. Device nodes are created to represent these different logical tapes. DAT drives can quickly scan for the location of subsequent partitions (as they are called), making searches much faster than with backups to single tapes.

One thing to watch out for is that data written to DATs are not as standardized as data written to QIC tapes. Therefore, it is possible that data written on one DAT drive cannot be read on another.

There are two reasons for this problem. This first is the blocking factor, which is the minimum space each file will take up. A 1Kb file with a blocking factor of 20 will have 19Kb of wasted space. Such a situation is faster in that the tape drive is streaming more, though there is a lot of wasted space. DAT drives use either a variable or fixed block size. Each drive has a default blocking factor that is determined by the drive itself.

Another problem is data compression, which, if it is done, is performed at the hardware level. Because there is no standard for data compression, it is very unlikely that two drives from different manufactures that both do data compression will be able to read each others tapes.

Keep in mind that that’s not all. There are many more standards that I didn’t list here. One place to start is the QIC consortium’s home page at *www.qic.org*, which lists dozens of tape standards and associated documents.

Before you buy a tape drive, be sure to find out how easy it is to get the tapes and how expensive they are. I bought a tape drive once that was fairly inexpensive, but the tapes were hard to find and more expensive than others. Eventually, I had to special order them from a distributor on the other side of the country, because my local vendor stopped carrying them (I was the only one who used them). The initial cost might have been more for a different tape drive, but I would have saved in the long run.

Tape Loaders

If you have a lot of data to backup, tape loaders can be a real time saver. In essence, a tape loader is a single tape drive with the ability to store multiple tapes. Because the mechanism can load any tape you choose, they function similarly to music jukeboxes. As a result, tape loaders are sometimes called tape jukeboxes.

Most of the tape loaders I have seen come with either five or seven slots. You can fill up all of the slots on Monday and write to a different tape each day of the week. Although this saves time, I would still recommend taking the tape out every day and storing it separately from the machines.

Even so, I still feel it is a time saver to fill the loader once on Monday for the week, particularly if you have a large pool of tapes. For example, in one company, we had enough tapes for a couple of months worth of backups. Our backup software keep track of which tapes were in the drive as well as on which tape any given file resided. We checked once on Monday to see what tapes were needed for the week, filled up the loader and then simply removed each tape as it was used.

On Friday, we did a full backup of every file on the system. This required the loader be filled up completely, since we had some much data. Therefore, having the loader was a necessity for the weekend backups. Therefore, we simply used the available functionality during the week.

As the company and quantity of data grew, we eventually needed more tapes than could fit in a single loader. That meant we had to get a second loader for that machine. Although most of the more advanced backup packages can handle loaders not all of them work well with multiple loaders. Therefore, you should check in advance before buying something that cannot grow with you.

8.9 CD-ROMS

Linux distribution media is becoming more and more prevalent on CD-ROMs. One CD-ROM takes a lot less space than 50 floppies or even one QIC tape, so the media is easier to handle. In fact, I would be hard-pressed to find a version that is still being distributed on floppies. Added to this, CD-ROMs are significantly faster than either floppies or tape media.

Another important aspect of CD-ROMs when it comes to installing media is their size. Therefore, it is possible to get numerous products on the CD-ROM. You can get a single CD-ROM that contains a runnable copy of Linux and the source code with room to spare.

CD-ROMs, in fact CD-ROMs technology in general, have always fascinated me. I am amazed that you could get so much information into such a small place and still have such quick access to your data.

The basic principle behind data storage on a CD-ROM is really nothing more than Morse code. A series of light and dark (dots and dashes) compose the encoded information on the disk. Commercial CD-ROMs, whether music or data, almost universally have data on one side of the disk. Although there is nothing technologically preventing a CD-ROM from having a flip side, current convention limits data to just a single side. This is enough when you consider

that you can get more than 600Mb of data on a single CD-ROM. As the technology improves, the amount is steadily increasing. In addition, certain manufacturers are working on dual-sided CD-ROMs.

On the surface of the disk are a series of dents, or holes, called "lands." The areas between the lands are called "pits." A laser is projected onto the surface of the disk and the light is either reflected by the pits or scattered by the lands. If reflected, the light reaches a light-sensing receptor, which then sends an electrical signal that is received by the control mechanism of the CD-ROM drive. Just as the pattern of alternating dots and dashes forms the message when using Morse code, it is the pattern of reflected light and no light that indicates the data stored on the disk.

When I first thought of CD-ROMs, I conceptualized them as being like WORM (Write-Once, Read-Many) drives, which they are, somewhat. I visualized them as being a read-only version of a hard disk. However, after looking more closely at the way data is stored, I saw that CD-ROMs have less in common with hard disks.

As you remember from our discussion of hard disks, each surface is composed of concentric rings called tracks, and each track is divided into sectors. The disk spins at a constant speed as the heads move in and out across the drives surface. Therefore, the tracks on the outer edges move faster than those on the inside.

For example, take a track that is a half-inch away from the center of the disk. The diameter of the circle representing the track is one inch, so the radius of that circle is approximately 3.1415 inches. Spinning 60 times a second, the track goes at a speed of about 190 inches per second. Now, take a track at one inch from the center, or twice as far. The diameter of the circle representing that track is 6.2830 inches. It, too, is going around at 60 revolutions per second. However, because it has to travel twice as far in each revolution, it has to go twice as fast.

A CD-ROM isn’t like that. CD-ROMS rotate in a manner called "constant linear velocity." The motor keeps the CD-ROM moving at the same speed, regardless of where the CD-ROM reader is reading from. Therefore, as the light detector moves inward, the disks slows down so that each revolution takes the same amount of time per track.

Lets look at hard disks again. They are divided into concentric tracks that are divided into sectors. Because the number of sectors per track remains constant, the sectors must get smaller toward the center of the disk (because the circumference of the circle representing the track is getting smaller as you move in).

Again, a CD-ROM isn’t like that. Actually, there is no reason why it should work that way. Most CD-ROMs are laid out in a single spiral, just like a phonograph record. There are no concentric circles, so there is no circumference to get smaller. As a result, the sectors in a CD-ROM can remain the same size no matter what. The added advantage of sectors remaining the same size means there can be more on the disk and therefore more data for the user.

Currently, CD-ROMS have a capacity of *at least* 650Mb, although I am aware of some that are already more than 700Mb. Several companies are currently working on technology to get even more out of them. The simplest technology involves making the CD-ROM writable on *both* sides, which simply doubles the capability. Others involve storing the data in different layers on the CD-ROM and using light that is polarized differently to read the different layers.

The single-speed CD-ROM drives are the oldest and have a transfer rate of about 150Kb a second. Recently (as of this writing), eight-speed CD-ROM drives are available. I often see both four-speed and six-speed drives sold in machines, but the single- and double-speed drives are slowly dying out.

Most CD-ROMS are formatted with the ISO-9660 format, which allows them to be read by a wide range of machines, including DOS/Windows, Macintosh, as well as Linux and other UNIX dialects. The shortcoming, though, is just that: short, as in short file names. The ISO-9660 format only recognizes the DOS 8.3 file names, so the long file names that we have grown to love in UNIX are not possible. As a result, the Rock Ridge Extensions to the ISO-9660 format allow many of the typical UNIX file stuff, like longer file names, file ownership, and even symbolic links, etc.

Another standard is the PhotoCD, which was developed by Kodak to store photographic information. Not every CD-ROM drive can access these CD-ROMs, which require special software. (As a side note, one system administrator working on the PhotoCD project at Kodak had Linux installed to do network monitoring!)

CD-ROMs first became popular as SCSI devices. However, because most people could not afford a SCSI host adapter, companies began producing CD-ROM drives that ran on IDE controllers, and therefore looked just like a hard disk, or would run on their own controllers. Some even enabled you to run the CD-ROM off a sound card or through your parallel port. For details on how to get these to work correctly, see the CD-ROM HOWTO.

8.10 Serial Ports

For the most part, all PC computers, even laptops are sold with at least one serial port and desktop machines or servers typically have two. These can be built into the motherboard, as part of a serial/parallel card, or as part of an "all-in-one" card that has serial ports, parallel ports, games ports, and even hard disk and floppy controllers. (*Note: I have been told that newer laptops, particularly those from Compaq and IBM have only USB and no serial ports!*)

As the name implies, serial device communicate serially. That is, each byte is split into 8 bits, which are sent one after the other. At the pyhsical level, what this means is that voltage on the transmit pin (line) is toggled to indicate the bits: negative is a 1 bit, and a positive voltage is a 0 bit. On the receiving end, the serial port accept the data on the receive pin and recognizes the voltage and therefore whether the bit is a 0 or 1.

Agreeing on the speed at which the voltage is toggled, is an import apect of the communication. The speed of the change determine how many bits per second can be sent, which is thus the speed of the serial port or serial device. A mouse will communicate with speeds between 1,200 and 9,600 bps, whereas modems can reach as high as 115,200 bps.

Deciding on speed is not the only thing used with serial communication to data arrives correctly. In some cases, like with modems, additional pins are used to say that one side is read to send the data (using the RTS pin) and the other side says that they are clear to send (CTS). Here a voltage of +12 is used for the respective signal. Other kinds of devices uses the pins data terminal ready (DTR) and data set ready (DSR). This has the same functionality, but just uses different pins. Typically, serial terminal use the DTR and DSR pins

Regardless of what pair of pins are used, this mechanism is used to ensure that data only flows when the appropriate side is ready. That is, it is used to control the flow and such a mechanism is thus called "flow control". Since it the hardware that is used to control the flow, using pins like this is logically called hardware flow control. In other cases, such as when there is no phyiscal connection, you can use "software flow control", which uses special "control characters". For example, pressing the control key (CTRL) and the the "s" key (CTRL-S) you can stop the flow. Pressing CTRL-Q, you can start it up again.

With modems, you will typically find that a couple of other pins are used as well: ring indicator (RI) pin and the carrier detect (CD) pin. These indicate (respectively) an incoming call and that a the carrier signal has been detected.

A serial board is an expansion card that translates bus signals in which at least eight bits arrive simultaneously into signals that send single bits at a time. These bits are encapsulated into groups of one byte. The encapsulation contains other signals that represent the start and end of the byte, as well as a parity bit. Additionally, the number of bits used to represent data can be either 7 or 8.

Parity is the mechanism by which single-bit errors can be detected during transmission. The number of bits set to one is counted and based on whether even or odd parity is used, and the parity bit is set. For example, if even parity is used and there are three bits set, then the parity bit is also set to make the total number of bits set an even number. However, if odd parity is used, the number of bits set is already odd, therefore the parity bit is left unset. When you use some other means to detect errors, you can turn parity off, and you are said to be using no parity. This is the default for modems in Linux.

Serial communication parameters must be agreed upon by both ends. These parameters are often referred to in triples, such as 8-1-N (read as eight-one-none). In this instance, there are eight data bits, 1 stop bit, and no parity is used. This is the default for Linux systems.

One key element of a serial board is the Universal Asynchronous Receiver-Transmitter, or UART. The transmitter portion takes a byte of parallel data written by the serial driver to the card and transmits it one bit at a time (serially). The receiver does just the opposite: It takes the serial bits and converts them into parallel data that is sent down the bus and is read by the serial driver.

Originally, Linux only provided drivers for standard serial ports; intelligent boards are often installed to allow many more logins (or other connections) to the system. The most significant difference is that intelligent serial boards (often referred to as smart serial boards) have a built-in CPU, which allows it to take all responsibility for processing the signals away from the system CPU.

In the newer versions, you can find drivers for commercial multiport boards, such as the Stallion Easy IO, which allows you to quickly extend the number of serial ports on your system as the drivers are built in. Stallion is very supportive of the Linux world and even advertises the fact that their boards run on Linux.

In addition, intelligent serial boards can better buffer incoming signals that keep them from getting lost. With non-intelligent boards, the system may be so busy that it does not get around in time to read characters off the board. Although the 16550 UART, common on most serial boards today, contains 16-byte buffers, this is often not enough. Under heavy load, the serial driver does not react fast enough and the characters are overwritten.

Serial board performance is also increased by intelligent boards. Because signals are buffered and sent in large chunks, there is less overhead on a per-character basis. With non-intelligent boards, single characters are often transmitted, so the per-character overhead is much larger. In fact, most non-intelligent boards generate and interrupt the associated overhead with *each* character.

It is possible to obtain supported serial boards that have multiple ports. Although such boards have multiple UARTs, they do not have the performance of intelligent boards, though they do provide a low-cost alternative. For a discussion on the device nodes used for such boards, see the section on the device directory.

The pin assignments mentioned above and the commincation protocol (among other thinfs) is known as the RS-232 standard. RS-232 is easily effected by electrical noise, and thus has some limits the length and speed at which it can communicate. The longer the cable the slower the connection must be to ensure communication is not disrupted.

Originally designed to connect mainframe computers to modems, the RS-232 standard is used exclusively for serial port connections on PCs. Two kinds of devices are considered with RS-232: Data Terminal Equipment (DTE) and Data Communication Equipment (DCE). DTE is the serial port side and DCE is the modem side.

Two types of connections are used: DB25 (with 25 pins) and DB9 (with 9 pins). Although they both serve the same basic function, the numbering of the pins is slightly different. Table 0-3 lists the main pins of the DB25 connector, their functions, and a mnemonic that is commonly used to refer to them. Table 0-4 lists the pins for the DB9 connector. The physical layout of the pins is shown in Figure 0-13.

Pin	Function	Mnemonic
2	Transmit	TXD or TX
3	Receive	RXD or RX
4	Request to send	RTS
5	Clear to send	CTS
6	Data set ready	DSR
7	Signal ground	GND
8	Carrier detect	CD
20	Data terminal ready	DTR
22	Ring indicator	RI

Table -3 Common Pins on the DB25 Connector

Pin	Function	Mnemonic
1	Carrier detect	CD
2	Receive	RXD or RX
3	Transmit	TXD or TX
4	Data terminal ready	DTR
5	Signal ground	GND
6	Data set ready	DSR
7	Request to send	RTS
8	Clear to send	CTS
9	Ring indicator	RI

Table -4 Pins on DB9 Connector

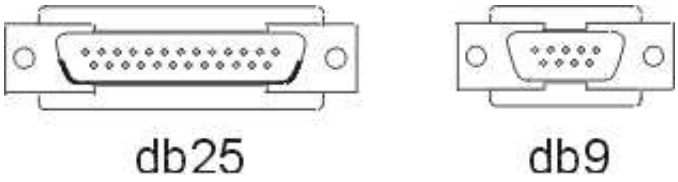


Figure - The Physical Layout of Pins on Serial Cables

Note that on a DB25 connector, pin 1 is *chassis ground*, which is different from signal ground. Chassis ground ensures that both serial connectors are operating at the same electric potential and keeps you from getting a shock.

To communicate properly, the DTE device must say that it is ready to work by sending a signal on the DTR line. The DCE device must do the same on the DSR line.

One side indicates that it has data by sending a signal on the RTS line (it is requesting to send data). If ready, the other side says that it is ready by sending a signal on the CTS line (the sender is clear to send the data). What happens when the receiving side can't keep up (that is, if the sending side is sending too fast)? If the receiving side needs to stop (perhaps a buffer is full), it stops the CTS signal (meaning the sender is no longer clear to send the data). This causes the sending side to stop. This process is referred to as *hardware handshaking*, *hardware flow control*, or *RTS/CTS flow control*.

Problems arise when connecting other types of devices. Some devices, such as printers, are themselves DTE devices. If you tried to connect a standard RS-232 cable, TX is connected to TX, RX is connect to RX, DSR is connected to DSR, and DTR is connected to DTR. The result is that nothing happens. The solution is a *cross-over* cable, which internally swaps the appropriate signals and makes sure they go to the right place.

If you have a terminal, things are easier. First, though the data is going in both directions, the data coming from the terminal will never exceed the speed of the serial port (I'd like to see you type 240 characters per second). Data heading toward the terminal is displayed on the screen, which will display it as quickly as it comes. Therefore, you only need three signals: send, transmit, and ground.

If the terminal displays the data too fast for you to read, you can stop it by sending an XOFF character back to the system. This is usually Ctrl+S and unless it is turned off, it will stop incoming data. To turn the flow of data back on again, send the system an XON (Ctrl+Q) character. This type of flow control is called *software flow control* or *XON/XOFF flow control*. In some cases, depending on how the terminal is configured, sending *any* character restarts the flow.

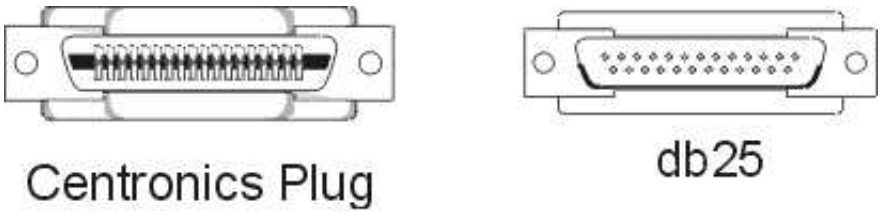
8.11 Parallel Ports

Parallel ports are a common way printers are attached to an Linux system. Although many different problems arise with printers attached to parallel ports, not many issues arise with parallel ports themselves.

First, lets take a look at how parallel ports work.

One key difference between a parallel and serial port is the way data is sent. From our discussion of serial ports, you know that data goes across a serial line one bit at a time across a single data line. Parallel ports send data across a byte (eight bits) at a time across eight data lines.

Another key difference is the cable. Looking at the computer end, you can easily confuse the cable with a serial connector. Both have 25 pins in the same layout. On the printer end, though, things are different. There is a special kind of 36-pin connector called a *Centronics* connector, named after the printer manufacturer Centronics. A cable that has a 25-pin *D-shell* connector on one end and a 36-pin on the other is called a Centronics or parallel cable. (see Figure 0-14) Unlike serial cables, there are not different kinds of cables (like straight-through or crossed). Because of this, all you usually need to do is to plug in the cable at both ends and go.



Comparison of Centronic and DB25 Connectors

Although some devices allow communication in both directions along a parallel port, Linux does not support this communication. In fact, the only thing that Linux directly supports on parallel ports are printers.

Because there is no guarantee that all the data bits arrive at the port at the same time, there must be some way of signaling the printer that the data is ready. This is done with the *strobe* line. Once a character (or any byte of data) is ready, the system sends a signal along the strobe line. Using the strobe line also prevents characters from being read more than once.

Often, the printer cannot keep up with the data flow from the parallel port. Just like RTS-CTS flow control on a serial port, parallel ports also need a way to be told to stop. This is done with the *busy* line. Actually, the busy line is set after each character in case the printer cannot process the character fast enough. Once the character is processed, the printer can turn off the busy signal.

However, this is not enough to get the parallel port to send the next character. The printer must first tell the parallel port it has received the character by sending a signal along the acknowledge line. Note that this acknowledge occurs after *every* character.

The printer also uses other control lines. One is the *select,* which indicates that the printer has been selected or is on-line. There is also a special line that indicates when the paper source is empty. This is the *paper empty* line. If the problem is unknown, the printer can send a signal along the *fault* line that says that "something" is wrong.

One thing that comes up regularly is the confusion about which physical parallel port is related to which lp device. For your parallel ports to work correctly, you must configure them according to Table 0-5.

Table -5 Default Parallel Port Devices

Device name	Address	IRQ
/dev/lp0	0x378	7
/dev/lp1	0x3BC	7
/dev/lp2	0x278	5

8.12 Video Cards and Monitors

Without a video card and monitor, you don't see anything. In fact, every PC that I have ever seen won't even boot unless there is a video card in it. Granted, your computer could boot and even work without being attached to a monitor and I have seen those, but it's no fun unless you get to see whats going on.

When PCs first hit the market, there was only one kind of video system. High resolution and millions of colors were something you read about in science-fiction novels. Times changed and so did graphics adapters. The first dramatic change was with the introduction of color with IBM's color graphics adapter CGA, which required a completely new and incompatible video subsystem. In an attempt to integrate color and monochrome systems, IBM came out with the enhanced graphics adapter EGA.

But I'm not going to talk about those. Why? First, no one buys them anymore. I doubt that anyone still makes them. If you could find one, there would be no problem at all to install them and get them to work. The second reason why I am not going to talk about them is because they are not that common. Because "no one" uses them any more, the time I spend telling you why I won't tell you about them is already too much.

What are we going to talk about instead? Well, the first thing is Video Graphics Array VGA. VGA is the standard by which virtually all video card manufacturers base their products. Though enhancements to VGA Super VGA or SVGA exists, it is all based on VGA.

When talking about VGA, I first need to talk about some basics of video technology. The first issue is just how things work. Digital signals are sent by the operating system to the video card, which sends them through a digital-to-analog converter DAC. Usually a single chip contains three DACs, one for each color red, green, and blue, or RGB. The DAC has a look-up table that determines the voltage to be output on each line for the respective color.

The voltage that the DAC has found for a given color is sent to the three electron guns at the back of the monitors cathode ray tube CRT, again, one for each color. The intensity of the electron stream is a result of this voltage.

The video adapter also sends a signal to the *magnetic deflection yoke,* which aims the electron beams to the right place on the screen. This signal determines how far apart the dots are, as well as how often the screen is redrawn. The dots are referred to as *pixels*, the distance apart they are is the *pitch,* and how often the screen is redrawn is the *refresh rate.*

To keep the beams precisely aligned, they first pass through a *shadow mask*, a metal plate containing hundreds of thousands of little holes. The dot pitch is distance between the holes in the shadow mask. The closer the holes, the lower the pitch. A lower pitch means a sharper image.

The electrons from the electron guns strike the phosphors inside the monitor screen and make them glow. Three different phosphors are used, one for each color. The stronger the beams, the more intense the color. Colors other than RGB are created by varying the amount displayed each of these three colors, that is, changing the intensity of each color. For example, purple would be created by exciting red and blue but no green phosphors. After the beams stops hitting the phosphor, it still continues to glow for a short time. To keep the image on the screen, the phosphor must be recharged by the electron beam again.

The electron beams are moved across the screen by changing the deflection yoke. When the beams reach the other side, they are turned off and returned to the starting side, just below the line where they left off. When the guns reach the last line, they move back up to the top. This is called *raster scanning* and it is done approximately 60 times a second.

When the beam has completed a line, it needs to return to the other end to begin the new line. This is called *horizontal retrace.* Similarly, when the beam has reached the bottom, it needs to move back to the top again. This is the *vertical retrace.* In both cases, the beams intensity is reduced to the point that nothing is drawn on the screen. This is called *blanking.*

Some monitor manufacturers try to save money by using less expensive components. The trade off is that the beams cannot scan every line during each pass. Instead, they scan every other line during the first pass then scan the lines they missed during the second pass. This is called interlacing because the scan lines are *interlaced.* Although this provides higher resolutions in less expensive monitors, the images will "flicker" because the phosphors begin to fade before they can be recharged. This flickering gives me, and other people, a headache.

For most users, the most important aspect is the *resolution*. Resolution determines the total number of pixels that can be shown on the screen. In graphics mode, standard VGA has a resolution of 640 pixels horizontally and 480 pixels vertically. By convention, you say that your resolution is 640-by-480.

A pixel is actually a set of three phosphors rather than just a single phosphor. So, in essence, a pixel is a single spot of color on the screen. What color is shown at any given location is an *interaction* between the operating system and the video card. Years ago, the operating system or program had to tell the video card where each dot on the screen was. It had an internal array or table of pixels, each containing the appropriate color values. Today, some video cards can be told to *draw*. They don’t need to know that there is a row of red dots between points A and B. Instead, they are simply told to draw a red line from point A to point B. This results in faster graphics because the video card has taken over much of the work.

In other cases, the system still needs to keep track of which colors are where. If we had a truly monochrome video system, any given pixel would either be on or off. Therefore, a single bit can be used to store that information. If we go up to 16 colors, we need 4 bits, or half a byte of information $2^4 = 16$. If we go to a whole byte, then we can have 256 colors at once 2^8. Many video cards use three bytes to store the color data, one for each of the primary colors RGB. In this way, they can get more than 16 million colors!

Now, 16 million colors seems like a lot, and it is. However, it's actually too much. Humans cannot distinguish that many colors, so much of the ability is wasted. Add to that that most monitors are limited to just a few hundred thousand colors. So, no matter what your friends tell you about how wonderful their video card is that does 16 million colors, you need not be impressed. The odds are the monitor can't handle them and you certainly can't see them.

However, don't think that the makings of video cards are trying to rip us off. In fact, it's easier to design cards that are multiples of whole bytes. If we had a 18-bit display which is needed to get the 250K of colors that monitors could handle, we either use 6 bits of three different bytes or two whole bytes and 2 bits of the third. Either way, things are wasted and you spend time processing the bits. If you know that you have to read three whole bytes, one for each color, then there is not as much processing.

How many pixels and how many colors a video card can show are interdependent of each other. When you bought it, your video card came with a certain amount of memory. The amount of memory it has limits the total number of pixels and colors you can have. If you take the standard resolution of a VGA card of 640x480 pixels, that's 307,200 pixels. If we want to show 16 colors, that's 307,200 x 4 bits or 1,228,800 bits. Dividing this by eight gives you 153,600 bytes needed to display 640x480 in 16 colors. Because memory is usually produced in powers of two, the next smallest size is 256 Kilobytes. Therefore, a video card with 256K of memory is needed.

Maybe this is enough. For me, I don't get enough on the screen with 640x480, and only 16 colors looks terrible at least to me. However, if you never run any graphics applications on your machines, such as X-Windows, then there is no need for anything better. Operating in *text mode,* your video card does fine.

As I said, I am not happy with this. I want more. If I want to go up to the next highest resolution 800x600 with 16 colors, I need 240,000 bytes. I still have less than the 256K I need for 640x480 and 16 colors. If, instead, I want 256 colors which requires 8 bits per pixel, I need at least 480,000. I now need 512K on the video card.

Now I buy a great big monitor and want something closer to "true color." Lets not get greedy, but say I wanted a resolution of 1,024x768 the next highest up and "only" 65,635 colors. I now need 1,572,864 bytes of memory. Because my video card has only 1Mb of memory, I'm out of luck!

But wait a minute! Doesn't the VGA standard only support resolutions up to 640x480? True. However, the Video Electronics Standards Association VESA has defined resolutions more than 640x480 as Super VGA. In addition to the resolutions I mentioned previously 800x600 and 1,024x768, SVGA also includes 1,280x1,024 and 1,600x1,200.

Okay. The mere fact that you have a video card that handle SVGA resolutions does not mean you are going to get a decent picture or at least not the picture you want. Any system is only as good as its worst component, and this also applies to your video system. It is therefore important to understand a characteristic of your monitor: pitch. I mentioned this briefly before, but it is important to talk about it further.

When shopping for a monitor, you will often see that among the characteristics used to sell it is the pitch. The values you would see could be something like .39 or .28, which is the spacing between the holes in the shadow mask, measured in millimeters. Therefore, a pitch of .28 is just more than one-quarter of a millimeter. The lower the pitch, the closer together the holes and the sharper the image. Even if you aren't using any graphics-oriented programs, it's worth the few extra dollars to get a lower pitch and the resulting sharper image.

8.12.1 Video Card Common Problems

A common complaint with fresh Linux installs on modern hardware is the lack of an acceptable device driver to take advantage of the capabilities that modern graphics cards possess. While numerous possibilities exist to harness these capabilities, both the ATi and Nvidia companies provide binary drivers that enable the 3D portions of their graphics chips to work under Linux.

Specific information regarding ATi graphics cards can be found here

Linux drivers for ATi graphics cards can be found here.

Specific information regarding NVidia graphics cards can be found here (Intel x86 Platform)

Linux drivers for NVidia graphics cards can be found here: (General) (Intel x86 Platform)

Driver distribution among ATi and NVIDIA varies significantly. ATi provides a single RPM containing its drivers. Inside this RPM are a number of pre-compiled drivers for specific versions of the RedHat distribution. For setups sporting a custom kernel, or a distribution other than RedHat, ATi provides a script that compiles a driver suitable for the specific configuration. This compilation process will only work with the gcc compiler, version 2.96 or greater.

ATI provides only two driver downloads. One driver is designed for distributions running XFree86 version 4.1.0, and the other is designed for distributions running 4.2.0. Most recent distributions include 4.2.0. It is recommended that you check the specifics of your distribution to determine which version of XFree86 it uses.

Upon downloading the RPM, navigate to /lib/modules/fglrx , where the unpacked drivers now reside. As root, run the "make_install.sh" script. This can be accomplished by issuing the command "./make_install.sh". This script detects whether an appropriate driver exists for your kernel. There are several mainstream kernels that are supported. On computers with unsupported kernels, make_install.sh will stop with an error message, indicating you need to build a custom module. Navigate to the build_mod directory, and type "make". This will produce a module compiled specifically for your kernel. Now /lib/modules/fglrx/build_mod can be run without error.

ATI's driver is now ready to be installed. Navigate to /usr/X11R6/bin. From there, run "fglrxconfig". This script will update your xconfiguration to properly reflect the new drivers. Upon completing that, restart XFree86.

NVIDIA provides in two parts. The first is called the "GLX File." This provides the OpenGL software layer to applications in Linux. To avoid version uncertainty with regards to these files, NVIDIA suggests downloading and running the "NVChooser" script available at their website.

The second part of NVIDIA's driver is called the kernel driver. The kernel driver provides an interface between Linux's "guts" (the kernel) and the hardware of your graphics card. Given that the kernel can be compiled numerous ways, NVIDIA provides distribution-specific drivers. These drivers are compatible with the default kernel of a number of versions of Redhat, Suse, Mandrake, and United Linux. If you have one of these distributions, sporting the original kernel, then simply download the driver appropriate for it. If you have compiled a custom kernel, or have a distribution other than those supported by NVIDIA, then download the Source Driver. The Source Driver provides a way to create drivers appropriate for your kernel, no matter which it is. The README file found in the Source Driver tarball gives directions for compiling this driver.

To make sure that new graphics drivers for 3D equipment have been properly installed, run "glxgears" from an X-Terminal. A frame rate around 2000fps indicates that hardware acceleration is being used. Also, "glxinfo" will display information regarding the 3D acceleration XFree86 is using. When 3D drivers have been installed, glxinfo will usually provide the name of the card and information about it. If 3D drivers have not been installed, it will simply display information about the Mesa 3D software library.

8.13 Modems

Up to this point, I've talked about things that almost every computer user has and almost always come with the system. We are now moving into a new area, where we get one computer to talk to another. In my opinion, this is where computers begin to show their true power.

Perhaps the earliest means of getting computers to talk to one another at least over long distances was the modem. Modem stands for Modulator/Demodulator, which is its basis of operation. It takes the digital signals that come across the bus from the CPU and converts them into a signal that can be understood by the telephone wires. This process is called *modulation*. On the other end, the signals are converted back from telephone signals into computer signals by a process called *demodulation*. Graphically this looks like Figure 0-15.

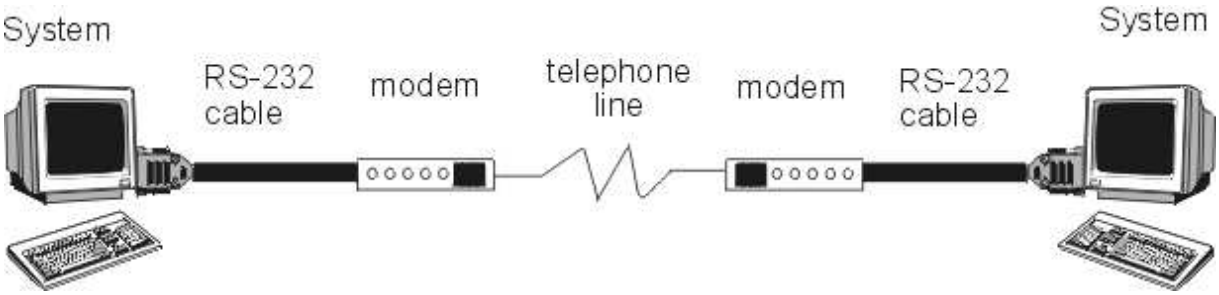


Image - Transfer of Data Across Modems

The underlying concept of the transmission of data across the telephone line is that of a *carrier*. A carrier is a signal that runs along the phone line that is at a constant strength amplitude, frequency, and phase. Because all of these are known values, changes in them can be detected. It is the changes that are used to encode the data.

When data are sent at relatively low speeds, the exact timing of that data being sent is not important. Markers, the start and stop bits, used within the transmitted data indicate the beginning and the end of each piece of data. Note: You could have two stop bits. If each modem is set to the same values, it knows when one piece of data stops and the next one begins. This is called asynchronous transfer.

How big is that piece of data? Usually just a single character. All the modems that I have ever seen have either 7 or 8 bits for data. That means that there are 7 or 8 bits between the start-stop bit pairs .This, too, is something that both modems need to agree on.

Parity works like this. Lets assume that a specific byte has 3 bits that are set. If you are using even parity, then the parity bit would be set to make the total number set four, which is an even number. If you are using odd parity, the number of the bit is already an odd number three, and the parity bit would not be set.

When the settings at which your modem must be are determined, the order is usually the number of data bits, followed by the number of stop bits, then the parity. By default, Linux uses eight data bits, one stop bit, and no parity. It is common to refer to this as "eight, one, and none," or "8-1-N." Other settings might be 7-2-E for seven data bits, two stop bits, and even parity.

Another important characteristic is the speed at which the modem transmits the data. Although the exact timing is not critical, signals must be received within a certain time or problems happen. For example, you could be waiting for months if the connection suddenly dropped.

Now, lets go back to the modulated carrier wave. The term for the number of changes in the carrier wave per second is *baud*, named after the French telegraph expert, J. M. E. Baudot. One way of encoding data, based on the changes to the carrier wave, is called *frequency shift keying*, or FSK. The number of changes that can take place per second is the number of bits of information that can be sent per second one change = one bit.

Lets consider a modem connection that operates at 2400 baud, eight data bits, one stop bit, and no parity. This gives us a total of 10 bits used for each character sent. Did you forget the start bit? Because baud is a measurement of the number of *bits* sent per second, 2400 baud means that 240 characters can be sent per second.

Other encoding methods result in getting more bits per baud. For example, the Bell 212A standard operates at 300 baud. However, because it gets four bits of data per baud, it gets 1,200 bits per second for those 300 baud. This rate is accomplished by changing more than just the frequency. If you changed both frequency and amplitude, you have four distinct values that you could use.

Have you ever had someone tell you that you have a 9600 baud modem? Don't believe them! There is no such thing. In fact, the fastest baud rate is only 2400. So what are people taking about when they say their modem goes 9,600 or 14,400? They are talking about the *bits-per-second* bps. If you get one bit-per-baud, then these terms are synonymous. However, all 9600 modems get more than that. They operate at 2400 baud but use a modulation technique that yields 4 bits per baud. Thus, a 2400 baud modem gives 9,600 bits per second.

As with all the other kinds of hardware I've talked about, modems must have standards to be useful. Granted, you could have a modem that can only communicate with another from the same manufacturer, but even that is a kind of standard.

Modem standards are like opinions: everyone has one. There are the AT&T standards, the Bell standards, the International Telecommunications Union ITU standards which was formally the Comite Consultatif International Telegraphique et Telephoneique, or CCITT, and the Microcom Networking Protocol MNP standards.

As of this writing, the two most common standards are the CCITT and MNP. The MNP standards actually work in conjunction with modems that adhere to the other standards and for the most part define technologies rather than speeds or other characteristics.

The CCITT/ITU standards define among other things modulation methods that allow speeds up to 9,600bps for the V.32 standard and 14,00bps for the V.32 bis standard. The new V.34 standard supports 2,800bps. One newer standard, V.42, is accepted worldwide and provides error-correction enhancements to V.32 and V.32 bis. The V.42 standard also incorporates the MNP 4 standard, enabling one modem that supports V.42 to communicate with another that supports MNP 4. For much more detail on the different modem standards, see *The Winn L. Rosch Hardware Bible*, Third Edition, and the *Modem Reference*, Second Edition, by Michael Banks. Both are published by Brady Books.

One standard we need to go into is the *Hayes command set*. This set was developed by and named for the modem manufacturer Hayes and is used by almost every modem manufacturer. It consists of dozens of commands that are used to modify the functionality as well as read the characteristics of your modem. Most of the commands in this set begin with AT which is short for "attention", so this is often referred to as the AT command set. Note that the AT and almost every other letter is capitalized.

Several AT commands can be combined in a single string, and this is often used to initialize the modem before first use. These commands can set the default speed, whether the modem should automatically answer when someone calls in, and even how many rings it should wait for. I'll talk about these in more detail later when I talk about configuring modems.

Modems come in two forms: internal and external. Because a modem is a serial device it communicates serially as opposed to parallel, it will always take up a serial port. With an external modem, you must physically make the connection to the serial port, so you are more conscious of the fact that the modem is taking up a port. Internal modems also take up a serial port, but it is less obvious. Because you don't actually see the modem, some users don't realize that they no longer have a COM1 or COM2.

External modems are usually connected to the computer via a 25-pin RS-232 connector. Some serial ports have only a 9-pin serial port, so you need an adapter to convert the 25-pin to 9-pin, because *every* modem I have every seen has a 25-pin connector.

So, what happens when I want to dial into another site or send an e-mail message to my sister in Washington? Well, the communications software maybe cu or uucp sends a signal an increase in voltage along pin 20 data terminal ready, or DTR to tell the modem that it is ready to transmit data. On the modem, the equivalent pin is pin 6, data set ready, or DSR.

The modem is told to go "off hook" via the transmit data line TX, line 2. Shortly thereafter, the system sends the AT-commands to start the modem dialing either with pulses ATDP or with tones ATDT. The modem acknowledges the commands via line 3 receive data, or RX.

The modem dials just like a phone and tries to connect to some device on the other end, probably a modem. If *auto answer* is enabled, the modem being called should answer, or *pick up*, the modem. When the connection is made, the calling modem sends a high-pitched signal to tell the receiving modem that a modem is calling. The receiving modem sends a higher-pitched acknowledgment. You can hear this if your modem has a speaker.

The carrier signal is then established between the two modems, which is kept at a steady, predetermined frequency. This is the signal that is then modulated actually to transmit the data. When the modem has begun to receive this carrier signal, it sends another signal back to the system via line 8 carried detect, or CD. This signal is held active for the duration of the call.

The two modems must first decide how they will transmit data. This negotiation is called a handshake. The information exchanged includes many of the things that are defined in the different standards I talked about earlier.

When the system is ready to send data, it first raises line 4 request to send, or RTS. If it is ready, the modem says okay by raising line 5 clear to send, or CTS. Data then is sent out on line 2 and received on line 3. If the modem cannot keep up, it can drop the CTS line to tell the system to stop for a moment.

8.14 Printers

Although more and more companies are trying to transform into a "paperless office," you will undoubtedly see a printer somewhere. Even if the office is paperless internally, it will have to use paper of some kind to communicate with the rest of the world.

Printers come in many different shapes, sizes, formats, means of connection to the system, ways of printing characters, speeds, and so on. The two most common ways to connect printers are by serial port or parallel port. Linux also supports Hewlett-Packard Laser Jet printers equipped with JetDirect cards. These cards allow the printer to be attached directly to a network, thereby increasing its speed. I'll talk more about these later. In addition, although they are not supported by Linux as of this writing, SCSI printers have appeared on the market.

In previous sections, we talked about serial and parallel connections, so I don't need to go into detail about them. I do talk about these connections in more detail in the second part of the book, however, when I talk about installing and configuring printers.

There are two kinds of printers that, although were once very common, are now making way for more advanced successors: the daisy-wheel and chain printers. The distinction these printers had is that they had preformed characters.

In the case of a daisy-wheel printer, printing was accomplished by means of a wheel, where the characters were at the end of thin "leaves," which made the daisy shape. The wheel rotated very fast and as the appropriate letter came into position, it was struck with a hammer that forced the leaf with the character on it against the ink ribbon, which then struck the paper. This mechanism uses the same principle as a normal typewriter. In fact, there are typewriters that use the same daisy-wheel principle.

Chain printers also have preformed letters. Instead of a wheel, however, the letters are on a long strip called a chain. Instead of rotating, the chain moves back and forth to bring the appropriate letter into position.

Although these printers are fairly quick, they are limited in what they can print. You could get pretty tricky in which characters you use, and come up with some rather cute pictures. However, these mechanisms aren't able to do anything very detailed.

The next step in printers was the impact dot-matrix printer. These, too, had hammers, but rather than striking preformed letters, the hammers themselves struck the ink ribbon. Instead of a single hammer, there was a column of usually 9 or 24 hammers, or pins. Such printers are called 9-pin or 24-pin printers.

As the printer prints, the heads move across the page and print out columns of dots. Depending on what character is to be printed, some of the pins do not strike the ink ribbon. For example, when a dash is printed, only the middle pins strike the ribbon. When printing a more complex character like an ampersand &, the pins strike at different times as the print head moves across the page.

As with monitors, the more dots you have, the sharper the image. Therefore, a 24-pin printer can produce a sharper image than one with only 9 pins. In most cases, the type of printer used is obvious the moment you see something printed with a 9-pin printer. Some 24-pin printers require a closer look before you can tell.

Next, printers began to get rid of the ink ribbon and replace the pins with little sprayers connected to a supply of ink. Instead of striking something, these sprayers squirt a little dot of ink onto the paper. The result, similar to that of an impact dot-matrix printer, is what an ink jet printer does.

Ink jet printers have two advantages over impact dot-matrix printers. First is the issue of noise. Because no pins are striking the ink ribbon, the ink jet printer is much quieter. Second, by extending the technology a little, the manufacturer increased the number of jets in each row. Also, instead of just squirting out black ink, you could squirt out colored ink, which is how many color printers work.

The drawback is the nature of the print process itself. Little sprayers squirting ink all over the place is messy. Without regular maintenance, ink jets can clog up.

Using a principle very similar to video systems, laser printers can obtain very high resolution. A laser inside the printer hence the name scans across a rotating drum that has been given a static-electric charge. When the laser hits a spot on the drum, that area looses its charge. Toner then spreads across the drum and sticks to those areas that retain the charge. Next, the drum rolls the paper across, smashing the toner onto the paper. Finally, the toner is fused into the paper by means of a heating element.

Although laser printers may appear to print a solid image, they still work with dots. The dots are substantially smaller than those of a 24-pin dot matrix, but they are still dots. As with video systems, the more dots, the sharper the image. Because a laser is used to change the characteristics of the drum, the areas effected are very small. Therefore, with laser printers, you can get resolutions of even 300dpi on even the least expensive printers. Newer printers are approaching 1,200dpi, which is comparable to photographs.

Some laser printers, like HP’s LaserJet, use a technology called resolution enhancement. Although there are still a limited number of dots-per-inch, the size of each dot can be altered, thereby changing the *apparent* resolution.

Keep in mind that printers have the same problem with resolution as do video systems. The more dots desired, the more memory is needed to process them. An 8 1/2" x 11" page with a resolution of 300dpi takes almost a megabyte of memory to print.

With printers such as daisy-wheel and chain printers, you really don’t have this issue. Even a buffer as small as 8K is more than sufficient to hold a whole page of text, including *control characters* that can change the way the other characters appear. While such control characters may cause the text to be printed bold or underlined, they are relatively simple in nature. For example, underlining normally consists of printing the character, backing up one space, then printing an underline.

Multiple-character sets fonts are something that this kind of printer just can't handle. Different character sets e.g., German or changing the characters form e.g., italic can easily be accomplished when the letter is created "on-the-fly" with dot-matrix printers. All that is needed is to change the way the dots are positioned, which is usually accomplished by using *escape sequences*. First, an *escape character* ASCII 27 is sent to the printer to tell it that the next character or characters is a command to change its behavior.

Different printers react differently to different escape sequences. Although there is a wide range of sets of escape sequences, the two most common sets are those for IBM Proprinters and Epson printers. Most dot-matrix printers can be configured to behave like one of these. Some, like my very old Panasonic KX-P1123, can be configured to behave like either one.

The shortcoming with this is that you are limited to a small range of character types and sizes. Some printers, like mine, can get around this limitation because they can print in graphics modes as well. By viewing the page as a one complete image composed of thousands of dots, they can get any font, any size, with any attribute assuming the software can handle this. This is how printers like mine can print charts, tables, and, to some extent, pictures.

Viewing the page as a complete image works when you have graphics or diagrams, but it's a waste of memory when you're dealing with straight text. Therefore, most laser printers operate in *character-mapped mode*, in which the characters are stored in memory and are the dots are generated as the page goes through the printer.

Printers are controlled by other means than just escape sequences of treating the page as a single image. One most widely used means of control is Adobe Systems Postscript page description language, which is as much a language as the programming languages C or Pascal, with syntax and vocabulary. To use it, both the software and the printer have to support it. However, the advantage is that many applications allow you to print Postscript to a file. That file can then be transferred to a remote site with a Postscript printer. The file is then sent to a printer as raw data and the output is the same as though it were printed directly from the application. The nice thing is that the remote site does not even have to have the same application as long as its printer is Postscript-capable.

Selecting the best printer is more than just choosing the one with the highest resolution and fastest print speed. Although these are two of the most commonly quoted characteristics, they do not represent everything you need to consider.

One commonly overlooked thing is administration. Most business are at a single site, with a handful of people. Even if everyone had their own printers, walking a few feet to figure out what's wrong or make changes is no big deal. However, if you are dealing with dozens or even hundreds of printers, spread out all over the world, physically going to the printer is not always practical.

In many cases the only solution is to physically be at the printer, such as adding paper or changing the toner. You hope that the people on site are capable of doing that much However, there are a number of problems and configuration issues that most users are notable to handle. Since calling in a service technician for mundane issue might be too expensive, it would be able to do some kind of administration remotely.

There are many printers on the market available that have built-in network cards and others can be connected to printer servers to allow you to do certain administrative functions across the network. You simply use telnet to connect to a specific port where you get a command line interface to the configuration options. Although you can generally do all of the configuration across the network that you can do locally, you still have the command line interface, which is typically not all that easy to use.

If you can build a telnet server into the printer or print server, why can't you build in an HTTPD server. Well that's is what Brother did with a number of their printers. Using any standard browser which supports, JavaScript you can administer any of the Brother internal or external print servers.

Their an external print servers are just like many others on the market in that you can most any printer to it. It has both twisted-pair and thin-wire connectors, which allows them to be placed in most any network. In addition, the NC-2100h supports either 10 or 100Mbit Ethernet, making it perfect for high use printers.

The internal print server is basically an Ethernet card built into the printer, with the same connectors as the external version. This are essentially the same products with slightly different constructions, which means the administration is identical. As with the external printer, the Ethernet connector is auto-sensing. In addition, both support a large list of network protocols, including:

lpr/lpd
HTTP
BOOTP
WINS, DHCP
TELNET with user-definable ports
SNMPincl. proprietary MIB
SMTP, POP3
IPX/SPX NDS and Bindery
EtherTalk
DLC/LLC
NetBEUI
NetBIOS support TCP/IP and NetBEUI
Banyan Vines
DEC LAT

One of the most interesting things for me was the inclusion of DHCP. I used network printers from other companies before, which only supported BOOTP. This meant that we either had to configure our UNIX machines to support BOOTP, just for these printers, or configure them by hand. With DHCP, you can configure all of your nodes using just a single protocol.

However, if you look at the list, the Brother print servers are not just limited to specific protocols. Basically, all of the most common protocols are supported, allowing the Brother printers to fit into any network environment. In addition, the Web configuration interface allows you to switch between Printer Control Language PCL and PostScript.

Near top end of the scale is the Brother HL 1660N, which is designed for very demanding businesses. It can provide resolutions as high as 1200x600 dpi, in 256 shades of gray. Although it has a default of only 4Mb of RAM it can be expanded to 66Mb using industry standard SIMMs. This is an important issue, because some hardware manufacturers require you to buy your memory upgrades directly from them, although they are the same as what you buy from other places. The result is that you can pay as much as ten times the streets price just to have the hardware vendors name on it. I realize that many companies make most of their money through after sales service, but this is ridiculous and unnecessary.

The HL-1660N is ready to work amazingly fast. Many printers can take several minutes to warm up, even if just in standby mode. However, the HL-1660N is usually up in about a minute, meaning basically no waiting when you walk from your desk to the printer. Keep in mind that if 10 people a day have to wait an average of three minutes for the printer to warm up, that's 2.5 hours a week or over 500 hours a year!

The speed of printing is also another factor in determining how much time your printer can save. Depending on the amount of text, resolution and other factors, the HL-1660N can get up to 17 pages a minute or just under 4 seconds per page.

The HL-1660N can also help you save paper. When you print, you can tell the printer driver to print in "draft" mode which decreases the resolution. This is useful for seeing exactly how the print out will look or in cases when high quality is not necessary. In addition, it supports 2-in-1 and 4-in-1 printing so you can get multiple pages of your document on a single piece of paper.

For business with less demanding requirement and even for home users, Brother also produces a number of printer with slightly less speed and throughput. For example, the HL-1040 has a resolution of 600 dpi, but only gets about 10 pages per minute. It also includes an internal processor and supports Brothers data compression, thereby increasing throughput.

Brother also produces several color laser printers. The HL-2400C has a resolution of 300x300dpi in color mode and 1200x600dpi mono, with a throughput of 4 and 16 pages per minute, respectively. Once again, throughput is enhanced with an internal processor, this time with a SPARClite and a default of 32Mb RAM. The HL-2400CN is network ready and supports all of the features discussed early including SMTP and POP3 allowing your to automatically print out incoming mail.

If you work with people like some that I do, then you will appreciate the additional security features. The HL-2400C and the HL-2400CN both allow you to block access to the printer based on IP address. Therefore, you won't have certain users blocking the printer by outputting all of those images they downloaded from the Internet.

One group of users whose printer needs are often forgotten as those that are always on the road out of sight, out of mind. If they are visiting a customer site, for example, it is either embarrassing or cumbersome to get the customer to make a print out for them. Therefore, it would be nice to have a portable printer. Many vendors provide solutions which require cumbersome parallel cables and the inconvenience of a bulky power-supply.

Brother's answer to this is the MC-21P series of "mobile" ink jet color printers. Connectivity to the printer for both data and power is provided by a PCMCIA card. Although it can only get about 2.5 pages per minute, the convenience far outweighs the delay in getting your print out. In addition, the MC-21P can print on transparencies, as well as plain paper, which helps you make last minute changes to your presentations, reports and so forth.

From a business perspective it is important to look at having a single company satisfy all of your printing needs. The larger your company the greater the need is. With a hand full of printers, the need is not as great. However, I can speak from experience when I say how hard it can be to manage a large number of different kinds of printers.

Keep in mind that you not only need to deal with different drivers, but with different quality of printouts and different materials i.e. toner cartridges. In addition, there is the issue of support. You need to keep track of different warranty information, different support numbers, as well as different problems. If you have discovered how to solve one specific problem on one printer, you will end up having to call to another vendor when the problem arises on a different printer.

One thing Brother printers and other devices emphasize is straight-through printing. This can make them slightly larger than similar devices from other vendors. However, I get annoyed when my pages come out with a slight curve to them.

Brother also provides multi-function printers, which are slightly different than their multi-function centers. As with the multi-function centers, these provide printer, scanner and copier functionality, but no fax or other communication. The MFC-P2000, for example, is a laser printer, which gets up to 10 pages per minutes with a resolution of 600x600, which is perfect for the small or home office. It can scan at the same resolution and comes with a copy of the Xerox TextBridge OCR soft. So, what do you get when you combine the functionality of a scanner with that of a printer? A copier, which is the third function the MFC-P2000 provides. It, too, has a built-in processors and warms up in under a minute.

Keep in mind this is not all that Brother has to offer. I barely touch on what printers and multi-function device are available. If I hadn't I would have needed an entire book. To find the exact printer to suit your needs, check out the brother web site www.brother.com

There are also a number of ink jet printers that brother produces. At the high end of the scale is the HS-5300. This gives you a resolution of 600x600, with a quality that is extremely close to a laser printer. It too comes with a built-in processor and default 24 Mb RAM, but can be increased up to 72Mb. As an upgrade option, you can get it with the NC-2010H network card, which then gives it the same functionality as any of the other Brother network capable printers.

It too, has the built in compressor of the driver, which helps increase speed across the network. In addition, the ink cartridges are replaced through a panel in the front of the printer. No need to open up the cover and deal with the cartridge attached to the print head.

One important aspect that I often see overlooked with printers is the total cost of ownership. Some companies will consider it for their computers, but often overlook it for their printers. The reason is often that many people are not aware of what aspects can increase the total cost

of owning a printer. One important aspect is the expendable materials that have to be replenished at regular intervals or the parts that can wear out and need to be replaced.

Let's take a laser printer as an example. As you print, you use toner and eventually you will need to replace the toner cartridge. Normally, there is at least one cartridge provided by the vendor when you first purchase the printer. Therefore, you may not be aware of how much a new toner cartridge costs. In many cases, it can be anywhere from $50 to $100, or more. The more often you have to change the toner cartridge the more you pay in total for the toner and the more the total cost of ownership.

Let's take two theoretical printers. One costs $300 and the other $500. Assume that both have the exact same quality and speed, and each can print 10,000 pages before the toner needs to be replaced. You might think that $300 printer is less expensive. Let's assume that the toner cartridge for the $300 printer costs $70, but the one for the $500 printer costs only $50. It has either 10 or 100Mbit Ethernet interface, making it reasonable to expect the printer to last 3 years and in that time, you also expect to print over 200,000 copies. This means, you will need to buy twenty new cartridges. With a price difference of $20, that means you will pay $400 extra for the toner cartridges for the less expensive printer. Therefore, the more expensive printer actually comes out cheaper.

Unfortunately, the calculations are not always as easy as that. Often the total number of pages you can print with a single cartridge will differ from printer to printer. Therefore, you need to first make an estimate of how many pages you will print during the expected lifetime of the printer and then calculate how many cartridges your will need. In addition, you need to find out how long parts like the drum will last before it needs to be replaced. This also adds to the total cost. When you have done your calculations, the best choice is the printer that has the lowest cost per page.

This is one place where I often seen people complain about Brother printers. If you are using your printer at home with just a couple of dozen pages per month, then perhaps many of the Brother printers are not for you. However, once you start getting toward hundreds or thousands of pages per month, this is where Brother printers become extremely attractive. In some cases, Brother printer can be as little as half as much per page.

Another problem I often see is buying generic toner or ink. As with other products, generic or less known printer supplies are often cheaper than their brand-name counter parts. I intentionally used the word "cheaper" here, as such products often take on the other meaning of "cheap." For example, I have found many vendors who sell ink for ink-jet printers that has a lot higher water content than the ink from the printer vendor. It doesn't dry as quickly and therefore produces a less than acceptable printout. Maybe it's okay for a draft, but nothing you would want to send to a customer.

However, this is not always the case and it often depend on the paper. Therefore, you might want to test a single cartridge and package of paper before you buy them in bulk.

With color printers another place to save money is if it is a color printer and there are separate cartridges for each color. If your company logo has red letter on a yellow background, then you might end up using more yellow and magenta. The cyan cartridge could be almost full,

but you end up having to through the ink away if there are separate cartridges.

You should also look into refills for the ink cartridges. This usually allows you to refill a specific color, without having to replace the entire cartridge. However, this can be a messy job if you are not familiar with it. In addition, how easy it is to use the refills will be different from vendor to vendor. If you only do refills a few times a year, the savings compared to buying completely new cartridges may not be worth the hassle.

8.15 Mice

The basic principle is that by moving the mouse, the cursor (pointer) on the screen moves in the same manner. Actions can be carried out by clicking one of up to three buttons on the mouse.

As the mouse is moved across a surface, a ball underneath rolls along with it. This ball turns small wheels (usually three of them) inside the mouse. The amount each wheel turns is measured and this movement is translated into the movement of the cursor.

Because the ball underneath must roll for the mouse to work, it has to remain on a flat surface. The surface must also have a certain amount of friction for the ball to roll. Although you can get a certain amount of movement by shaking the mouse, picking it up and expecting the cursor to move is a waste of time.

Originally, mice were connected by a thin cable to the computer. As technology has progressed, the cable was done away with and replaced with a light-emitting diode (LED) on the mouse and a photodetector near the computer. This has the advantage of preventing the cable from tangling or getting buried under a pile of papers and thereby limiting the mouses movement. The disadvantage is that the LED must remain within the line-of-sight of the photodetector to function. Some manufacturers have overcome this disadvantage by using an alternate form of light that does not depend on line-of-sight: radio.

Another major problem with all of these kinds of mice is desk space. My desk is not neat. Space is at a premium. Even the small space needed for a mouse pad is a luxury that I rarely have. Fortunately, companies such as Logitech have heard my cries and come to the rescue. The solution is, as an old UNIX guru called it, a dead mouse.

This is a mouse that lies with its feet (or, at least, the ball) sticking up. Rather than moving the mouse to move the ball to move the wheels to move the cursor, you simply move the ball. The ball is somewhat larger than the one inside of a mouse, which makes it a lot easier to move. Such a mouse is called a trackball and is very common with laptop computers. Provided the signals sent to the operating system are the same, a trackball behaves similarly to a mouse.

The mouses interface to the operating system can take one of three forms. The mouse is referred to, based on this interface, as a *serial mouse*, *bus mouse*, or *keyboard mouse*.

As its name implies, a serial mouse is attached to your computer through a serial port. Bus mice have their own interface card that plugs into the bus. Keyboard mice, despite their name, usually do not plug into the keyboard. Though I have seen some built into the keyboard, these were actually serial mice. Instead, a keyboard mouse is plugged into its own connector,

usually next to the keyboard connector, which is then attached directly to the motherboard. These mice are usually found on IBM PS/2 and some Compaq computers, though more computer manufacturers are providing a connector for a keyboard mouse.

When people talk about the movement of the mouse, you often hear the term *resolution*. For a mouse, resolution is referred to in terms of clicks per inch, or CPI. A click is simply the signal sent to the system to tell it that the mouse has moved. The higher the CPI, the higher resolution. Both mice and trackballs have resolution, because both rely on the movement of a ball to translate the movement of the cursor.

Keep in mind that despite how it appears at first, a mouse with a higher resolution is not necessarily more precise. In fact, almost the opposite is true. Higher resolution means that the mouse moves *further* for each given movement on the ball. The result is that the movement is *faster,* not more precise. Because precision is really determined by your own hand movement, experience has shown me that you get better precision with a mouse that has a lower resolution.

8.16 Uninterruptable Power Supplies

The first thing I want to address here is the concept of uninterruptable power. If you take that term literally, a power supply that goes out at all has been interrupted. In that case, many UPS are not correctly named because there is a brief moment (ca. 30 milliseconds) between the time the computer notices the power has gone out and the battery kicks in. This time is too short for the computer to notice, but it is there. (Normally, power must be out for at least 300 milliseconds before the computer will notice.) As a result, most UPS should be referred to as stand-by power supply (SPS) because they switch to the battery when the primary supply shuts off. Because Underwriters Laboratories uses UPS to describe both, that's what I will do here.

The basic UPS provides limited power conditioning (keeping the voltage within a specific range) but no protection against surges and spikes. This is useful if the power goes out but doesn't protect you if the voltage suddenly jumps (such as the result of a lightning strike). A *double-conversion* model provides the power when the main power fails and also provides protection against surges by first passing the power through the batteries. Although this does provide protection, it is less efficient because power is constantly drawn from the battery.

Although no UPS vendor directly supports Linux, there are several sources for programs that can take advantage of existing UPSs. The best place to start is the UPS HOWTO.

8.17 Cases

A computer case is more that just something that protects the inside of you're your computer from spilled coffee and other dangers associated with the typical computer user. It also protects you from the inside of your computer. The 110 volts flowing through the power supply (220 in Europe and other places) is not something you want to meet up with unexpectedly.

In addition, the computer case is both the skin and skeleton for your computer. It holds everything together and hold them in the proper location. You motherboard is attached to the case, so the expansion slots need to be in the proper position when you insert a card, the connectors on the end of the card are sticking out of the computer and not pointing in.

Fortunately, there is a standard in both motherboard and case design that helps keep things from pointing in the wrong direction. However, idiots are getting smarter ever day, so this system will never be idiot proof.

One of the first things to look at is the physical construction of the case. Cases come in many, many, many different sizes and shapes. PCs cases are normally though of in terms of desktop, mini-tower, tower and maxi-tower, or something similar. As its name implies, a desktop case is one that is intended to sit on your desktop. The smaller the better. In fact, a new term "small footprint" case is used to refer to cases that are even smaller than traditional desktop cases (XT, AT, and Mini-AT).

These cases lay flat on your desk and often have a set of four compartments (or bays) arranged in a 2x2 square to hold hard disks, floppy drives and CD-ROM drives. The obvious limitation is expansion. The motherboard is often smaller and therefore there is less room for expansion cards. Plus, there is not much extra room in the case if you wanted to add too much more.

Tower cases stand vertically, with the bays one on top of the other. With mini-towers, the case is small enough that it might fit on your desk as this is generally just a re-arrangement of the desktop case. That is, you have room for four devices. Big towers are just that. There are usually at least six bays for devices which could have some access slot at the front (i.e. floppy-drive, CD-ROM, tape drive, etc.). However, this space could also be used for hard disk, as the cases are usually provided with some kind of plate covering the front. In addition, there is room for a couple more hard disk.

However, this is a classic case of not judging a book by its cover. I have seen large cases before that had some really goofy construction. The result was few bays and a lot of wasted space.

In the last couple of years, server cases have made it to the market, which are a lot larger than tower cases. This are normally the same height, but can be several times wider. There is room for a dozen or more hard disk and many are designed explicitly with the goal of have a large number of disks that you set up in some kind of redundant array (RAID). However, in every case I have seen, these were delivered with a pre-built system.

There are a couple of key aspects to consider when looking for a case. First, the case and the motherboard need to fit together. There are a number of motherboard types and for the most part, the holes in the motherboard match those for the case. There are usually more hole than you need and some combination is likely to fit.

However, what you need to consider it that the motherboard should not lay completely flat against the motherboard. It needs to be spaced above the case so that none of the metal on the case can touch the motherboard. This ensures that no short circuit is possible. Along with all of the motherboards I have purchased in recent years have been nylon spacers, which keep the

motherboard away from the metal frame of the case.

The next aspect is the placement of the bays for the hard disks. I have met my share of cases, where you almost need to take the case apart in order to get to the bays. If this is the case, you can end up spending a lot of time replacing the drives and a lot of money on band-aides for your scraped knuckles.

If you have a tower case and can use one of the front-loading bays, it makes it much easy to get the drives into the bays. However, it may make getting the cables onto the drives is another problem. Both of these issues become mute (for you) if you have someone else doing the maintenance. If, however, you are the ones building the machines or replacing defect drives, how hard it is to get access to the bays is an important consideration. DEC did it right for a few years until something caused them to change their minds. Drives were inserted into the case sideways and not front to back. Although they still laid flat, there was no problem getting the drives in and out. There were typically enough slots for four to six drives and several pairs brackets were provided that screwed onto the hard disk. With the brackets in place, the drives slide in, locked into place, with the plug end sticking out. It was also a simple matter to remove the drives when you need. Unfortunately, just prior to being purchased by Compaq, DEC changed the construction of their cases and getting at the drives was almost impossible. In fact, I have yet to see worse placement than I saw on some of the cases.

Also consider how easy it is to open and close the case. There should be no "trick" to opening closing the case. I have had some where you had to apply just the right amount of pressure on the top and bottom of the side panels that slid into the case from the back. First, this meant having to move the case far enough away from the wall so you had room to pull the sides off. Plus "just the right amount of pressure" was harder than it sounds.

The next important issue is cooling. Unless you are running an old 486, you probably have at least the fan on your CPU (as well as the one for the power supply). However, that may not be enough. All electronic devices generate heat and it can build up pretty fast in the enclosed space of your computer case.

The most common way for the computer cases to get cooled is simply letting the hot air escape out the holes in the case. However, I have seen places were the computer is hidden under a desk, with almost no circulation at it overheated. Even if the computer is in the open, you may still not have enough circulation inside the computer, particularly if you have a lot of peripherals in your machine. If this is the case, you might want to consider getting a fan card. As its name implies, it is a fan that sits in one of your expansion slots and helps circulate air.

You should also consider the design of the case. I have seen many where an extra fan was built into the case that blew directly onto the hard disks. In other cases, an additional fan would blow onto the CPU.

8.18 The Right Hardware

Purchase

There are three different ways of obtaining your workstations and servers. Each as their own and advantages and disadvantages. The first is to build all of your machines from scratch. This can be cheaper in terms of actual cash outlay for the hardware. However, you also need to consider that a lot more time will be spent putting the pieces together. This can mean that the actual cost is higher.

The second alternative is to buy whatever computer happens to be on sale when you need it. This decision saves the time of putting together the machine as well as saves you money. The downside is that you may end up with as many different configurations as you have machines. This means additional administrative work to manage all of these different configurations.

Finally there is drawing all of your hardware from a single vendor. This helps standardized your machines, while at the same time shifting a lot of the work to the vendor. In many cases you may not work with the vendor directly, but rather a distributor or reseller. In essence this has the same advantages and disadvantages of working with the vendor directly.

Which of these methods you choose will depend a lot on your unique circumstances. Although building machines from scratch may not be the best solution for smaller companies, you may discover a financial advantage in purchasing then all from the same vendor (i.e. economies of scale). In order to get the best deal, you'll need to negotiate with each vendor individually. What additional services are provided vary dramatically from company to company. Because it requires a fair bit of administrative work, let's look at some of the issues involved in dealing with a single vendor.

You can obviously save yourself a great deal of administrative work by simply buying the same brand and model from your local computer superstore. However, this defeat the purpose and loses many of the benefits of going directly to through the vendor or through one of their official resellers.

The first thing that you will be missing is having a personal contact within the company. Although it is possible to get an individual account representative when buying through a large chain, you typically do not get the same level of personalized service. This is important issue in that part of what you're trying to do is to shift much of the responsibility onto your supplier. It is possible that the local retailer is in a position to support you directly (e..g in-house tech support), so it is worth investigating.

There is a large variation in the sales program and services the vendors will provide. This not only depends upon the vendor themselves, but also of the size of *your* company. You are obviously in a much better position to make "demands" of the vendor if you're planning to purchase several hundred machines. However, even in smaller units you have a great deal of negotiating power. Remember there are dozens of computer vendors. If the first one cannot (or will not) meet your needs, the logical alternative is to go somewhere else. Depending on your area, you may even find companies which will even build the machines for you according to your exact specifications.

One thing that is often forgotten is that buying the computer outright is just one alternative. Some companies will rent the computers to you. Others will lease them. Still others have programs in which you can buy the computers at the end of the lease. Although this is slightly

more expensive then buying the computers outright due to the amount you need to pay in interest, you can end up saving time end money, because the vendor is responsible for getting the computer repaired (just like a leased car). Depending on your contract, the vendor may be obligated to provide you with a replacement while your computer is being repaired.

Another benefit in purchasing from the vendor is that you commonly get quantity discounts. This does not mean you're required to buy in large number of computers at once, rather you need to buy them within a specific time frames such as six months or a year. In addition, the software may also be part of the deal. Normally, you can't lease the software, but you can spread out the purchase price of the software over the lifetime of the lease. (although this is less of an issue with Linux)

One thing to look at the leasing contract is whether or not hardware upgrades are included in the lease. One possibility is that you can upgrade the hardware for a moderate payment. This keeps you up-to-date without making you go bankrupt.

One important thing to keep in mind is that standard machines from brand-name companies are often not as standard as one might think. I once worked on a server from DEC (first Compaq and then HP) which had a defective drive in a mirror set. When the replacement drive was sent, we discovered that it was smaller than the original although it had the exact same part number, the drive was from a different manufacturer who had a slightly smaller size than the other (although both were sold as 4 Gb). If the drive had been bigger, it would not have been a problem. However we could not simply recreate the mirrors set after adding new drive. Instead, we had to install and format the new drive, copy all the data to it, and use it as the base for the new mirrors set. This meant that some of the space on the older drive was lost, but more importantly it cost us great deal of down-time.

This example is not intended to demonstrate problems with brand-name computers, nor is it intended to put DEC/Compaq/HP in a bad light. Instead, it is intended to point out that there are things which you need to look out for. In this case the new drive was not an adequate replacement for the old one. If the vendor is not willing to provide an appropriate replacement, then you might want to consider looking for a new vendor.

The steps to take when a machine or component needs to be replaced should be *formally* documented. Every administrator should be in the position to replace or repair machines as needed, this is not to say that every administrator has the ability to get out and fix damaged cards. Instead, the administrators should be a position to get the machine repaired. Your company may be different in that there is a special group which is responsible for repairing PCs. However the principal stays the same. As long as defined procudures are followed, they should be able to get the machine repaired.

Another important aspect of standardization is defining exactly what information will be kept, where it will be stored and in which format the data will be in. If different groups used different formats to store their information, sharing becomes difficult and in many cases impossible. This may be as simple as creating an database to which everyone has access, but you also need to ensure that the databases are all compatible.

Part of this is determining who has what access to what information. Although everyone on your help desk (if you have one) should at least be able to read this information, is not absolutely necessary that they be able to update or change it. If multiple people are all allowed to change this information, then the procedures also need to be documented.

Depending on the number of users in your company, you probably will end up with several different kinds of standard workstations. Each will depend on the type of work that is done on that machine. For example, you might have machines that are used solely to input data into databases and others which are used for your technical applications. By defining a standard you are likely to find that repairing or replacing the machines is much easier. You know which components go into that machine so you can simply order that exact same part. You know what the standard workstation is so you do not need to waste time trying to figure out what the replacement machine should need.

Because your business depends on getting work done, it is not unreasonable to expect timely deliveries of your computers. Therefore, you should consider including penalties for late delivery in the contract. Some vendors can take six weeks or longer to deliver the machines. If you plan delivery on a specific date and that date is not met you may be out of some money. Why shouldn't the vendor take responsibility?

In one instance we were repeatedly promised delivery to one of our branch offices by a specific date. Since the installation was being done by administrators from the head office, plane and hotel reservations needed to be made and work schedules needed to be adjusted. The day before the scheduled departure the vendor announced they would not be able to deliver on time (although we had ordered six weeks earlier). Since we planned well enough in advances, we got a substantial discount on all our airline tickets. However, two days before the flight the tickets were neither refundable nor could they be exchanged. This was money that was lost due to problems within the vendors organization and therefore they should be responsible for the damages.

Fortunately, this contract had penalties for late delivery. The contract went so for as to stipulate that all machines had to be up and running for the contract to be considered fulfilled. For each machines that was not running the specific penalty would be applied. Here we had ten PCs that were already delivered, but the server was not. Since this was a Windows NT domain and we needed the server to do the installation and allow the user to to work, it meant that neither the server nor any of the workstations were operational. Bad planning on the part of the vendor resulted in a 10 fold penalty.

Even if you do not lose money because of non-refunded airline tickets, you do lose time and sometimes money when deliveries are not made on time. If users are waiting for the machines, they cannot work as effectively as they should be able to. You have increased administrative costs for the time you spend tracking down the status of the order. If the machines are replacements for existing computers, you may have the problem of different users running different software. These problems can be compounded when you have a large number of machines spread out across multiple deliveries. You have the extra administrative burden of insuring everything is delivered properly.

In such cases, it is in your best interest to have the contract stipulate what things cause a contract violation and what the penalty is in each case. And should also stipulate what your responsibilities are, for example, making arrangements with an ISP.

When deciding on a vendor, one thing to ask is how long they will guarantee the delivery of specific equipment. That is, how long will they ensure that a specific model is available. It defeats the purpose of standardization if you get different machines with each order. One vendor I have worked with changed models every three or four months. With a 6 to 8 week delivery time, there was a high probability that orders made two months apart would have machines that were not the same model.

An argument in favor of the vendor might be that this was necessary in order to keep up with the technology. To some extent this is true, but it makes managing standard PCs extremely difficult. If you are contractually obligated to purchase a specific number of machines within a specific period of time it is not unreasonable to expect that be vendor is obligated to provide you the same product during the lifetime of the contract. Besides, in a business, you are usually less concerned with being on the "bleeding edge."

Due to the frequent changes in the computer market, it might not be possible to *completely* guarantee the availability of machines over the course of the year. Therefore, you'll need to work closely with your vendor to see that they come as close as possible.

We next come to the magic phrase total cost of ownership (TCO). Eventually the computer will need to be repaired. The question is what kind of service are you getting from the computer discount store as compared to the major vendor? Even if you are not buying hundreds of computers at a time of you can still get good support from the major vendors. Obviously, the quicker and more detailed the support the more you are liable to pay.

However, I need to warn you here. Just because you're buying workstations or servers from one of the major vendors it does not mean that what you're getting will be consistent. One prime example is the example with the hard drivers mentioned above. In such cases, you cannot always be sure that the components are consistent even within a particular model. That is the description of two machines may be identical. However, they could have different motherboards, different video cards, or different hard disks. This makes mass installations of machines extremely difficult. For example if you are expecting a particular video card to be in a machine, the installation may not work correctly because of the different video card.

Another advantage of having a single vendor is that you have a single point of contact. If something goes wrong, for example a computer breaks down, you know exactly who to call. On the other hand, if you have machines from different manufacturers, you need to first check to see her the manufacturer is of the computer in question. In addition, if you have warranty or maintenance contracts, you'll need ten separate contracts, one for each of the different vendors. A side benefit of having a single vendor with a single maintenance or support contract is that you build up a relationship with the company. Depending on the company, this can meet to sometimes getting more benefits that are then are defined in your contract.

One thing that we have found out is vital when ordering hardware is that the invoice should be as detailed as the original order. For example, if you ordered and Adaptec 2940 PCI host

adapter , then this should be listed on the invoice. In some cases I have received invoices that simply stated that there was a SCSI host adapter. This is not enough, especially later when you need to make warranty claims. Also, simply stating that there is 32 MB of RAM is not enough. The invoice should state exactly what you get like 60 ns EDO RAM.

Sometimes buying hardware is more than just finding the fastest or largest for the best price. Often, it is a question of "piece of mind." It's like accident insurance. You know to be careful, but sometimes accidents happen. That's why it's called "accident" insurance. In essence, you are betting against yourself. However, the costs of being wrong are not worth the risk. Therefore, you pay a little extra each month, "just in case." Even if you never have to use it, there is the piece of mind of knowing that if something were to happen, you would be covered.

I take this same approach to hardware. For example, I will never again put an IBM hard disk in any machine. In the space of about a year, I had three different IBM hard disks, all with similar problems. The first was a SCSI drive. It repeatedly reported read and write errors. I brought it to my dealer, who kept it for several days to "test" it. After three days of not being able to work, I got the same drive back with the report that the dealer could not recreate the problem.

When I put it back in the machine, I got the same problems. At that point the first thought was a defective controller or cable, but the drive reported the same problems on a completely different machine.

Next, was an IBM IDE drive. I brought it home, pluged it in and it wasn't recognized at all. By that time the dealer was closed, so I had to wait until the next day. In this case, they could easily recreate the problem, since it wasn't recognized on their machine either. They gave me a new one which I installed. A couple of months later the same read and write errors started appearing as with the IBM SCSI drive (also on different machines).

One day when I turned on the machine, the system reported that it could not find the primary hard disk (which was supposed to be the IBM drive). Although I could boot from a floppy and access the hard disk, I could not read anything from the root directory. When I ran CHKDSK.EXE (It was a Windows machine), I was left with about 3 directories that were not damaged. All others were now in the root directory with names like DIR00001.

Needless to say, this forced me to re-install my system, dozens of applications and reconfigure the entire system. This cost me two days (actually two evenings) until I was back at a place where I could work effectively. However, it was more than a week until I had re-installed all the applications.

Two months later, it happened again. The primary hard disk was not recognized and I could not access the root directory when I booted from a floppy. A low-level format revealed a dozen bad tracks. Obviously after the low-level format I had to re-install the system, my applications and reconfigure everything.

This taught me three important lessons. First, even if the data is on a different system and backed-up regularly, a crash of the primary hard disk means a lot of time is wasted reinstalling everything. Therefore, you need some mechanism to quickly reinstall and

reconfigure your system. Some of these techniques we will get to in later chapters.

The second lesson I learned is to get a dealer that won't waste my time "testing" the drive and then return it to me when they cannot find any problems. If the problems are intermittent, the dealer should expect that the problem cannot be re-created so easily. In my opinion, the dealer should simply give you a replacement and be the one who has to deal with the manufacturer. For private customers, the dealer may not be so willing to go out of his way. However, if you are business spending hundreds of thousands of dollars a year, they *must* be willing to make the extra effort.

The last lesson is never to put an IBM drive in my machine again. IBM "claims" that they have a large number of drives in all sorts of machines and do not have these problems. There are probably many of you, who are reading this who have not have had the same experience. However, I am not willing to take the risk, just like I am not willing to take the risk of driving without insurance. The consequences of being wrong are two high.

Every single IBM drive I have had has exhibited some kind of problems which put my data at risk. Even if a drive from another vendor costs $20 or ever $50 more than one from IBM, I am willing to pay it for the piece of mind of not having to worry about whether the system will boot the next time I turn it on or that the data actually gets written to the drive. Even though I know people who have not have the same problems, I want the piece of mind.

An additional advantage is the learning curve is much steeper with hardware from single vendor. This is because the construction of the machines and components are generally the same. Again, this is not an absolute. Because the hardware is similar with each machine, the administrators and users do not need to re-learn each time a new piece of hardware is acquired.

On the other hard, sticking with a single vendor can become a trap.Can you really wait six weeks for your machines? In one company where I worked, we stuck with a single vendor who could not promise delivery in less than six weeks. The IS manager thought it was more beneficial to have 100% standard PCs than having a PC at all. Even though the machines differed from country to country and sometimes even between deliveries, it was pretty much standardized within each office.

In one company they had the best of both worlds. PCs were grouped into classes with similar characteristics. However, being 100% identical was not as important as having a machine. Therefore, the company stuck with three vendors. If the first one could not deliver fast enough, they went to the second.

Note that this requires a little discipline on the part of the administrators. They needed to keep track of which machine had which configuration. However, with any of the asset management tools available on the market this is fairly easy.

Going to three vendors also has the added advantage of being better able to dictate prices. Obviously, if you must have the machine within specific amount of time, you will probably end up paying more. However, if you are willing to wait a little, you might be able to cut a good deal. For example, if one vendor says they can deliver in four weeks and another says they can deliver in two weeks, you might be able to get a good deal by waiting the extra two

weeks. Remember: supply and demand. You are supplying the money, so you can make demands.

Repair

You might find it useful to develop a "maintenance pool" for your workstations. This pool consists of complete PCs as well as copies of the different kinds of hardware. Standardization of your machines means the you need to have fewer different models and different brands of hardware. This allows you to choose between replacing the entire machine or just individual components.

In some cases it is not always necessary to have exactly matched hardware such as hard disks. However, with devices which require a specific driver, such as hard disk controllers or network interface cards, it is in your best interest to standardize this as much as possible.

If your hardware pool is designed to swap out the defective machine, repair it, and then return it to its original owner, you can decrease the number both spares you need body using removable media. Your maintenance pool consists both of middle to high-end machines. Each with their own removable media drive. When you needed to exchange machines it is not matter who needs the machine, but you simply take one from the shelf and stick in the appropriate drive.

The downside of this is that it requires much greater standardization of hardware and there is the additional expenses of the removeable media drives. These have a much higher cost per megabyte as standard hard disks. Alternatively, you could simply swap out the hard disks and save on the cost. The downside of this is the additional time it takes to install the harddisk. On the other hand, as hardware prices continue to sink having a few extra computers on hand probably isn't enough to make you go bankrupt.

If you have a maintenance contract with your computer vendor, you should really investigate how things are handled internally weigh all in advance of your first call. In some cases, the sales and support organizations may have the same company name, but are actually separate entities, which have a hard time communicating with each other. I have experienced it myself that after purchasing a large number of computers from one vendor, we could not support from them, because the sales organization had not yet gotten around to submitting the necessary paperwork.

Depending on how much the salesperson pushes the after sales service aspects of their company, you might want to try getting a penalty clause built and should the vendor not repair or replaced the equipment within the time they promise. When they say that a particular service contract "guarantees" a specific response time, what does this actually mean? If a tangible product such as a computer fails to meet the guarantee, you can always return it. How do you return a promised level of support?

Just like when the computer is not delivered on time, a non-functioning computer can cost you money in terms of loss of productivity. If you don't pay for the computers like you promise the vendor can simply repossess it. What *your* options when the vendor does not meet their obligations? You couldn't go to another vendor the next time, but you already lost the money. If the vendor wants your business, some kind of tangible penalty should be applied if they do

not fulfill their obligations.

The repair or replacement of defective machines is another place where you can save time, at least from the users perspective. This is one of the strongest motivations for having standard PCs. Should the machine go down you can swap the machine in just a few minutes allowing the user to get back to work. You can then repair the machine or replace defective components at your leisure.

In some companies, assets such as computers are assigned to specific departments or even individuals. Therefore, you may need to return that computer to its original owner. However, if that's not the case, the repaired machine can then be used as a replacement the next time a computer goes down.

If your workstations are standardized, it is much easier to keep a stock of spare parts on hand. Swapping a defective component is often as cost-effective as replacing entire machines, especially if you do not have the money for machines which are not regularly being used.

Regardless of how you obtain your hardware, you should always have spares on hand. This is not as easy when you get your computers for major vendors, as it is when you build the machines yourself or order them to be build for you. Often times when you order from major vendors, the description may simply be something like "3.4 GB SCSI hard disk". This tells you nothing about the manufacturer. (Although with hard disks this is often less of a problem.)

However, with other pieces of hardware the difference becomes more critical. For example, if your network card fails you would probably want to replace it with the same kind of card. This enables you to simply remove the old card and insert the new card without having to install a new driver.

I need to emphasize the need to have the spares on hand as opposed to ordering them from the vendor. Even if your vendor promises a one-day turnaround, this usually means one business day. Therefore, if the hard disk dies on Friday, you may not be able to continue work until Monday. If Monday is a holiday, you may not get it until Tuesday. Your business week may not be the same as the vendors.

If your company uses standard computer configurations, there may actually not be any need to have spares of individual components. Instead you might want to consider having spares of entire machines. In one company I worked, we had a few spares for each of the different classes. When one computer failed, we simply replaced it with a comparable machine. Since all of the data was stored on the server, the only thing the user lost was local configuration settings. However, using NIS for users and group names and some cases NFS, even that was kept to a minimum. The result was we were able to replace the computer in less than 10 minutes. That meant the user was back to work in 10 minutes. If we were replacing components, they can take anywhere from a half an hour to two hours, or even longer.

Which method is most effective will depends on your company. The larger the company, the more likely you would want to employ a system by which you replace the entire computer. This is simply because there are more problems occurring in a larger company. You will need to make replacements more often. Therefore you will need to limit the downtime for the users.

If you do decide for systems where you swap components, I would recommend having several spares of each type of components. Once again, standardization of hardware is vital to make the system more efficient. You must be able to swap out like components. You do not know in advance when a component will fail and which component it will be. What do you do if you have only a single spare and the same component decides to break on two machines on the same day?

The odds are generally low for such things, but how much money do you stand to loose if you are wrong? If you have standard machines from a single vendor (or just a few vendors), the vendor may have bought a whole batch of defective parts. Remember, they are trying to keep their costs low and do no always put in brand names. The more components that are defective, the greater the likelihood that even more will have problems.

What you do with the defective computer will be determined by your company's policy. In some companies, the computer is repaired and the old one is returned to the user. This is because there is a 1:1 assignment between computers and the users. In other companies, where there is no 1:1 assignment, there is no need for the old computer to be returned to a specific user. Therefore, it is returned to the computer pool and waits to be used as a replacement somewhere else.

8.19 HW Diagnostics

Hardware Diagnostic Tools

Since the world is not perfect you will eventually have to deal with a crashed system. In many cases, how the system behaves when it boots or doesn't boot will give you an indication of what is going on. However, it also will happen that there is nothing that specifically identifies the problem. It is also possible that your system boots fine, but exhibits odd behavior has it is running. The most common solution for this kind of problems on Windows machines is to re-install. However, this only corrects the problem if is related to the software. What about hardware problems?

There are a number of hardware diagnostic tools on the market. Some run under Windows, whereas others have their own "operating system" which you can boot, allowing you to directly access the hardware. Those that run as stand alone products, typically have a much wider range of tests they can conduct because they are not limited by the operating system. Keep in mind that this more that just reporting the IRQ or base address of the devices. These products actually test the various components of your system.

Personally, I think you should use tools which run under the operating system in conjunction with stand-alone products. It is possible that you might get incorrect results if you are running under any operating system as it often "interprets" the information for you. Although this is useful for "configuration" issues, defects and other problems are often missed.

There are also a few products that come with interface cards that are inserted to the bus, allowing you to diagnose problems even when your system cannot boot. These have a small, digital display on the card which shows you the post code being sent across the bus. Based on the code, you can determine where the problem lies.

In general, the software products have a common set of tests they run through. The tests normally include:
System Board
Video Alignment Aids
Video Adapter
Parallel Port
Serial Port
Floppy Disk Drive
Hard Disk Tests 0 & 1
Main Memory Tests

One of the key features to look at is the extent to which you can configure these tests. This might mean defining a specific set of tests to run, as well as how many times to run each test. Both are important aspects. If you already have an idea of where the problem is, you should not have to wait for the program to run through unnecessary tests. Also, with hardware you often have sporadic problems. Therefore, you might have to run the test continually for an extended length of time before the problem re-appears.

Another thing to look at is what values or configuration settings can be changed. Keep in mind that changing settings is not always a good thing. Particularly if a novice is running the tests.

Micro2000

If you are concerned with diagnosing PC hardware problems, take a look at the wide range of products that Micro2000 has to offer. The products range from self-booting diagnostic tools to POST reader cards to remote diagnostics and beyond.

Micro-Scope is their self-booting diagnostic tool that can run on any PC. Regardless of the CPU manufacturer Intel, Cyrix or AMD or bus ISA, EISA, MCA, PCI, and PCMCIA, Micro-Scope can identify problems on your PC. Version 7 the newest, as of this writing contains tests for your CD-ROM drive, without the need to load DOS-based CD-ROM drivers. Something which many other diagnostic tools do not have. In addition, the version 7 also contains support for the AMD K6-II and Intel Xeon processor, even those with a clock speed above 500Mhz. Upgrades for new processors are available for download from the internet.

Many tools simply report on the problems they find. However, Micro-Scope not only allows you to make changes, but also gives you detailed benchmarks of your system. This is useful when you "feel" something is wrong with your machine, but there is no identifiable hardware problem. With the report generated by the benchmark, you can see if the machine is performing as it should.

During the testing, Micro-Scope examines the CMOS and POST information. Anything that is inaccurate or questionable is flagged, allowing you to change it as needed. Part of this is being able to accurately identify your hardware, including brand name and model. This is extremely useful when buying brand name computers, which normally do not tell you exactly what components you have.

Microscope supports all common bus types including ISA, EISA, PCI and Microchannel. You can even display the POS registers on IBM PS/2 systems, including all slots, which adapters are in which slot, which ADF adapter description file to use and whether the ADF is loaded.

In addition to being able to diagnose CD-ROM problems, Micro-Scope can test many other multi-media components, such as DVD drives and sound cards. It has full synthesizer tests and can test the volume and left-right channels of your sound card.

Tests can be run once or repeatedly. The results of which can either be printer out or saved to disk or just viewed on-screen if you want. In addition, you can use the printscreen capability to print directly from the application.

As with other products, Micro-Scope will thoroughly check your memory, using all of the common tests checkerboard, walking-ones, etc.. Low memory is tested before the entire program is loaded, which is then relocated in memory to enable you to test all of your memory, regardless of how much you have. In addition, Micro-Scope will tell you exactly what bank is failing. This includes the ability to test internal and external system cache, as well as video RAM up to 64Mb.

Another bonus is the tools Micro-Scope has for data recovery. It can identify and correct many problems in the master boot record of your hard disk. It also includes an editor to allow you to make changes your self anywhere on the disk assuming you have the knowledge to do it.

In addition, to free download of patches, Micro-Scope comes with lifetime technical support. After using the program, I find it difficult to conceive of a reason why someone would need to call to support, as it is so intuitive, but the offer is nice. The product package contains both 3.5" and 5.25" disks, a use’s manual, as well as 9 pin serial, 25 pin serial, and 25 pin parallel loopback connectors, to diagnose serial and parallel port problems.

Unfortunately, something like Micro-Scope cannot always do the job. This happens when your system just won’t boot for any number of reasons. Using a diskette with it’s own operating system does no good, because the computer does not get that far to boot from anywhere. This is where Micro2000’s product POST-Probe comes in handy.

As it’s name implies, POST-Probe monitors the POST codes being sent across your system bus as the computer is booting. It can fit into any ISA, EISA, PCU or MCA slot although it requires the included adapter for the MCA. These codes are displayed on two seven-segment displays, indicating what the POST is testing at the moment. There are also four LEDs which monitor the power, as well as four voltage pads +5vdc, -5vdc, +12vdc, -12vdc and an additional 3.3V for PCI to test the system using a voltmeter.

There are additional LEDs which monitor clock signals, the RESET signal and I/O reads and writes. You can therefore use POST-Probe after your system is running to identify other bus problems and possible problems with specific cards.

When your system stops, the last code displayed gives you an indication of what is wrong. Although the code does not always tell you the exact place where there is a problem, the included user’s manual lists these phases of the POST. By looking at the steps around where it

stopped, I have never not found the problem. In one instance, I accidentally loosed up the cable to my hard disk. When I tried to boot nothing happen. Using the POST-Probe I quickly found the problem.

As I will talk about in later chapters, I am a stickler for documentation. I am really impressed with the POST-Probe manual. It is written in an easy to understand language. POST failure codes are on the left side of each page, with the description the device or chip that is causing the problem. This helps finding and understanding the problem.

For the true profession, Micro200 has combined Micro-Scope and POST-Probe into a single product, which they call the Universal Diagnostics Toolkit. Both products are combined in the full version within a case, which is not only large enough to hold both product, but tools and many other things. Each product as the same lifetime technical support as the stand-alone versions.

Micro2000's product Burn-In takes much of the functionality of Micro-Scope to the next level. As it's name implies, it is used to conduct "burn-in" tests of computers. This can be either new machines or ones that you have repaired. This is an extremely useful tool to prevent deploying products that will only cause you problems down the road. Particularly in cases where machines have multiple problems and only one is correct, burn-in tests can save you a great deal of both time and money.

Like Micro-Scope, Burn-In is compatible with all CPU manufacturers and system buses. In addition, Burn-In performs all of the same tests that Micro-scope does.

Burn-In has a couple of very useful features for companies that install a larger number of PCs at once. First, the tests can be run without a monitor or keyboard. Therefore, you need a lot less space allowing you to simply stack up the PCs and run a large number of tests at once. Using the floppy drive light and speaker, the program send a few signals to the technician when it needs a "scratch" disk or the loopback plugs. Other than that, the program runs completely on it's one, saving the results to disk.

As the tests are run, Burn-In writes a complete log to the scratch disk you provided. Since the log is ASCII, you can read it with any text editor. In addition, the log is being update the entire time. Therefore, if something should happen to the machine like someone accidentally pulling the plug, Burn-In will be able to continue where it left off.

In addition, you only need to run the setup once. The test configuration is then saved and performed the same way each time the disk is booted. If the program determines that hardware is not present for a test is was selected to do, that test is simply skipped, without the need to configure the test for different hardware variations.

TuffTEST

TuffTEST from Windsor Technologies is a powerful and very inexpensive stand-alone diagnostic tool. Although you could order it with all of the packaging, you can save time, money and trees by ordering and then downloading it from the web. As of this writing it is just $9.95, which is a fraction of most other products.

One key aspect is that it is designed specifically for users with less experience. Although it has most of the features of high-end tools, the emphasis is on ease of use, as well as providing the user with sufficient information to diagnose the problem.

This is a stand-alone product, in that it can be booted from a floppy. This sounds confusing at first, because you download it from the Internet. What you download is a program which allows you to create the bootable floppies. Once booted, TuffTEST "takes over" the computer, without the need for an operating system like DOS or Windows. As I mentioned before, often this yields more accurate results. TuffTEST has its own set of device drivers, which can access hardware directly.

Windsor boasts that TuffTEST is "safe for use by anyone." This is because none of the tests change data on the hard disk. In addition, the program is so configure that once it boots, it will wait 10 seconds for a menu selection and if no key is pressed it runs through the complete suite of tests.

Another advantage of TuffTEST is that it is complete written in assembly language which means more compact code, and faster execution. In addition, it take up just 125K of memory, which is actually relocated when then program runs. This ensures that every memory locations is tested. In other cases, the program is actually too large to be able to check all of memory.

TuffTEST is not just a diagnostic tool as it can also display all of your hardware configuration information. This information can then be printed or saved to the disk. Each saved session contains the test results as well as the system configuration. Since you can save up to five previous sessions, you can compare the results from multiple tests.

Higher up on the scale is TuffTEST PRO, this is intended for the professional. This has the same basic functionality plus you can edit your configuration and make other changes to your system. Like TuffTEST. TuffTEST PRO is a stand-alone product, meaning you boot your operating system from the diskette and it becomes your operating system.

In addition, there are a number of tests that TuffTEST PRO has that are not included in TuffTEST. For example, TuffTEST PRO can report in the switch positions on your motherboard, conduct I/O tests on your serial and parallel ports, determine the optimal interleave and low-level your harddisk, and many other tests. Using the optional loopback test, you can do I/O tests on your serial and parallel ports.

One of the most interesting aspects of TuffTEST is sales approach. You can order a packaged version f the product, including a printed manual , if you feel it is necessary. However, there really is no need. The on-line manual contains all of the necessary information, plus the product is extremely intuitive.

Lifetime support is provided for free. However, the product is so easy to use it is hard to think of reason why you would need to call them. In addition, updates range from free for minor changes to a slight fee for major new releases.

Other Goodies

The PC Technology Guide

If you feel you are overwhelmed by all the different hardware technologies, you're not alone. Quite rankly there are a lot of different kinds of hardware and it is extremely difficult to be really familiar with all of it, especially if you have other duties to perform. However, at one time or another you're going to need to know about different kinds of hardware, so you will either need to learn all yourself, hire someone who already know it all, or find a source of information that can deliver the necessary information efficiently.

In this context, efficiency has two key aspects. The first is related to speed. If it takes until tomorrow to get the information you need like being through stacks and stacks of old magazines, it is not very efficient. Even if you know that the information is in that stack "somewhere," it does you little good if you spend hours or even days looking for it.

Second, efficiency is also related to the amount of information available to you. This does not mean you need access to all of the material ever written about PC hardware. Instead, the opposite is often more true. That is, sometimes the information available is just too much. You cannot see the forest for the trees. If the amount of information keeps you from getting at what you need, it is not very efficient.

I often encounter "information overload" when I'm confronted by new technology. I'm interested in earning what it's all about and browse the Web site of a vendor who offers this technology. It is loaded with technology briefs, press releases, white papers, and many other documents describing the technology. Unfortunately, more often than not where I would expect to find a description of the technology, I end up with what equates to nothing more than the marketing blurb about how wonderfully this company has implemented the technology.

Even if there are technical papers about the technology, I'm often confronted with a problem with new technology that there is often not enough background information to understand what the papers talking about. So I have to go running to some other site to find out the technical background to understand technical background of this new technology.

Therefore, when I came across the PC technology guide Web site www.pctechguide.com, I was amazed to find so much useful information in one place. The site explicitly states it is"aimed more at the PC hobbiest than the IT professional." However, I feel that the authors "aim" is off as this site provides the wealth of information for the hobbiest and the IT professional.

One key issue that is overlooked by many people is just what "IT professional" means today. As late as the mid-90s, and IT professional had to be a jack of all trades. You had to know about hardware, operating systems, networking, programming, and all of the other areas which falls under the heading of "information technology." Today, people within the IT profession are becoming more and more specialized. It is not unreasonable to find an outstanding database administrator, who knows little about the underlying hardware. He or she does not need to know this as the operating system should provide the necessary level of abstraction.

Therefore, something like the PC technology guide provides an outstanding resource for anyone including IT professionals who need to find out more information about the hardware technology aspects of IT. Plus, it is loaded with detailed images, which help you to understand the technology being discussed. Being Web pages has the extra added advantage of having links to different places on the site. Therefore, you do not need to go hunting defined details. You just click on the link and you are there.

The site's author, Dave Anderson, has also considered the wishes of people who want continual access to the information. Dave provides the PC technology guide on CD, which you can order on the Web site. This is definitely well worth the money. For less than the cost of any good book on the subject, you have immediate access to the information you need. Since it only takes up about 25 MB, including all of the information stored in MS-Word documents, there is no reason not to copy it to your hard disk.

Although a separate browser is provided with the product, which allows you to access the pages, preferred using a separate browser such as Netscape Navigator. By purchasing a site license, you could easily link in the PC technology guide into your Intranet, making it available to all of your users. In addition, regular updates of the CD are provided.

Chapter 9 Networking

Long ago (at least in terms of the history of electronic data processing) having two computers at the same time was something you read about in science fiction novels. As systems became more common, the time eventually arrived when a company or university would have two computers. The need then arose that data be exchanged between the two machines. This was the beginning of SNEAKER-Net (Slow, Non-Electrical Activity, Keeping Everyone Running), which was developed in the 1950s. With SNEAKER-Net, the technician copied data onto a tape or other media and, using his sneakers, ran the tape over to the other machine to be loaded. In many organizations, SNEAKER-Net is still employed today as this is often the only type of network some people think they can afford.

In 1976, researchers at AT&T Bell Laboratories came to the rescue. This was the development of a serial line protocol to exchange data between UNIX machines, which came to be known as UUCP, for Unix-to-Unix Copy. Over the years there were several changes, upgrades revisions, etc. In 1983, AT&T released a new version that came to be known as Honeydanber UUCP, as it was developed by Peter **Honey**man, **D**avid **A**. **N**owitz and **B**rian **E**.**R**edman.

Although, UUCP was a good thing, system speed was limited by the serial line connecting the two computers, the slowest component of the system. Since the system could only be as fast as its slowest component, there needed to be a way to speed up that slowest component. Well, serial line speeds increased, but that still was not enough. In the 1970s, Xerox came out with *Ethernet*, which made high speed communication between computers possible. It was now possible for users to access remote computers and expect response times comparable to being logged in locally, rather than experiencing delays as was common with the serial line communication of the day. (We'll get into more details on Ethernet later.)

Today, both are still widespread. Although prices have dropped to the point that Ethernet networks are commonplace (I even have one in my house), UUCP is still used regularly when distances prevent other types of connection, or when the connection is going to be quick or short term and the administrator doesn't want the added hassles of first installing the Ethernet cards.

Unfortunately, going into details about UUCP is beyond the scope of this tutorial. Therefore, I leave it to you to take a look at the UUCP HOWTO if you are interested in configuring UUCP.

9.1 TCP-IP

Before we talk about the details of networking, we should first talk about the process of network communication. Let's take a network program such as telnet. The telnet program allows you to login to a remote system. You end up with a shell just as if you had logged in locally. Although you are inputting commands on your local keyboard and the output is appearing on your local screen, all other activity is happening on the remote machine.

For simplicity's sake, we can say that there is a telnet program running on each computer. When you are inputting input something on local keyboard, the local copy of telnet is accepting input. It passes the information through the network to the telnet on the remote machine. The command is executed and the output is handed to the remote telnet. That information is passed back through the network to the local telnet, which then displays the information on your screen.

Although it may appear as if there is a constant flow of information between your local machine and the remote one, this is not the case. At any given time there may be dozens, if not hundreds of programs using the network. Since only one can use the network at a time there needs to be a mechanism to allow each program to have its turn.

Think back on our discussion on the kernel. When we need something from the hard disk, the system does not read everything at once. If it did, one process could hog the computer if it needed to read in a large file. Instead, disk requests are send in smaller chunks and the program only thinks that it gets everything it wants. Something similar is done with network connections.

Computers are like humans beings in that they need to speak the same language in order to communicate. Regardless of how they are connected, be it serial or Ethernet, the computers must know how to talk to each other. The communication is carried out in a pre-defined manner, called a "protocol". Like the protocols diplomats and politicians go through, computer protocols determine how each side behaves and how it should react to behavior by its counterpart. Roughly speaking even the interaction between the computer and the hardware, such as the hard disk, can be considered a protocol.

The most common protocol used by UNIX variants, including Linux, is TCP/IP. However, it is more accurate to call TCP/IP a protocol suite, or protocol family. This is because TCP/IP actually consists of several different protocols. Even the name consists of two different protocols as TCP/IP stands for Transmission Control Protocol/Internet Protocol.

TCP/IP is often referred to as protocol *suite* as it contains many different protocols and therefore many different ways for computers to talk to each other. However, TCP/IP is not the only protocol suite. There are dozens, if not hundreds of different ones, although only a small portion have gained wide acceptance. Linux only uses a few itself, although the TCP/IP family is what is delivered by default and most commonly used.

Although the name refers to two specific protocols, TCP/IP usually refers to an entire suite of protocols and programs. The result of many years of planning and discussion, the TCP/IP suite includes a set of standards which specify how computers ought to communicate. By following these standards, computers "speak" the same language and can therefore communicate. In addition to the actual means of communication that the TCP/IP suite defines conventions for connecting different networks and routing traffic through routers, bridges and other types of connections.

The TCP/IP suite is result of a Defense Advanced Research Projects Agency DARPA research project on network connectivity. However, its availability has made it the most commonly installed network software. Many versions provide source-code which reside in the

public domain allowing users to adapt it to many new systems. Today, essentially all vendors of network hardware e.g. bridges, routers support the TCP/IP suite as it is the standard protocol suite on the Internet and in most companies.

Whereas the data being transferred to and from the hard disk is talked about in terms of blocks, the unit of information transfer across a network connection is referred to as a *packet*. Depending on the program you are using, this packet can be a different size. In any event they are small enough to send across the network fast enough, so that no one process hog the network. In addition, the packets go across the network so fast that you don't notice that your data is broken in to packets. This is similar to the way the CPU manages processes. Each one gets a very small turn on the processor. Because it switches so fast between processes it only seems like you have the processor to your self.

If we take a step back and look at the process of network communication more abstractly, we see each portion supported by and supporting another. We can say that each portion sits on top of another. Or in other words the protocols are *stacked* on top of each other. Therefore, TCP/IP is often referred to as a *protocol stack*. To see how these layers look graphically, take a look at Figure 0-1.

Each portion of the stack is referred to as a *layer*. At the bottom of the stack is the layer that is responsible for the physical connected between the two computers. This is the physical layer. Sitting on top of the physical layer is the layer that is responsible for the network portion of the stack. That is, it ensures that packets either stay on the network or get to the right network and at the same time ensures that packets get to the right network address. This is the network layer.

On top of the network layer is the layer that ensures that the packets have been transmitted correctly. That is, there are no errors and all packets have been received. This is the transport layer. Finally, at the top of all of this is the layer that the user sees. Since the programs that we use are often called applications, this upper layer is called the application layer.

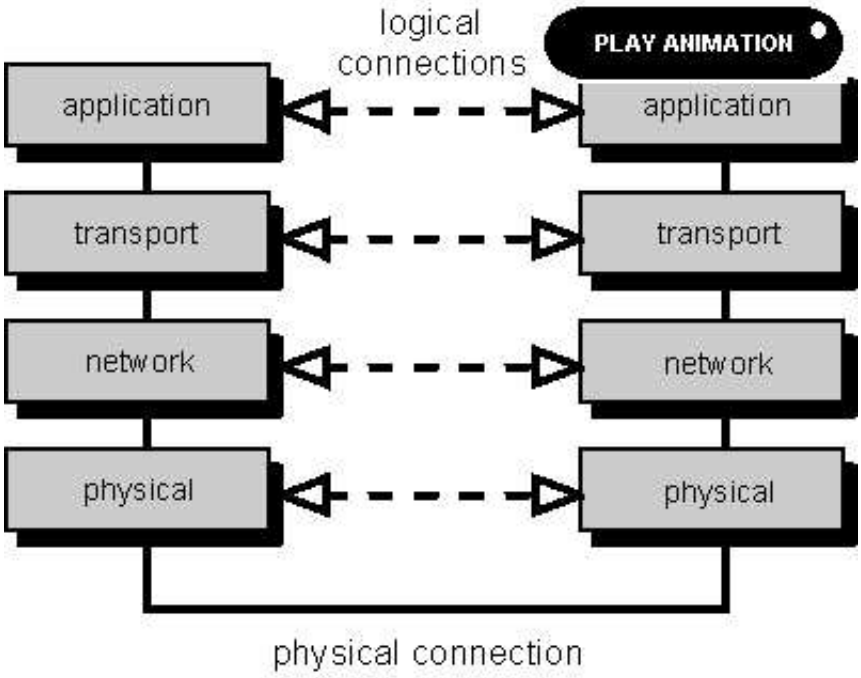


Image - Network Layers

Conceptually, each layer is talking to its counterpart on the other system. That is, telnet on the local machine is passing data to telnet on the remote machine. TCP on the remote machine sends an acknowledgment to TCP on the local machine when it receives a packet. IP on the local machine gets information from IP on the remote machine that tells it that this packet is destined for the local machine. Then there are the network interface cards that communicate with each other using their specific language.

This communication between corresponding layers is all conceptual. The actual communication takes place between the different layers on each machine, *not* the corresponding layers on both machines.

When the application layer has data to send, but prepends an *application header* onto the data it needs to send. This header contains information necessary for the application to get the data to the right part of the application on the receiving side. The application then calls up TCP to send the information along. TCP wraps that data into a TCP packet, which contains a *TCP header* followed by the application data including header. TCP then hands the packet also called a TCP *segment* to IP. Like the layers before it, IP wraps the packet up and prepends an *IP header*, to create an IP *datagram*. Finally, IP hands it off to the hardware driver. If Ethernet, this includes both an Ethernet header and Ethernet trailer. This creates an *Ethernet frame*. How the encapsulation looks graphically, take a look at Figure 0-2.

As we see, it is the TCP layer that the application talks to. TCP sticks the data from the application into a kind of envelope the process is called *encapsulation* and passes it to the IP layer. Just as the operating system has a mechanism to keep track of which area of memory belongs to what processes, the network has a means of keeping track of what data belongs to what process. This is the job of TCP. It is also the responsibility of TCP to ensure that the packets are delivered with the correct contents and then to put them in the right order.

Encapsulation is show graphically in Figure 0-2.

Error detection is the job of the TCP *envelope* which contains a checksum of the data contained within the packet. This checksum information sits in the packet header and is checked on all packets. If the checksum doesn't match the contents of the packet or the packet doesn't arrive at all, it is the job of TCP to ensure that packet is resent. On the sending end, TCP waits for an acknowledgment that each packet has been received. If it hasn't received one within a specific period it will resend that packet. Because of this checksum and the resending of packets, TCP is considered a *reliable connection*.

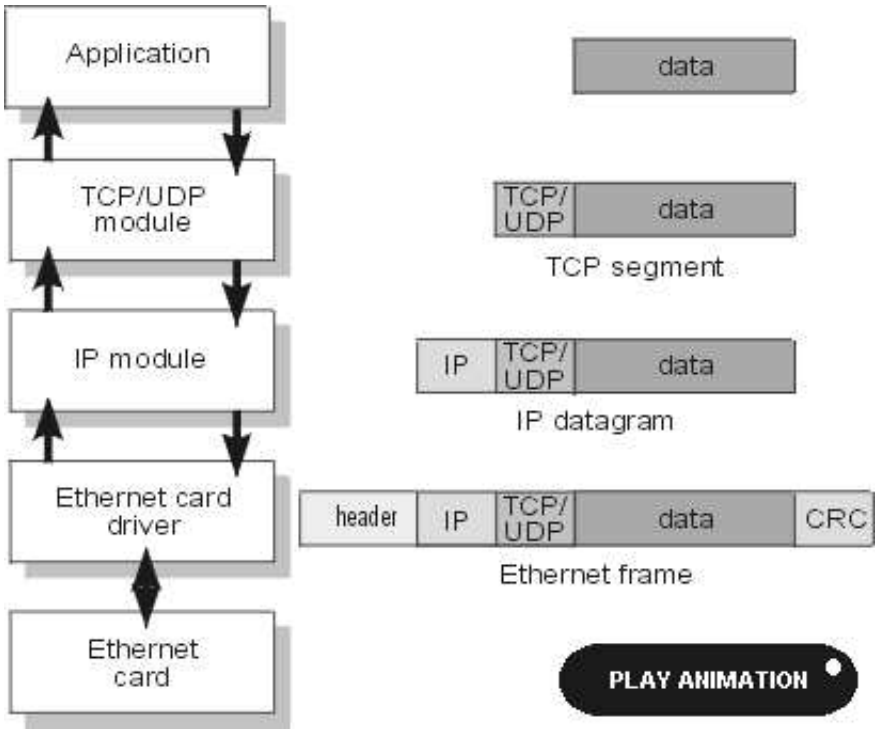


Image - Encapsulation of data

Another protocol that is often used is the User Datagram Protocol UDP. Like TCP, UDP sits on top of IP. However, UDP provides a *connection-less* transport between applications. Services, such as the Network File Service NFS, that utilize UDP, must provide their own mechanism to ensure delivery and correct sequencing of packets. Since it can be either broadcast or multicast, UDP also offers one-to-many services. Because there is no checking by UDP it is also considered unreliable.

Closest to the hardware level, IP is a protocol that provides the delivery mechanism for the protocols. The IP layer serves the same function as your house addresses, telling the upper layers how to get to where they need to. In fact, the information used by IP to get the pieces of information to their destination are called IP addresses. However, IP does not guarantee that the packets arrive in the right order or that they arrive at all. Just like a letter to your house requires it to be registered in order to ensure that it gets delivered with the content in-tact, IP depends on the upper layers to ensure the integrity and sequencing of the packets. Therefore, IP is considered *unreliable*.

Since the hardware, that is the network cards do the actual, physical transfer of the packets, it is important that they can be addressed somehow. Each card has its own, unique identifier. This is the Media Access Control, or MAC, address. The MAC address is a 48 bit number that is usually represented by 6 pairs of hexadecimal numbers, separated by usually dashes or colons. Each manufacturer of network card is assigned a specific range of addresses which usually are specified by the first three pairs of numbers. Each card has its own, individual address: the MAC address.

When sending a packet, the IP layer has to figure out how to send the packet. If the destination is on a different physical network, then IP needs to send it to the appropriate *gateway*. However, if the destination machine is on the local network, the IP layers uses the Address Resolution Protocol ARP to determine what the MAC address of the Ethernet card is with that IP address.

To figure this out, ARP will broadcast an ARP packet across the entire network asking which MAC address belongs to a particular IP address. Although every machines gets this broadcast, only the one out there that matches will respond. This is then stored by the IP layer in its internal ARP table. You can look at the ARP table at any time by running the command:

arp -a

This would give you a response similar to:

```
siemau 194.113.47.147 at 0:0:2:c:8c:d2>
```

This has the general format:

```
<machine name> IP address at <MAC address> >
```

Since the ARP table is cached, IP does not have to send out an ARP request every time it needs to make a connection. Instead, it can quickly look in the ARP table to make the IP-MAC translation. Then, the packet is sent to the appropriate machine.

Status and error information is exchanged between machines through the Internet Control Message Protocol ICMP. This information can be used by other protocols to recover from transmission problems or by system administrators to detect problems in the network. One of the most commonly used diagnostic tools, "ping", makes use of ICMP.

At the bottom of the pile is the hardware or link layer. As I mentioned before, this can be represented by many different kinds of physical connections: Ethernet, token-ring, fiber-optics, ISDN, RS-232 to name a few.

This four layer model is common when referring to computer networks. This is the model that is most commonly used and the one that I will use through the book. There is another model that consists of seven layers. This is referred to as the OSI model, but we won't be using it here.

9.1.1 IP Addressing

In today's world of inter-connected computers, you may have a connection to hundred of thousands of other machines. Granted there is no single cable connecting all of these computers, however there is a logical connection in that you can use the telnet program from your PC in California and connect to a machine in Germany. The problem is, how do the packets get from one end to another. Added to that, how do you keep your local network in California from getting overloaded with packets that are being sent between machines in Germany and at the same time making sure that those telnet packets do get through? The answer is provided by the Internet Protocol (IP).

Just as a street address is not always sufficient to get your letter delivery, so is the IP not always sufficient to get the packet delivered. If I sent you a letter, it could be sent to a single, central post office, whose job it was to distribute mail throughout the entire US. Because of the incredibly large number of pieces of mail, this is impractical. Instead, there are thousands of offices, all over the country, whose job it is to route the mail for us.

If we lived in a small town, the local post office could catch a letter destined for a local address before it goes further. Mail with addresses outside could be sent to other post offices to be processed.

A similar situation applies to IP addresses. In local, self-contained networks, the IP address alone is sufficient. However, when multiple networks are combined, machines spend more time trying to figure out if the packet belongs to them than actually processing information. The solution is a network mask. Just as a zip code tells a postal worker whether to process a particular piece of mail locally or not, the network mask (or netmask) tells machines whether or not they can simply ignore a packet or need to process it further. How this works, we'll get to in a moment.

Every machine on the network, needs to have its own, unique IP address. Just like every house has a unique mail address. If that network is connected to the rest of the world, that address must not only be unique within the local network, but unique within the rest of the world, as well. With the most common IP version (IPv4), IP addresses are 32-bit values. They are usually represented by four sets of numbers, ranging from 0-255 separated by dots (.). This is referred to as *dotted-decimal notation*. In dotted-decimal notation, an address might look like this:

147.132.42.18

Since each of these numbers range between 0-255, they can be represented by eight bits and are therefore referred to as an *octet*. This IP address is often thought of as being composed of a network portion (at the beginning) and a node (or machine) portion at the end. This would be comparable to writing a street address as:

95061.Main_Street.42

Where 95061 is the zip code and Main Street is the street and 42 is the address on that street. The reason we write the street address in this fashion, is that it's common to think of the IP address as moving from the general to the more specific.

Currently, there are three classes of networks in common use, which are broken down by both the range used in the first octet and the number of octets used to identify the network. Class A networks are the largest and use the first octet as the network address. Networks in the class will have the first octet in the range 1-126. Class B networks used the first two octets, with the first being in the range 128-192. The smallest networks, class C use the first three octets in the network address and with the first in the range 192-223. How IP addresses are broken down by the different network classes is shown in Table 0\1.

Class	Range within 1st octet	Network ID	Host ID	Possible networks	Possible hosts per network
A	1-126	a	b.c.d.	126	16,777,214
B	128-191	a.b	c.d	16,384	65,534
C	192-223	a.b.c	d	2,097,151	254

Table - IP Address Breakdown by Network

There are a couple of things I would like to point out about this table. First, the network address 127 represents the local computer, regardless of what network it is really on. This is helpful for testing as well as many internal operations. Network addresses 224 and above are reserved for special purposes such as multicast addresses. The terms "possible networks" and "possible hosts per network" are those that are calculated mathematically. In some cases, 0 and 255 are not acceptable values for either the network address or the host address. However, 0 can be used in a network address for either the second or third octet (for example, 10.2.0).

Keep in mind that a Class A address does not necessarily mean that there are 16 million hosts on a single network. This would be impossible to administrate and would over burden most network technologies. What normally happens is that a single entity, such as Hewlett-Packard is given a Class A address. They will then break it down further into smaller *sub-nets*. We'll get into more details about this shortly.

A network host uses the network ID and host ID to determine which packets it should receive or ignore and to determine the scope of its transmissions (only nodes with the same network ID accept each other's IP-level broadcasts). Because the sender's IP address is included in every outgoing IP packet, it is useful for the receiving computer system to derive the originating network ID and host ID from the IP address field. This is done by using subnet masks, as described in the following section.

In some cases, there is no need to have unique IP addresses, since the network will *never* be connected to the rest of the world. For example, in a factory where the machines communicate with each other via TCP/IP. There is no reason for these machines to be accessible from the Internet. Therefore, there is no need for them to have an official IP address.

You could just randomly assign IP addresses to these machines and hope that your router is configured correctly not to route the packets from these machines. One slip and you have the potential for not only messing up your own network, but someone else's as well.

The solution was provided in RFC-1918. Here, three sets of IP address were defined for use in "private" networks. These won't be routed and there is no need to coordinate their use with any of the registrations agencies. The IP addresses are:

10.0.0.0 - 10.255.255.255
172.16.0.0 - 172.31.255.255
192.168.0.0 - 192.168.255.255

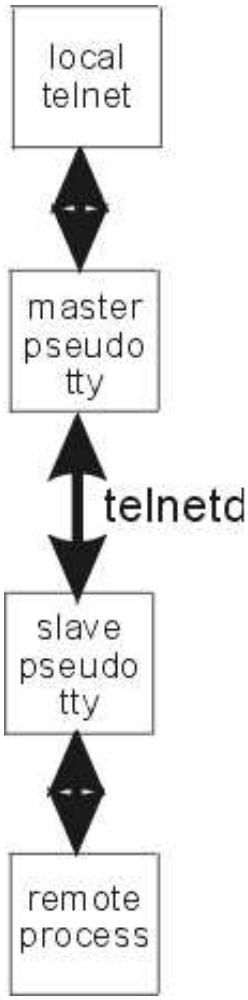
As you can see that there is just a single class A address, but 16 class B and 255 class C networks. Therefore, no matter what size your network is, you can find a private network for your needs.

9.1.2 Pseudo Terminals

The telnetd daemon is a server which supports the telnet program. Makes sense huh? Telnet is a terminal program that allows you to work interactively with remote machines, just as if you would with the local machine. When inetd receives a incoming telnet request, it invokes telnetd.

What you then see is no different that if you had logged in locally to that machine (probably). You are presented with a login: prompt, you enter you logname and password. If these are correct, you then are given a shell that you can enter commands starts applications, etc.

The way telnetd works is that it allocates a pseudo-terminal device for you. This pseudo-terminal has the same behavior as a "normal" terminal in that you input commands and see the results on your screen. Internal the pseudo-terminal is broken down into two parts. The master portion is the side that you see. Since your side is the one that is controlling things, your side is the master. The master side accepts input from your telnet program and passes them to telnetd on the remote side. As you might guess, the side that has to listen to the master is the slave. The slave side of the pseudo-terminal serves as stdin, stdout, and stderr for the remote application.



Pseudo-ttys(interactive)

Similar in functionality to telnet is rlogin. The server for rlogin, is rlogind, and like telnetd, is started by inetd. One of the primary differences is that, if configured, rlogind can provided a connection without the normal login procedures.

The functionality of rlogind is very similar to that of telnetd. Pseudo-terminals are allocated and the slave portion becomes the stdin, stdout, and stderr. During login, rlogind uses an authentication procedure called "host equivalence", which sets up remote machines as being "trusted". If rlogind on the destination machine authenticates the source machine, the user is automatically logged in. If the authentication fails, the user must go through the normal login procedure. How to set up host equivalence, we'll get to later.

9.1.3 Network Services

In the discussion above, I used the telnet command as an example of one of the programs that use TCP/IP. However, there are many others which provide additional services such as transferring files, electronic mail, networking printing, and access to remote filesystems. Other products, such as database applications may have one central machine containing all the data and access is gained from the other machines via TCP/IP. Often this access is invisible to

the user who just sees the "front end" of the database.

This configuration, where one machine contains the data or resource that an other machine uses is very common in computer networking. The machine with the resource that it is providing to other machines is referred to as the *server*, because it is serving the resource to the other machine. The machine that is using the resource is called the *client*. This model, where one machine is the server and the other is the client is referred to as a client-server model.

Another common network model is the *peer-to-peer model*. In this model, there is no one central machine that has all the resources. Instead, all machines are on equal status. Often times, these two models sort of blend together. In Linux networks, it is possible to have multiple servers, each providing many of the same resources. In can also happen that multiple machines all have resources that the others need so everyone is acting as both a client and a server, similar to peer-to-peer, which is common in Microsoft Windows networks.

On Linux systems, there are dozens of resources available. Many of which are well-known such as telnet, others, such as ntp are more obscure. Like calling into a large office building with a central switchboard, our server needs to know what numbers are associated with which programs in order to make the proper connection. In the same regard, you need to know what office you want to reach before you call. In some cases you can call and say you want a particular extension. In other cases, you say you want a particular office. In a office building there is a list of available "services", called a phone book. On a Linux system the phone book is the file **`/etc/services`**.

The **`/etc/services`** file contains a list of what services a particular machine may have to offer. The concept of a service is slightly different than the concept of a resource. A machine may provide many resources in the form of login shells that it provides to remote users, however all of them are accessing the machine through the one service: telnet.

In addition to what service the machine provides, **`/etc/services`** also lists the port. To understand the idea of a port, think about this as being the telephone number. When I call in to a machine say using telnet, I am connected to the telnet program on the other side through a particular port. This is as if I were calling a large office building with a single switchboard. When I reach that switchboard, I tell the operator which office or person I want to talk to. In the ancient history of telephones, that operator had to make the connection between the incoming line and the office herself.

A port can also be thought of as the socket that the operator plugs the phone lines into. Like in that office building, there may be a set of these sockets, or ports, that are directly connected to a specific person i.e. service. These are *well-known* ports. There may be offices with their own operator maybe just a receptionist who passes the incoming phone call to the right person or may even pick someone themselves to take the call such as when you call a government agency with a generic question and there is no one person responsible for that area.

On a Linux system using TCP/IP, the principle is the same. There are dozens of services that one can connect to, but only one way into the system, that’s through your network interface card. In order for you to be able to connect to the right service, there has to be something like

an operator to make the connection for you. This is the program **/etc/inetd**. This is the "Internet Daemon" and often referred to as a "super server" since it is inetd's responsibility to wait for requests to access the other servers on your system and pass you along.

Like in our office building, you may know what number you want, that is, which port. When you make the connection to inetd, your process tells it what port you want to connect to and inetd makes the connection. On the other hand, you may have a program that does not have a well-known port. Therefore a new port needs to get created.

The inetd daemon "listens" for the connections. You can say that it is listening on multiple ports in the sense that it manages all the ports. However, it is inetd that makes the connection between the incoming connection and the local port, and therefore to the local server. This mechanism saves memory since you don't need to start up the servers you aren't going to use. This is similar to having a central switchboard and not requiring every office to have their own. You can see how this looks graphically here:

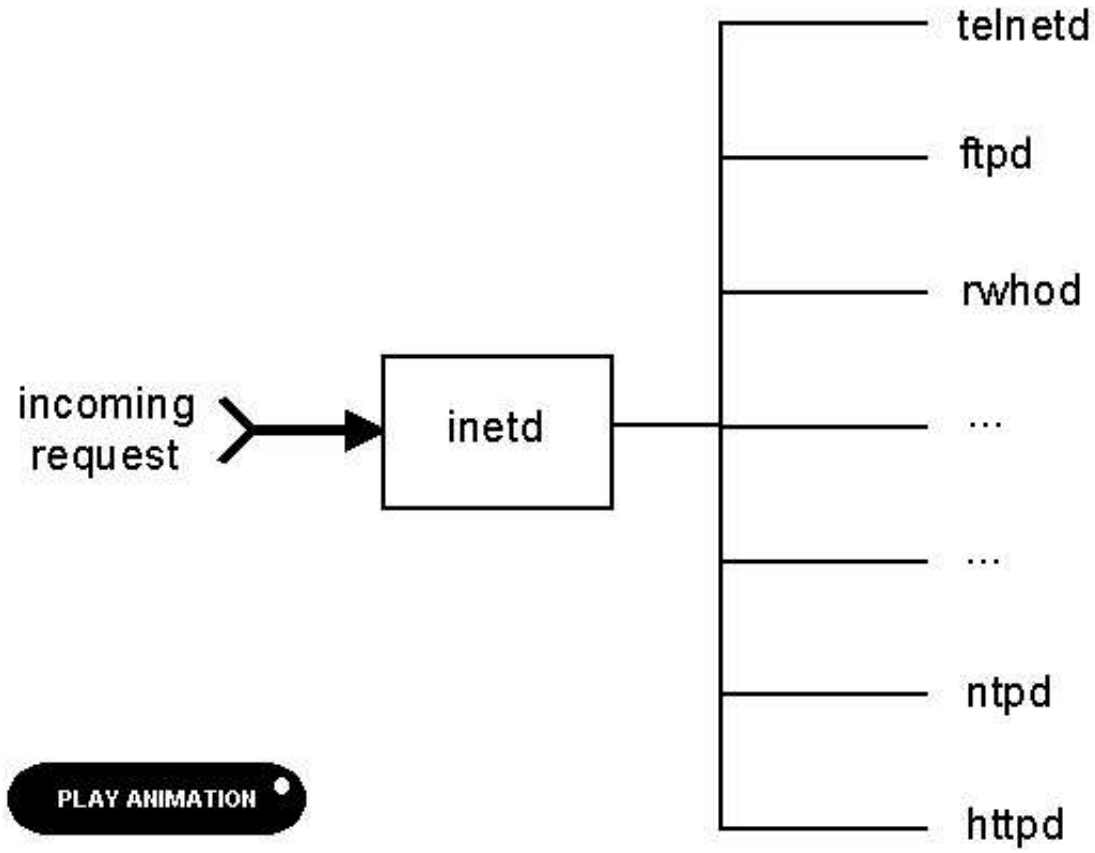


Image - Graphical representation of the inetd daemon

Normally, inetd is started during system start up from a script under **/etc/rc.d**. When it starts, inetd reads its configuration file **/etc/inetd.conf** to obtain the necessary information to start the various servers. It then builds the logical connection between the server and its respective port. Kind of like laying the cable from the central switchboard to the various offices. Technically it creates a *socket*, which is *bound* to the port for that server.

When **inetd** gets a connection request the phone rings for a connection-based port, it "accepts" the incoming call which creates a new socket. That is, there is a logical connection between the incoming request and the server. Inetd can now continue to listen on the original port for addition incoming calls.

If the port is connection-less UDP, the behavior is dependent on entries in the **/etc/inetd.conf** file. If inetd is told to wait there is a wait in the fourth column, then the server that was called must process the incoming message before inetd can go on. If told not to wait there is a nowait in the fourth column, inetd will continue to process incoming requests on that port. If you look in **/etc/inetd.conf** you see that almost exclusively TCP ports are no wait and UDP ports are wait.

Note that the inetd will start the program listed in **/etc/inetd.conf** based on the port requestion which is listed in **/etc/services**. However, if the appropriate program is already running, there is no need for inetd to start it. Therefore, you may have an entry in /etc/services, but not in /etc/inetd.conf. Services that are not started by **inetd** are usually referred to as "stand-alone" services. You may have case like HTTP or FTP where the program i.e. the service is already running, there is **no** entry in **/etc/services**. This is because such services us well-known ports and typically nothing else is going to try to use them. However, if you have a "non-standard" program using a special port, then it needs to write an entry in /etc/services to ensure that other programs do not inadvertently use that port.

9.1.4 Network Standards

Here we need to side step a little. We need to first talk about what goes into making a standard. Without standards, it makes communication between computers of different type very difficult. Just like you have bus standards like ISA and PCI so hardware can communicate with the CPU, you need some kind of standard.

In the Internet community, standards are both suggested and established through Request for Comments or RFCs. To some extent this is the "law". If one product claims to comply with a particular RFC, you know that any other application that does so should be able to communicate with it. However, RFCs include other things such as lists of previous RFCs and basic introductions to things like TCP.

Becoming a standard is a three step process. Usually, the first few paragraphs of an RFC will tell you to what stage it applies. Assuming of course, that the RFC is part of a standards proposal. At the first stage, the standard is proposed. Organizations then decide to implement the proposed standard. It requires three separate implementations before the proposal becomes a standard. (Finally, it becomes a standard. This is an oversimplication of the process, since there will also be a lot of discussion about the proposed standard.)

If you need information about a specific network standard, the first place to look is the most current RFC index, which is also published as an RFC. Not only does this list all the RFCs, but will also tell you if one RFC has been replaced and by which one.

Originally I had planned to include a list of the more commonly used and significant RFCs. I eventually realized that this was an unending task. When I started this, there were just over 1800 RFCs. The last time I checked before wrapping up this book, there are well over 2000. Instead I will simply tell you where to get them.

The first place is from the "central repository." These are obtainable using ftp from *ftp.ds.internic.net*. There is an rfc directory, which contains the RFCs in ASCII as well as many in postscript format. If you know what RFC you want, this can be obtained by sending an email message to mailserv@ds.internic.net. List each RFC you want in the format:

document-by-name rfc**XXXX**

where **XXXX** is the number of the RFC. You can obtain the index by including the entry:

document-by-name rfc-index

In addition, the RFCS are available from archives all over the Internet. However, rather than tying up the Internet bandwidth with a lot of copy of files you may not need. Check out *www.cdrom.com* or mail *info@cdrom.com*. This is for Walnut Creek CD-ROM and the sell a CD packed with thousands of documents related to the Internet, including the RFCs. Another site with pretty much the same kind of offering is InfoMagic. They can be reached at *www.infomagic.com* or *info@infomagic.com.*

For Linux systems running TCP/IP one of the most important standards deals with Ethernet. The encapsulation (packaging) of IP datagrams is defined for Ethernet in RFC 894. Developed in 1982 by Digital Equipment Corporation (DEC), Intel and Xerox Ethernet (spelled with a capital) is a standard, rather than a physical entity. Several years later, the 802 Committee of the Institute of Electrical and Electronic Engineers (IEEE or I-triple E), published standards of its own that differed in many ways from the original Ethernet standard. Collectively, these are referred to as the 802 IEEE standards. The 802.3 standard covers networks similar to Ethernet. The IEEE 802 encapsulation was defined in RFC 1042. Both of these use an access method called Carrier Sense Multiple Sense with Collision Detection or CSMA/CD.

Both of these framing types (RFC 894 and RFC 1042) use a 48-bit addressing scheme. These are generally referred to as the MAC or hardware address. The six bytes of both the destination and source machine are included in the header of both framing types. however, the remainder of the frame is different. As we talked about earlier, this layer is responsible for sending and receiving the IP datagrams. It is also responsible for sending and receiving other kinds of packets as well. These are packets from the Address Resolution Protocol (ARP) and the Reverse Address Resolution Protocol (RARP). We'll talk about both later on.

9.1.5 Subnet Masks

Subnet masks are 32-bit values that allow the recipient of IP packets to distinguish the network ID portion of the IP address from the host ID. Like an IP address, the value of a subnet mask is frequently represented in dotted decimal notation. Subnet masks are determined by assigning 1's to bits that belong to the network ID and 0's to the bits that belong to the host ID. Once the bits are in place, the 32-bit value is converted to dotted

decimal notation, as shown in the table below.

Address class	Bits for subnet mask	Subnet mask
Class A	1111111 00000000 00000000 00000000	255.0.0.0
Class B	1111111 11111111 00000000 00000000	255.255.0.0
Class C	11111111 11111111 11111111 00000000	255.255.255.0

Table - Default Subnet Masks for Standard IP Address Classes

The result allows TCP/IP to determine the host and network IDs of the local computer. For example, when the IP address is 102.54.94.97 and the subnet mask is 255.255.0.0, the network ID is 102.54 and the host ID is 94.97.

Keep in mind that all of this with the subnet masks is the principle and not necessarily the practice. If you (meaning your company) has been assigned a Class B address, then the the first two octets are assigned to you. You could then breakdown the class B net into Class C nets. If we take a look at Table 0\1, we see that there are 65,534 possible nodes in that network. That is really too many to manage.

However, if we considered each of the third octets to represent a sub-net of our class B network, they would all have 254 possible nodes per sub-net. This is basically what a class C net is anyway. We can then assign each sub-net to a department or building and then assign one person to manage each of the class C sub-nets, which is a little easier to do.

To keep the different class C subnet from interfering with each other, we give each sub-net a *Class C* subnet-mask, although the first octet is in the range for a Class B network. That way machines on this subnet are only concerned with packets for the subnet. We can also break down the sub-nets physically so that there is a gateway or router between the subnets. That way the physical network is not overburdened with traffic from 65,534 machines.

Let’s look at an example. Assume your company uses the Class B address 172.16.0.0. The different departments within the company are assigned a class C address that might look like this: 172.16.144.0. Although the first octet (172) says that this is a class B address, it is really the subnet-mask that makes that determination. In this case, our subnet mask would be: 255.255.255.0. Therefore, any packet that is destined for an address other than one starting 172.16.144.0 is not on this network.

It is the responsibility of IP to ensure that each packet ends up going to the right machine. This is accomplished, in part, by assigned a unique address to each machine. This address is referred to as the Internet address or IP address. Each network gets a set of these IP addresses that are within a specific range. In general, packets that are destined for an IP address within that range will stay within the local network. Only when a packet is destined for somewhere outside of the local network is it "allowed" to pass.

In other words, IP is responsible for the delivery of the packet. It functions similar to the post office, whereby you have both a sending and receiving address. Often times you have many more letters than a single mail bag can handle. The mail carrier (or someone else at the post office) will break down the number of letters into sets small enough to fit in a bag. This is what IP does.

Since there are many people using the line all at once, IP will break down the TCP packets into units of a specific size. Although often referred to also a *packets*, the more correct terminology is to refer to IP packets as *datagrams*. Just like bags of mail need to go from one post office to the next to reach their final destination, IP datagrams must often go through different machines to reach their final destination.

Saying that IP routing can be accomplished completely in software isn't entirely accurate. Although, no physical router is needed, IP can't send a packet to someplace where there is no physical connection. This is normally accomplished by an additional network card. With two (or more) network cards a single machine can be connected to multiple networks. The IP layer on that one machine can then be used to route IP packets between the two networks.

Once configured (how that's done, we'll talk about later), IP maintains a table of routing information, called (logically) a routing table. Every time the IP layer receives a packet, it checks the destination address

9.1.6 Routing and IP Gateways

I mentioned previously that IP is an unreliable, connection-less protocol. That is, it contains no provision to ensure that the packet arrives correctly at the destination, nor is there anything that guarantees that when packets do arrive they arrive in the correct order. Although IP is responsible to ensure that the packets get to the right machine, it has essentially no understanding of the physical connection between the two machines. IP will happily run on machines that are connected with something as simple as a telephone wire, to something as complex as satellites. IP depends on some other means to "physically" carry it across the network.

What this means is that the system administrator or network administrator is responsible for laying the "map" that is used to define which network address go with what sets of machine and what IP addresses are assigned to individual machines.

One important job that IP does is routing. That is, getting the packet to the right machine. If the source and destination machines are directly connected, that is on the same network, then routing is easy. Essentially there isn't any. IP sees this fact and simply hands the packets off to the data link layer. Otherwise, IP has to figure out how and where to send it.

Usually the 'how' is over a *router*. A router is some piece of hardware that acts like an air traffic controller send one packet off one way and others off a different way. Often routers are separate pieces of equipment that can be configured in very detailed ways. The disadvantage to this is that with power comes price. The ability to configure a router in many different ways usually means a high price tag. Fortunately, many operating systems, including Linux allow IP to serve as router-software. Thereby avoiding the cost of router hardware.

In comparison to the router is the concept of a *gateway*. Like a router, a gateway has knowledge of other networks and how to reach them. In general, we can think of a router as a special piece of hardware that does the work for us. In fact, there are companies that sell equipment called routers. A gateway is more of a concept, in that is the means by which you go from one network to another. Today, the distinction between a router and a gateway is blurred. Originally, a gateway was a machine that converted from one protocol to another. However, in common usage today, routers can server as gateways, gateways can serve as routers.

The path the packet takes from one machine to the next is called a *route*. Although each machine can maintain static routes for specific destinations, the default gateway is usually used to find remote destinations. The default gateway is needed only for computers that are part of an internetwork. If you have a gateway connected to several other networks, there will probably be route definitions for each of those other networks.

Let's look at this process as if we were sending a letter, as we did a little while ago. Each letter we send has an envelope which contains a message. On the envelope we write the source and destination addresses. When we mail the letter it gets to the post office and the person sorting the mail checks the destination zip code. If it's the same as the local zip code, the envelope is sent to one of the carriers for delivery. If the zip code is different, then it is sent to some other location. Perhaps all non-local mail is sent to the same place.

If you live across the country from me, the letter probably doesn't go directly from my local post office to yours. assuming I don't live in San Francisco and you don't live in New York The same applies to IP packets. My letter first goes to my local post office, if it is destined for a local address it is processed there. If not, it is sent along to a larger post office. If I sent a letter from Santa Cruz, California destined for Annsville, Pennsylvania, it will probably go first to San Francisco and then to New York or Philadelphia before it gets sent to Annsville.

Again, the same applies to IP packets. If I were communicating with a network on the other side of the country, my machine needs to know how to get to the other one. This is the concept of a "gateway". A gateway is the first step in the path, or "route" to the remote machine. Just as there are a couple of post offices between Santa Cruz and Annsville, there can be multiple gateways between computers.

Since San Francisco is the closest "major" city to Santa Cruz, it is possible that all mail bound for points beyond must first go through there. What if I lived in Fresno, which is about halfway between San Francisco and Los Angeles? If I sent a letter to Annsville, it could go through Los Angeles or it could go through San Francisco. To make things easy, it might always get sent through San Francisco it not destined for a local address.

What if the letter is bound for Los Angeles? It seems silly to go through San Francisco first when it is bound for LA. At the post office in Fresno, they might have a special procedure that says all remote mail goes through San Francisco, except for those with a zip code in a special range.

Here, too, the same applies to IP addresses. One machine may be defined as the "default" gateway, but if an IP packet was bound for a particular network it could be told to use a

completely different gateway. Which gateway to use to get to a particular machine or network is the concept of "routes." If I want all remotely-bound packets to use a particular route, I add that route as a default to my machine. If packets bound for a particular network are to go via a different route, I can add that route as well.

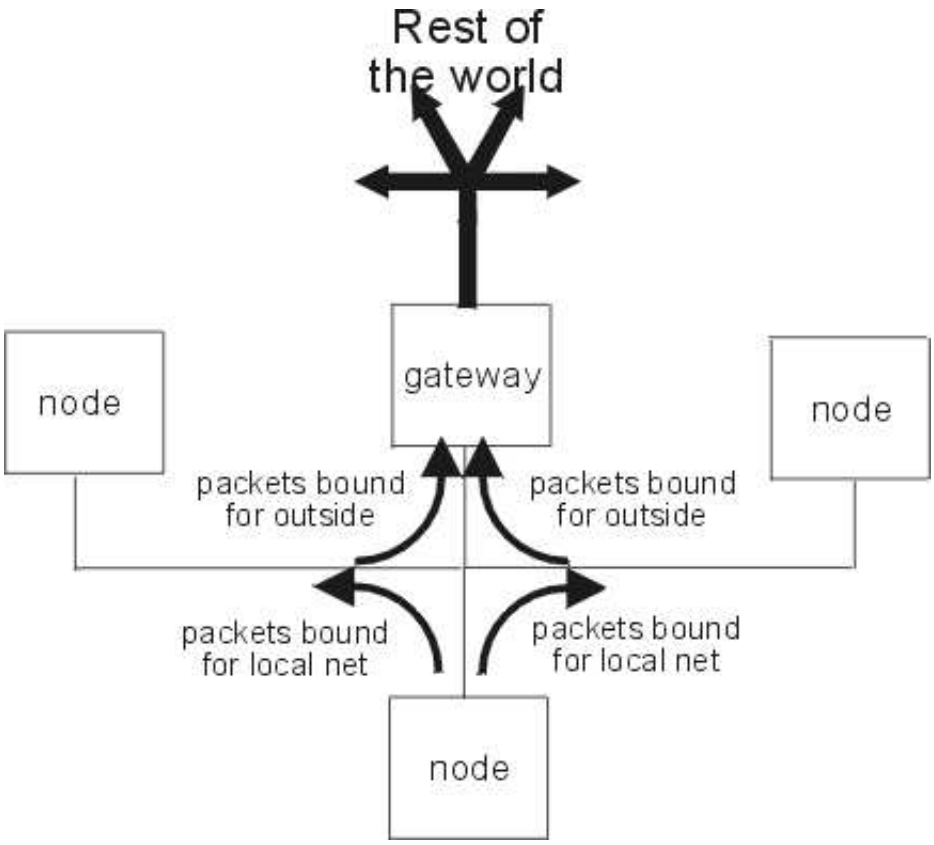


Image - Network gateway

When IP prepares to send a "message", it inserts the local source and destination IP addresses in the IP header. It then checks whether the network ID of the destination and source match the zip codes. If so, the packet is sent directly to the destination, since it is on the local network. If the network IDs don't match, the routing table is examined for static routes. If none are found, the packet is forwarded to the default gateway.

The default gateway is a computer connected to the local subnet and other networks that has knowledge of the IDs for other networks and how to reach them. Because the default gateway knows how to reach the other networks, it can forward the packet, either to other gateways or directly to that machine if the gateway is on the same network as the destination. This process is known as routing.

If you only have a single network, there is no reason to have a gateway, as each machine is directly connected to every other. It's possible that you only want certain machines within your network to go beyond the local net to the outside. In this case, these machine can have a default or static route default, while the others have none. However, users can add routes themselves, using the route command.

As we talked about earlier, TCP connections are not the only ones that are managed by inetd. Basically all network connections are. This can be understood if we go back to the telephone operator analogy. If the operator inetd is also the receptionist, we can then think of TCP connections as incoming telephone calls and UDP packets as incoming letters. Like incoming phones calls, the receptionist is responsible to route the incoming mail to the right person. This is a valid analogy, because like regular mail, there is nothing to guarantee the delivery of the message in the letter, although with TCP connections you can ask your partner to resend the message. Like TCP connections, UDP daemons are "listening" on specific ports. Also like TCP connections, these well-known ports are listed in /etc/services.

One common UDP connection is the routing daemon: routed. Routed supplies as you might have guessed routing information in the form of routing packets. If your system is serving as a router, then routed periodically sends copies of its routing tables to other machines.

One key difference is that routed is not actually started by inetd. Instead, it is normally started through one of the scripts under **/etc/rc.d** as the system is booting. This actually calls the script **/etc/sysconfig/network-scripts/ifcfg-routes**.

When it starts, routed makes the assumption that it will forward packets between all interfaces on the system. This only includes those that are "up" and does not include the loopback driver. The loopback driver is a special TCP/IP interface that simply loops the packets back to the local machine. Hence the name. Routed then transmits a REQUEST packet on each of these interfaces and waits for a RESPONSE packet for any other hosts. Potentially there are other machines on the network that are also sending REQUESTS packets, so routed can also respond to them.

The response routed gives is based on information it has in it's *routing tables*. This contains information about known routes, including how far away the destination machine is in turns of *hops* or intermediary machines. When routed receives a RESPONSE packet, it uses the information contained in that packet to update it's own routing tables. Look at the routed man-page for more information.

Routes are added to and removed from the system using the route command. The general syntax is:

route <option> command destination gateway metric

The two commands used are add and delete. The destination is the IP address of the machine or network you want to reach. You can also use tokens for the network name by including entries in the /etc/networks file. This is an ASCII file containing two columns. The first is the name of the network and the second column is the network address. You can then use that name in the route command.

The gateway is the IP address of the interface to which the packets need to be addressed. Keep in mind that the system must already know how to get to the gateway for this to work.

The metric is a value that normally indicates the number of intermediate machines hops. The system uses this value in determining the shortest route to a particular machine.

For example, let's assume we have an entry in /etc/networks like this:

```
siemau 132.147>
```

Let's also assume that the machine I need to use to access this route has an IP address of 199.142.147.1. I could then run the route command like this:

```
route add siemau199.142.147.1 0
```

This says that any packet destined for the *siemau* network as defined in /etc/networks should go to the IP address 199.142.174.1 with a metric of 0. Normally, 0 is used when the IP address you specify is directly connected to your machine.

If you have a single machine that serves as your gateway to the rest of the world, you can specify default instead of a specific address or network as your destination. In the example above, if we wanted to use the same machine for all networks instead of just siemau, the command would look like this:

```
route add default 199.142.147.1 0
```

As you move about the network, dynamic entries are created by the routing protocol that you use. Most commonly routed The routing protocol communicates with it's counterpart on other machines and adds entries to the routing tables automatically.

When it starts, routed looks for the file **/etc/gateways**, which contains a list of gateways. What else? The general format for this file is:

```
<net|host> name gateway metric type
```

The first field specifies whether the gateway is to a specific machine or network. The name field is the name of the destination host or network. This can either be an IP address or a token. If using a token, then the hostname must be located in **/etc/hosts** or can be determined through DNS. If through DNS, routed must be started after named. If a network, the name must be in /etc/networks.

The gateway field is the name or address of the gateway that is to be used. The metric is the same as for routes and indicates the number of hops. The type can be either passive, active or external. A passive gateway is one that is not expected to exchange routing information. Active gateways will exchange information and usually have routed running on them. External gateways are ones that are managed by another system, but alternate routes should not be installed.

9.1.7 The Domain Name System

If you have TCP/IP installed, by default, your machine is set up to use the /etc/hosts file. This is a list of IP addresses and the matching name of the machines. When you try to connect to another machine, you can do it either with the IP address or the name. If you use the name, the system will look in the /etc/hosts file and make the translation from name to IP address. The only real drawback with this scheme is that every time a machine is added or removed from the network, you have to change the /etc/hosts file on all the affected machines.

Those you of that have had to administer large networks know that updating every /etc/hosts file like this can be a real pain. There is always at least one that you forget or you mis-type the name or address and have to go back and change it on every machine. Fortunately, there is hope.

Provided with Linux is a hostname/IP address database called the Berkeley Internet Name Domain BIND service. Instead of updated every machine in the network, there is a Domain Name System DNS server that maintains the database and provides the client machines with information about both addresses and names. If machines are added or removed, there is only one machine that needs to get changed. This is the Name Server. Note: Some documentation translates DNS as Domain Name Server. Other references most importantly the RCFs call it the Domain Name System. I have seen some references call it Domain Name Service. Since we know what it is, I'll just call it DNS.

So, when do you use DNS over the /etc/hosts file? Well, it's up to you. The first question I would ask is "Are you connecting to the Internet?" If the answer is "yes", "maybe" or "someday" then definitely set up DNS. DNS functions somewhat like directory assistance from the phone company. If your local directory assistance doesn't have the number, you can contact one in the area you are looking. If your name server doesn't have the answer, it will *query* other name servers for that information. assuming you told it to do so.

If you are never going to go into the Internet, then the answer is up to you. If you only have two machines in your network, the trouble setting up DNS is not worth it. On the other hand, if you have a dozen or more machines, then setting it up makes life easier in the long run.

There are several key concepts that need to be discussed before we dive into DNS. The first is DNS, like so many other aspects of TCP/IP, is client-server oriented. We have the name server containing the IP addresses and names which serves information to the clients. Next, we need to think about DNS operating in an environment similar to a directory tree. All machines that fall under DNS can be thought of as files in this directory tree structure. These machines are often referred to as nodes. Like directories and file names, there is a hierarchy of names with the tree. This is often referred to as the domain name space.

A branch of the DNS tree is referred to as a domain. A domain is simply a collection of computers that are managed by a single organization. This organization can be a company, university or even a government agency. The organization has a name that it is know by to the outside world. In conjunction with the domains of the individual organizations, there are things called *top-level domains*. These are broken down by the function of the domains under it. The original top level domains are:

COM - Commercial
EDU - Educational
GOV - Government
NET - Network
MIL - Military
ORG - Non-profit organizations

Each domain will fall within one of these top-level domains. For example, there is the domain *prenhall*for Prentice Hall, which falls under the commercial top-level domain. It is thus designated as *prenhall.com*. The domain assigned to the White House is *whitehouse.gov*. The domain assigned to the University of California at Santa Cruz is *ucsc.edu*. Note that the dot is used to separate the individual components in the machine's domain and name

Keep in mind that these domains are used primarily within the US. While a foreign subsidiary *might* belong to one of these top-level domains, for the most part, the top level domain within most non-US countries is the country code. For example the geographical domain Germany is indicated by the domain abbreviations de for Deutschland. These are examples, however. I do know some German companies within the com domain. There are also geographic domains within the US, such as ca.us for California as compared to just .ca for for Canada. This is often for very small domains or non-organizations, such as individuals.

In many places, they will use a combination of the upper-level domains that are used in the US and their own country code. For example, the domain name of an Internet provider in Singapore is *singnet.com.sg*. Where *sg* is the country code for Singapore.

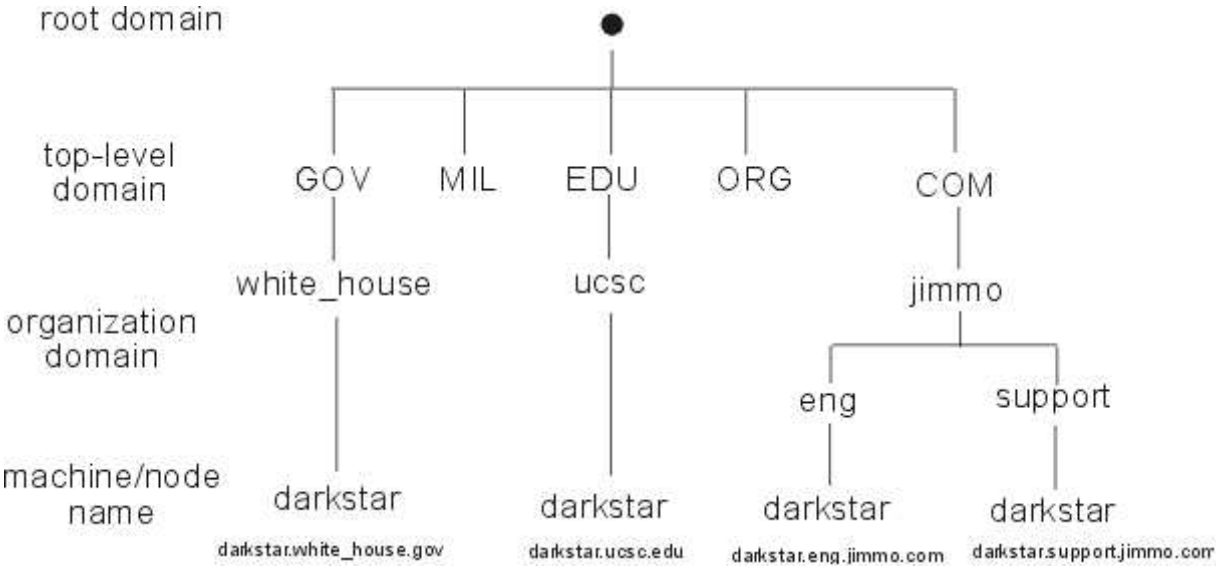


Image - Internet domains

Within each domain, there *may* be sub-domains. However, there doesn't have to be. You usually find sub-domains in larger domains in an effort to break down the administration into smaller units. For example, if you had a set of machines that was for use by Or, if your company had a sub-domain for sales it might be sales.yourdomain.com.

Keep in mind that these are just the domain names, not the machine, or node name. Within a domain there can be in principle any number of machines. A machine sitting on the desk in the oval office might be called *boss1*. It's full name, including domain would be *boss1.pres.whitehouse.gov*. A machine in your sales department called darkstar would then be *darkstar.sales.yourdomain.com*.

Up to now, I have only seen a machine name with five components: the machine name, two sub-domains, the company domain and then the top-level domain. On the other hand, if there was no *sales* sub-domain, and everything was under the *yourdomain.com* domain, the machine's name would be: *darkstar.yourdomain.com*.

You may often see the fully-qualified domain name FQDN of a machine listed like this:

darkstar.yourdomain.com.

Including the trailing dot.. That dot indicates the root domain. This has no name other that root domain or .read "dot". Very similar to the way the root directory has no name other than root or /. In some cases this dot is optional. However, there are cases where is it required and we'll get to those in the section on configuring DNS.

Like files, it is possible that two machines have the same name. The only criteria for files is that their full path be unique. The same applies to machines. For example, there might be a machine darkstar at the whitehouse. Maybe George is a closet Dead Head It's FQDN would be *darkstar.whitehouse.gov*. This is obviously not the same machine as darkstar.yourdomain.com any more than 1033 Main Street in Santa Cruz is the same as 1033 Main Street in Annsville. Even something like *darkstar.support.yourdomain.com* is different from *darkstar.sales.yourdomain.com.*

A zone is a grouping of machines that may, or may not, be the same as a domain. This is the set of machines over which a particular name server has authority and maintains the data. In our example above, there might be a zone for support, *even if* there was no sub-domain. On the other hand, there might be a team.support.yourdomain.com domain, but the zone is still yourdomain.com. Therefore, zones can be sub-ordinate or superior to domains. Basically, zones are used to make the job of managing the name server easier. Therefore, what constitutes a zone depends on your specific circumstances.

In DNS, there are a couple different types of servers. A primary server is the master server for one or more DNS zones. Each server maintains the database files, and is considered the authority for this zone. It may also periodically transfer data to a secondary server, if one exists for that zone.

DNS functions are carried out by the Internet domain name server, named. When it starts, named reads it's configuration file to determine what zones it is responsible for and in which files the data is stored. By default, the configuration file /etc/named.conf. However, named can be started with the -b option to specify an alternate configuration file. Normally, named is started from a script in /etc/rc.d.

For example, the primary server for the *yourdomain.com* domain needs to know about the machines within the *support.yourdomain.com* domain. It could server as a secondary server to the *support.yourdomain.com* domain, whereby it would maintain all the records for the machines within that sub-domain. If, on the other hand, it servers as a stub server, the primary for the *yourdomain.com* need only know how to get to the primary for the *support.yourdomain.com* sub-domain. Note here, that it *is* possible for a server to be primary in one zone and secondary in another.

By moving responsibility to the sub-zone, the administrator of the parent zone, does not need to concern him or herself with changing the configurations files when a machine is added or removed within the sub-zone. As long as the address of sub-zone primary server remains matches the stub server entry all is well.

A secondary server takes over for the primary, should the primary go down or be otherwise inaccessible. A secondary server maintains copies of the database files, and "refreshes" them at predetermined intervals. If it cannot reach the primary to refresh it's files, it will keep trying at again predetermined intervals. If after another predetermined time, the secondary still cannot reach the primary, the secondary considers it's data invalid and flushes it.

Caching-only servers saves data in a cache file only until that data expires. The expiration time is based on a field within the data that is received from another server. This is called the time-to-live. Time-to-live is a regularly occurring concept within DNS.

A slave server can be a primary, secondary, or caching-only server. If it cannot satisfy the query locally, it will pass, or forward, the request to a fixed list of forwarders forwarding server, rather than interacting directly with the primary name servers of other zones. These request are recursive, which means that the forwarder must answer either with the requested information or saying it doesn't know. The requesting machine then asks the next server, then the next and then the next until it finally runs out of servers to check or gets an answer. Slave servers never attempt to contact servers other than the forwarders.

The concept of recursive request is in contrast to iterative requests. Here the queried server either gives an answer or tells the requesting machine where it should look next. For example, darkstar asks, iguana, the primary server for support.yourdomain.com for some information. In a recursive query, iguana asks, boomer, the primary server for yourdomain.com and passes the information back to darkstar. In a iterative query, iguana tells darkstar about boomer, and darkstar then goes asks boomer. This process of asking name servers for information, whether recursive or iterative is called *resolution*.

Keep in mind that there is client software running on the server. When an application needs information, the client DNS server asks the server for the information, despite the fact that the server is running on the same machine. Applications don't access the server directly.

There is also the concept of a root server. These are severs located at the top of the domain tree and maintain information about the top-level zone. Root servers are positioned at the top, or root, of the DNS hierarchy, and maintain data about each of the top-level zones.

9.1.7.1 Configuring the Domain Name System DNS

In the first part of this section, I discussed DNS as being a means of centrally administering the file necessary for node name to IP-address translation. Although the relationship of the files is pretty straight-forward, they are rather intimidating to the uninitiated. Myself included

So, what do the DNS configuration files look like? Well, since the first file that named looks at is /etc/named.conf, that seems like a good place to start. Let's assume we wanted to set up a primary name server. We might have a file that looks like this:

```
; Boot file for Primary Master Name Server
;
;
; type  domain                    source file or host
;

directory    /etc/named.d
primary      siemau.com                   siemau.forward
primary      147.142.199.in-addr.arpa     siemau.rev
primary      0.0.127.in-addr.arpa         named.local
cache        .                            root.cache
```

Lines beginning with a semi-colon are considered comments and blank lines are ignored. The first line with configuration information is:

```
directory   /etc/named.d
```

This tells *named* that if no path is specified, it should look for the other configuration files in the specified directory. In this case, /etc/named.d. Since the named.boot file is read when named starts up, you could change it to anything you want.

The first primary line says that we are the primary name server for the domain siemau.com This says that the information to resolve forward requests are found in the file siemau.forward. Okay, so what are forward requests. Well, "forward requests" is my term. I use it for two reasons. First, the file containing the information, often ends with .forward. Second, I think the primary function of a name server is to translate names to IP addresses. Therefore, this is going forward. Translating IP addresses to names is going in reverse.

Note that you will often see that the forward mapping file is referred to as named.hosts or domain_name.host and the reverse mapping as named.rev or domain_name.rev. I like to call one .forward and one .rev so I know by looking at them what their function is. It doesn't matter what you call them as long as there are pointed to in named.boot.

In order to be the primary server we must say that we are the primary. This is accomplished through the Start of Authority SOA record, which says we are the start of authority for the given domain. That is, when trying to find the answer to a query, the buck stops here. We have all the right answers when it comes to this domain. The SOA record is required and might look like this:

```
siemau.com.   IN   SOA siemau.siemau.com. jimmo.siemau.com.
                       8675309 ; Serial
                       10800     ; Refresh
                       800         ; Retry
                       3600000  ; Expire
                       259200   ; Minimum
```

The fields in the first line are: domain, data class, type of record, primary name server, responsible individual. The data class will always be IN for Internet. Often you will see root or postmaster as the person responsible for this domain. Here, I picked myself. Note that the format is jimmo.siemau.com and not jimmo@siemau.com as one might expect.

The Serial number is a way for secondary servers to keep track of the validity of their information. If the secondary has a serial number that is lower than the serial number on the primary, it knows that the information is outdated. It will then pull over an updated copy.

The Refresh is how often in seconds the secondary servers should check the primary for updated information. In every implementation I have ever seen, this value is set to 10800 seconds, or three hours. You can change it if your site requires it.

The Retry is how often in seconds the secondary server should retry to contact the primary. This value of 3600 seconds one hour is also something I have seen in almost every case. Again, change it as you need it.

The Expire time is how long the secondary will try before it gives up and declares the data it has as invalid. This is based on the attitude that no data is better than old data. Here we have 1000 hours or almost 42 days.

The Minimum is the value that other resource records should use as their time-to-live, if no other value is defined for them. The time-to-live is how long a given piece of information is considered valid.

At the end of each of these records you see a semi-colon. This is used in DNS database files as the start of a comment. Any text from the semi-colon to the end of the line is considered part of the comment. You will also see that many lines have semi-colons as their first character. In these cases, the whole line is a comment.

Note also there is a dot after each .com entry. This indicates the end of the name. Remember I mentioned that the trailing dot indicates the root domain? In these cases here, this dot is required. If you leave it off, the system will assume that it should tack on the domain name onto the end. Therefore, you might end up with the domain name twice. This behavior can actually come in handy and we'll get to it shortly.

The SOA record is just one resource record that you find in DNS database files. There are several others that we will get through during the course of this discussion. Resource records have general format:

```
name {ttl} data-class record-type record-specific-data >
```

The name is simply something we are looking for. For example, we might have a machine name and we are looking for the IP address. We have the machine name, this is our value. On the far right is the record-specific-data or the IP address. The TTL value is the time-to-live. This is an optional value since we already defined a minimum in the SOA record. We could have also defined a ttl value for this SOA record, if we had wanted. The data-class can be one of several values. However, only the IN for Internet class is commonly used, therefore that is the only one we'll use here. The record-type tells us what kind of resource record we have. For example, SOA is one record type.

After the SOA record there is usually an entry saying which machines are name servers, such as:

```
siemau.com. IN   NS    siemau.siemau.com.>
```

The value we have is siemau.com. For this record type, this value is the domain name. The domain is the same for the SOA record, as we are defining this machine to be the name server as well. The data-class, again, is IN for Internet. Since we are defining which machine is the name server, the record type is NS, for name server. Lastly, we get the FQDN of the machine *siemau.siemau.com.*.Note that in both cases we had the dot at the end of each name.

One thing that I should point out here is that a good choice for which machine is the name server is one that is on multiple networks, that is one that serves as gateway. This is a good choice since it already has to know about multiple networks to be the gateway. It is said to be *well connected*. This saves managing one machine as the gateway and the other as the name server.

Next, we have the name to address mappings. Let's assume for simplicity's sake that I only have two other machines in my network. The entries for all my machines might look like this:

```
siemau.siemau.de.   IN A 192.168.42.1
vesta.siemau.de.    IN A 192.168.42.2
jmohr.siemau.de.    IN A 192.168.42.3
```

The general format is:

machine-name data-type record-type IP-address

Note that despite the fact that siemau is our name server, we still need to include it here. Otherwise there would be no way to translate it's name to an address. The new piece of information here is the A record-type. This simply says that this specific record is making the translation from machine name to IP-address. Each entry is referred to as an address record, or address resource record. Note again the trailing dot ..

We also need a mapping for the node "localhost". This is a special name for the local machine and is accessed using a special driver called the "loopback driver". Rather than accessing the card, the loopback driver knows that this is the local machine and does not need to go out to the network card. Certain function on the system take advantage of the capabilities of this driver.

```
localhost            IN   A     127.0.0.1>
```

One thing I need to point out is the dot . at the end of each FQDN. This says that the name stops here. Remember that the dot is use to indicate the root domain. By putting the dot here, this says that we have reached the root domain, so we won't go any further.

Leaving the dot off can be a mistake or intention. In these examples it would be a mistake. In fact, in the time I was doing tech support, leaving off the dot was perhaps the most common mistake made when configuring the name server. However, we can leave it off intentionally in certain circumstance and have it be correct. We can use abbreviations shorten forms to indicate the machine name. For example, we could have written the first entry like this:

```
siemau  IN A 192.168.42.1 >
```

Because we already defined what the domain name is in the named.boot file, the system knows what to append. Therefore, we can try to contact either siemau or siemau.siemau.de and the name server will translate that correctly to 192.168.42.1.

We now need to make the translations from IP address to name. As I mentioned before, there are "reverse" translations. The data for these translations is in the file siemau.rev as indicated by the line from named.boot:

```
primary     42.168.192.in-addr.arpa  siemau.rev >
```

In general, the format of the entry is similar to that of the forward entries. For our three examples they would look like this:

```
1.42.168.192.in-addr.arpa.   IN   PTR  siemau.siemau.de.
2.42.168.192.in-addr.arpa.   IN   PTR  vesta.siemau.de.
3.42.168.192.in-addr.arpa.   IN   PTR  jmohr.siemau.de.
```

There are a couple of new things here. First, is the record type. Here we have PTR for pointer records. These point to the machine name. The next is the "in-addr.arpa" after the IP address. To understand we need to take a step back.

Assume we wanted to make the translation from machine name to IP address and we had no idea where that machine was. As I mentioned there are name servers for all of the top-level domains that are aware of the name servers for all of the domains under it. For the .com domain, one such machine is kava.nisc.sri.com. So, if we had wanted to find the machine vesta.siemau.de, we could ask the name server that was responsible for the .com domain *kava.nisc.sri.com*. Since kava knows about the siemau domain, and knows that siemau.siemau.de is the name server for that domain it tells you to go ask siemau yourself.

Now, let’s go the other way. Question: is the domain with the first octet of the IP address 199 a .com, .edu or .gov domain? How can you tell? The answer is that there is no way to tell. IP addresses are not arrange by the type of organization. We can guess that the network 199 is probably a class C network since it is over 192, but it can just as easily be .com a .edu or anything else. So rather than trying to find the name server for every single domain and asking "Are you the right one?", a quicker way had to be developed.

The solution was to create a portion of the Internet name space that used the addresses as a name. This portion is considered a separate domain and is referred to as the in-addr.arpa domain. The names of both machines and sub-domains within the in-addr.arpa domain are simply the IP addresses. There are 256 sub-domains of the in-addr.arpa domain, 256 sub-domains of each of those domains and so on.

If you look, the "names" listed in the in-addr.arpa domain have the IP address reversed from the way we are accustomed to seeing it. This is keeping with the idea that in the names, the more specific names are on the left and get more general as you move to the right. It also makes things easier to managed since we can say that the 42.168.192.in-addr.arpa domain is administered by one organization. This is because 192.168.42 is a separate Class C network.

Note also that there is a dot at the end of the reverse address. Here, too, this tells the name server where the end is. Since we already said what the in-addr.arpa domain was in the named.boot file. We can make a short cut by listing only the host portion, just like we did with the FQDN. The entry for siemau would then look like this:

```
1 IN PTR siemau.siemau.de.
```

Note that here we have the dot a the end of the FQDN, but it wasn't at the end of the IP address in the address A record. This is because the dot comes at the end of a domain name. In in-addr.arpa notation, the IP address is part of a name, it just happens to look like an IP address, albeit a reversed one. Think of it this way, a period comes at the end of a sentence, which is a bunch of words. If you have a set of numbers, there is no period.

If we had a class B network, we could also make use of these abbreviations. For example, if siemau had an address of 159.144.147.1, it's pointer PTR record could have been written like this:

```
1.147 IN PTR  siemau.siemau.de.
```

This reminds me of the second most common error I say in support and that is using the abbreviations for the reverse address, but not reversing them! That is, in this example above, writing it as:

```
147.1 IN PTR  siemau.siemau.de.
```

Don't do that! A reverse domain has the IP address portion of the name reversed as well. No matter what part you include.

By writing the IP-Address reversed like this, we are essentially creating a new domain. The root domain, is still dot .. However, this time there is just the single top-level domain in-addr.arpa. This notation is often referred to as the *reverse domain*. Because we are defining a new domain in the siemau.rev file, we need a new Start of Authority record. We could copy the SOA record from the siemau.forward file, however the domain is wrong. The domain is now 147.144.199.in-addr.arpa. So, all we need to do is replace the old domain name with the new one and the entry would be identical. The first line would then look like this:

```
147.144.199.in-addr.arpa.    IN   SOA   siemau.siemau.de jimmo.siemau.de
```

We can now duplicate the remainder of the SOA record from the siemau.rev file. One thing I do to help keep things straight is to think of the NS record as part of the SOA record. In "reality", they separate records. However, if you think of them together, you won't forget and leave off the NS record. Since we are defining a new domain, we also need the NS record for this new domain. It's NS record would look like this:

```
147.144.199.in-addr.arpa. IN  NS siemau.siemau.de.
```

However, I don't like the idea of two SOA records. There is the chance that I update the database files, but forget to update one of the SOA record with the new serial number. To eliminate that problem, there is a directive that you can give the name server to include another file while it's reading the information. This is the $INCLUDE directive. To include a

single SOA record, we create a file, perhaps siemau.soa and use the $INCLUDE directive in both the siemau.forward and siemau.rev files. The line would look like this:

```
$INCLUDE siemau.soa
```

Since we already defined the directory in the named.boot file, there is no need for a path here. However, we have a problem. The SOA record in siemau.forward is for a different domain siemau.dom than in siemau.rev 147.144.199.in-addr.arpa. We can take advantage of a magic character: @. This will be read as the domain name, provided the domain name is same as the origin The origin is the machine that this data is on.

Let's create a single SOA file i.e. siemau.soa and make it identical to other others with the exception of the domain name. Instead we replace it with the "@." Next we remove the SOA records from the siemau.forward and siemau.rev file and replace it with the $INCLUDE directive above. When the name server reads the siemau.forward file, it gets to the $INCLUDE directive sees that it needs to include the siemau.soa file. When it gets to the "@", the system translates it as siemau.de. Next, when the system reads the siemau.rev file, it sees the same directive, includes the same file, however, this time the "@" is interpreted as "147.144.199.in-addr.arpa.

There are still two lines in the named.boot file that we haven't covered. The first sets up this servers as primary for the "local" domain. This is a special domain that refers to this host only. Remember from our discussion of IP address that the IP address for the local host is 127.0.0.1. The "network" that this host is on is 127.0.0. We always need to be the primary name server for this domain, there we have the line in our named.boot:

```
primary     0.0.127.in-addr.arpa      named.local
```

The named.local file could contain just two lines:

```
$INCLUDE named.soa  1     IN PTR  localhost
```

Note that here, too, we are including the named.soa file. When the system reads named.local, it includes named.soa and the "@" is translated to 0.0.127.in-addr.arpa as it should.

The last line tells us to read the cache file:

```
cache       .   root.cache
```

The root.cache file is the list of the root domain name serves. You can obtained the most current list root name servers using anonymous ftp from the *machine ftp.rs.internic.net*. The file is domain/named.root.

Let's assume we want vesta to be the secondary name server for this domain. We would then create a named.boot file on vesta, that might look like this:

```
directory      /etc/named.d
secondary      siemau.de                  192.168.42.1   siemau.forward
secondary      42.168.192.in-addr.arpa    192.168.42.1   siemau.rev
primary        0.0.127.in-addr.arpa       named.local
cache          .                          root.cache
```

If we look carefully, we see that the only difference is the that for the forward and reverse files, we change "primary" to secondary. Note that that vesta is still the primary for the domain 0.0.127.in-addr.arpa the local domain. The contents of the files are theoretically the same. This is where the concept of the serial number comes in. When the secondary loads it's file it compares the serial number to what it reads from the primary. Note also that the IP address, 192.168.42.1, the address of the *primary* server. In our case this is the machine siemau.

If we want a caching only server, the named.boot file is a little simpler:

```
directory /etc/named.d

primary  0.0.127.in-addr.arpa   named.local
cache    .                      root.cache
```

We still specify the directory and the root.cache file. However, we are now the primary for just a single machine, ourselves.

In any of the example named.boot files we could have include a line that simply said:

```
slave
```

That would be a name server, regardless of what of type, that forwards all requests that it cannot satisfy to a list of predetermined forwarders. If this sever does not have the answer, it will not interact with any other server, except for those listed as forwarders. Therefore, any time you have a slave server, you must also have a list of forwarders. The entry for the forwarders might look like this:

```
forwarders   199.145.146.1 199.145.148.1
```

The last kind of server I called a client. It's configuration is the simplest. You need to create a file called /etc/resolv.conf and include a line defining the domain and then a list of the name servers or *resolvers* as they resolve your queries. If we had siemau as the primary for this domain and vest the secondary, our file might look like this:

```
domain siemau.de
nameserver  199.144.147.1
nameserver  199.144.147.2
```

Note that if this file doesn't exits, your system will expect to get it's information from the /etc/hosts file. Therefore, you can say that on the client side that if /etc/resolv.conf doesn't exist, you are not using DNS.

If you have a larger network with many different departments, you might have already decided to have multiple name servers. As I mentioned, it is a good idea to have your name servers also be the gateway as they are "well connected." This also applies to the gateway you have to the Internet. To make life simpler for both you trying to reach the outside world and the people trying to get in, it is a good idea to have the Internet gateway also the primary name server for your entire domain.

If your organization is large, then having the Internet gateway a name server for your entire organization would be difficult to manage. Since you already decided to break your network down by department, then each department should have its own name server. One thing you could do is set up the domain name server as a secondary for the sub-domains. This is easy to set up as we described above and saves you from having to maintain a list of every machine within your organization.

There are still several record types that I haven't mentioned. One of them is machine aliasing. For example, you might have a machine that acts as your ftp server, your mail server and your World Wide Web server. Rather that requiring everyone accessing this machine to know that vesta.siemau.de is the server for all three of these functions, you can create an alias to make things easier. This is done by the CNAME canonical name record. Example entries would look like this:

```
ftp             IN CNAME vesta
mailserv        IN CNAME vesta
www             IN CNAME vesta
```

Any reference to these three machines is translated to mean the machine vesta. Keep in mind that if you use such an alias, this should be the only reference in you name server database. You should not have PTR record that point from an IP address to one of these aliases, instead use it's canonical real name, vesta.

We can also use the name server database to store information about the machine itself. This is done through the HINFO host information resource record. We could have the entry for our machine, siemau, that looks like this:

```
siemau IN A      192.168.42.1
       IN HINFO  Pentium  Linux
```

The record specific data on the right is composed of two strings. The first is the Hardware and the second is the Operating System. The strings may contain spaces or tabs, but you need to include them within quotes or the system will seem them as separate strings. "Technically" these two strings should be "machine name" and "system name" and match one of the strings in RFC 1340, but this requirement is not enforced. There is also the problem that man newer machines won't be on the list.

One thing that seems to be missing is the machine name from the HINFO record. Well, this is another short cut. By leaving the name field out of any record, it defaults to the same value as the previous entry. Here, the previous entry is the A record for siemau. Therefore, the name field of the HINFO record is also siemau.

We can also use the name server to manage certain aspects of our users. For example, you can have mail systems such as MMDF read the name server information to determine what machine a particular user gets his or her mail on. This is done with the MB mailbox resource record. An example, might look like this:

```
jimmo  IN  MB  siemau.siemau.de.
```

In this domain, mail for the user jimmo should be sent to the machine siemau.siemau.de. Note that this only works if you have unique users within the domain. In addition, there must only be one MB record for each user.

You can make things easier by specifying a single machine as the mail server. This is done with a MX mail exchanger resource record. The MX record can also be expanded to include sub-domains. For example, the name server for the siemau.de domain has MX records for all the sub-domains under it. The resource specific information contains the presence, which is a numerical value used to determined the order in which mail is sent to different machines. The preference should be 0 unless you have multiple mail servers within the domain.

Lets assume that this is a large company and we have given each department its own domain regardless if they have different IP sub-nets. We then decide that mail sent to any one in a subdomain goes to the mail-server for that sub-domain, but any mail to the parent domain, goes to a different server. Some entries might look like this:

```
siemau.de.                  IN      MX      0       siemau.siemau.de.
finance.siemau.de.          IN      MX      0       cash.finance.siemau.de.
sales.siemau.de.            IN      MX      0       buyer.sales.siemau.de.
                            IN      MX      1       cash.finance.siemau.de.
market.siemau.de.           IN      MX      0       editdmon.market.siemau.de.
images.market.siemau.de.    IN      MX      0       images.market.siemau.de.
```

In this example, mail sent just to a user in the domain siemau.de will go to siemau.siemau.de. Mail sent to either of the other three domains*finance, sales,* and *market* will be send to a machine in that respective domain. Note that there are two MX records listed for the *sales.siemau.de* domain. One has a preference of 0 and the other a preference of 1. Since the preference for *buyer.sales.siemau.de* 0 is lower than for *cash.finance.siemau.de* 1, the mail program will first try buyer. If buyer can't be reached it will try cash. Keep in mind that the numbers have only mean what order to check. We could have given one a preference of 45 and the other a preference of 66 and they would still have been checked in the same order.

Let's assume that we feel mail to the sales department is so important that we want it to try still another machine before it gives up. We could have a third MX record for sales.siemau.de that looks like this:

```
IN  MX  2 siemau.siemau.de.
```

In this case, buyer will be checked and if the mail message cannot be delivered, cash is checked. If cash cannot be reached, *siemau* is checked. If we changed the preference of siemau to 1, like the preference for *cash*, one of them will be chosen at random. This can be used if you want to spread the load across multiple machines.

There are a few other resource records types that we haven't discussed. There are not a commonly used as the others, so we will have to forgo talking about them. If you would like to learn more, check the book *DNS and BIND* by Paul Albitz and Cricket Liu from O'Reilly and Associates.

As I mentioned earlier, you can use the $INCLUDE directive to include a file containing the SOA record. However, you can use the $INCLUDE directive to include any file. This is very useful if you files have grown to unmanageable sizes and you need to break them apart. Assume your network contains 200 machines. There are A, PTR and possibly MX records for each machine. You could created three separate files for each of these. Normally, A and PTR are in separate files already. You could then use the $INCLUDE directive to include the MX records in one of the other files.

The Name Server

Sorry, you're going to have to do it. Unless you are a flawless typist and have every step written down exactly, one day you are going to forget something. As a result, the name server won't function the way you expect. Hopefully, it's something simple like forgetting a dot and you can quickly make the change.

The problem is what to do after you've made the change. Remember, named reads the configuration information when it starts. To get named to re-read the configuration file, you could stop and restart TCP. However, this would not be taken too kindly by the users who have their connection suddenly drop. The solution is to poke named in the ribs and tell it go re-read the files. This is done by sending the named process a hang-up signal with "kill -1 <pid>", where <pid> is the PID of named. To find the PID, either grep through ps -e or look in /etc/named.pid. This also has the side effect of having secondary name servers check the serial numbers, which can be used to force updates.

If you want to have named dump its current database and cache you can send named a interrupt signal SIGINT, kill -2. This dumps the database into **`/usr/tmp/named_dump.db`**. Sending named SIGUSR1 kill -16 you can turn on debugging, the output which is sent to /usr/tmp/named.run. Subsequent SIGUSR1 signals sent to named will increase the debugging a level. Sending it SIGUSR2 kill -17 turns off debugging completely.

You can also get named to trace all incoming queries, which is sent to /usr/adm/syslog. This is done by sending SIGWINCH kill -20. Be careful with this, however. Even on smaller networks, the amount of information logged in syslog can be fairly substantial. If you forget to turn it of, you can fill up your root file system. To turn of tracing, send SIGWINCH again. Note that all of theses options can be enabled from the start-up script in **`/etc/rc2.d`**.

Perhaps the most useful debugging tools is nslookup name server lookup. The nslookup command can be used either ly or non-ly, to obtain information from different servers. Depending on how it's set, you can input an IP address and get a name back or input the name and get the IP address back. If you are using the name server, either as a server or a client, nslookup can be used to gather information.

To start it ly, simply type nslookup at the command line. You are then brought into the nslookup "shell," where you can issue commands and set the options needed to get the information you need. By using 'set all' you can view the current options. By default, nslookup will return the IP address of the input machine name A forward query. For example, if we ran nslookup on vesta, nslookup would respond with something like this:

```
Default Server:  siemau.siemau.de
Address:  192.168.42.1

>
>
```

This tells us what the default server is and shows that it is ready for the first command by displaying the > prompt.

Let's say we wanted to see what information the name server when we run nslookup on siemau. We type in jmohr and press return. This gives us:

```
> siemau
Server:  localhost
Address:  127.0.0.1

Name:    siemau.siemau.de

Address:
192.168.42.1
>
```

As we can see this is what we expect. Note that in the first case, the default server was siemau. However, when we run it on the name server itself, the server is "localhost."

One question that comes to my mind is does it translate the IP address back to the host name correctly? Let's see. When we type in the IP address, we get:

```
> 192.168.42.3
Server:  localhost
Address:  127.0.0.1

Name:    vesta.siemau.de
Address:  192.168.42.3
>
```

We can list all the available information for a domain with the 'ls' command. The general syntax is:

ls [option] domain

Where domain is the name of the domain we would like the information about. If we want we can redirect the output to a file, we can use either type of output redirection > or >>. If we want to see it on the screen, we get:

```
> set all
Default Server:  localhost
Address:  127.0.0.1

Set options:
  nodebug         defname         search          recurse
  nod2            novc            noignoretc      port=53
```

```
  querytype=A     class=IN        timeout=5       retry=2
  root=f.root-servers.net.
  domain=siemau.de
  srchlist=siemau.de
>
```

If I want to see everything there is about a domain. I use the ls command. Keep in mind that by itself, the system will think it is a machine name and try to resolve it. However, followed by the domain name we get:

```
> ls
siemau.de

[localhost]

 siemau.de.  server = siemau.siemau.de
 siemau      192.168.42.1
 jmohr       192.168.42.2
 localhost   127.0.0.1
 vesta        192.168.42.3
>
```

However, this does not tells us anything about the mail exchanger or canonical names that we may have defined. To get everything, we us the -d option like this:

```
> ls -d siemau.de

[localhost]

siemau.de.  SOA siemau.siemau.de jimmo.siemau.de. 60001 1080 0 1800 3600000 86400

      siemau.de.       NS            siemau.siemau.de
      siemau.de.       MX      0     vesta.siemau.de
      jimmo            MB            siemau.siemau.de
      siemau           A             192.168.42.1
      jmohr            A             192.168.42.2
      siemau           HINFO         Pentium Linux
      localhost        A             127.0.0.1
      www              CNAME         vesta.siemau.de
      mailserv         CNAME         vesta.siemau.de
      ftp              CNAME         vesta.siemau.de
      vesta            A             192.168.42.3
>
```

As we can see, this gives us everything we can thing about including mail boxes, HINFO lines, canonical names, the SOA records and all of the address records. Note that there is only one MB record here. In reality, I probably would have had MB records for all the users on this system. If this network had been even a little larger, then this output would probably be too much to view. Therefore you can use other options to limit what you see. For example, the -t option is used to specify a type of query. If we wanted to look for all the mail exchangers, we could use the command "ls -t MX siemau.de," which gives us:

```
siemau.de.                          0    vesta.siemau.de
```

Which give us the domain, the preference of the mail-exchanger and the name of the mail exchanger. Which is all the information in the MX record.

We can also tell nslookup that we want to look for particular kind of records. Say I want to look for the MX record for a particular machine. I could set the query type to MX and look for that machine, like this:

```
> set type=MX

> siemau.siemau.de

Server:  localhost

Address:  127.0.0.1

siemau.siemau.de preference = 0, mail exchanger = vesta.siemau.de
siemau.siemau.de internet address =  192.168.42.3
>
```

Okay. This says that the mail exchanger for siemau is vesta. Are you sure? What nslookup is actually telling us is that *vesta.siemau.de* is the mail-exchanger for *siemau.siemau.de*. Why? Because we didn't put the dot at the end of the domain name. Like other aspects of the name server, nslookup tacked the domain name onto the end of *siemau.siemau.de* to give us *siemau.siemau.de*. If I just use a machine name, the domain name is tacked on as well, but it comes out differently:

```
> siemau

Server:
localhost

Address:  127.0.0.1

siemau.siemau.de preference = 0, mail exchanger = siemau.siemau.de
siemau.siemau.de internet address = 192.168.42.1
>
```

The nslookup program also has a configuration file that can come in handy. This is the .nslookuprc file in your home directory. Like the .exrc file for vi, the .nslookuprc is read every time you start nslookup. The format is also like .exrc, with one entry per line. Assuming I wanted to set the query time to PTR records and set the time out to 5 seconds, I could have these two lines in my .nslookuprc file, like this:

```
set querytype=ptr

set timeout=5
```

This would be the same as starting nslookup from the command line like this:

nslookup -query=ptr -timeout=5

Setting parameters is not the only thing that you can do from the command line. In fact, most anything you can do from inside of nslookup, you can do from the command. I could expand the above command to give me:

```
nslookup -query=ptr -timeout=5 192.168.42.3

Server:  localhost

Address:
127.0.0.1

Name:    vesta.siemau.de

Address:  192.168.42.3
>
```

So what is this all good for? The most important thing is tracking down problems you might be experiencing. For example, if a particular machine cannot reached, nslookup might show you that there is no A record for that machine. Perhaps mail to a particular machine never ends up where it should. Checking the MX record for that machine indicates it ends up going to a completely different machine than you though.

Unfortunately, I cannot list every problem that could arise and what nslookup would show. However, with the understanding of how DNS and nslookup work that you've gained in the last couple of sections, the best way to proceed is to look at what nslookup is telling you. Based on the way you think DNS is configured, is what nslookup records correct? This may sound like an over simplification. However, isn't that what problem solving really is? Knowing what should happen and what would cause it to happen differently?

Your Own IP Address

If you have a network that is completely disconnected from the rest of the world, then there is no need for you to adhere to any of these conventions. You might be a commercial organization, but still want to use the EDU domain. Nothing prevents you. There is also nothing preventing you from using IP addresses that are used by some other organization. However, once you decide to connect to another organization or the Internet at large, you need to ensure that both your names and IP address are unique.

If you would like to have one machine that connects to the internet, but have other machines that cannot. One solution is to use one of the IP address defined in RFC 1918. This RFC describes the need for "private" address and lists a range of class A, B and C addresses that can be used internally within a company.

Some routers will filter out this addresses automatically, other require that they be so configured. This allows you to not only limit access to and from the internet, but also limits the need for unique addresses. If you only have a handful of machines that need Internet

access, some internet providers will sub-net a Class C address and assign you a small block of addresses. See the RFC for more details.

In many cases, you Internet Service Provider ISP can apply for an IP address for you. Also, the ISP could provide you with a single IP address that is actually part of their network. Internally, you then use a private network as defined in RFC 1918. In other cases, when you have a dial-up connection to your ISP you may have a dynamically assigned address, so it isn't always the same.

File	Function
/bin/netstat	Show network status.
/bin/hostname	Delivers name of current host.
/bin/ping	Sends ICMP ECHO_REQUEST packets to network hosts.
/etc/dig	Send domain name query packets to name servers.
/etc/ftpaccess	FTP access configuration file
/etc/ftphosts	FTP individual host access file
/etc/ftpusers	List of users automatically denied FTP access
/etc/gated	gateway routing daemon.
/etc/hosts	list of hosts on network.
/etc/hosts.equiv	list of trusted hosts.
/etc/http/conf/*	HTTP configuration files
/etc/inetd.conf	Inetd configuration file.
/etc/named.boot	Name server configuration file
/etc/netconfig	Configure networking products.
/etc/networks	List of known networks.
/etc/protocols	List of Internet protocols.
/etc/services	List of network services provided.
/etc/tcpd	Access control daemon for Internet services.

/sbin/bin/arp	Delivers ARP information.
/sbin/ifconfig	configure network interface parameters.
/usr/bin/finger	Find information about users.
/usr/bin/ftp	Network file transfer program.
/usr/bin/logger	Make entries in the system log.
/usr/bin/nslookup	Query name servers ly.
/usr/bin/rdate	Notify time server that date has changed.
/usr/bin/rdist	Remote file distribution program.
/usr/bin/rlogin	Remote login program.
/usr/bin/route	Manually manipulate routing tables.
/usr/bin/rwho	Who is logged in on local network.
/usr/bin/talk	Talk to another user.
/usr/bin/telnet	Telnet remote login program
/usr/sbin/ftpshut	Shutdown ftp services
/usr/sbin/httpd	HTTP Daemon
/usr/sbin/inetd	Internet ’super-server’
/usr/sbin/routed	Network routing daemon.
/usr/sbin/in.ftpd	FTP daemon
/usr/sbin/in.rlogind	Remote login rlogin daemon
/usr/sbin/in.rshd	Remote shell rsh daemon
/usr/sbin/in.telnetq	Telnet Daemon
/usr/sbin/in.tftpd	Trivial FTP TFTP Daemon
/usr/sbin/in.fingerd	finger daemon
/usr/sbin/traceroute	Trace packet routeis to remote machines.

Table - Network commands

9.2 DHCP

One of the most tedious jobs of any system administrator is configuring each machine so it can talk to the network. In many cases, this means physically going to each machine and making the necessary changes. Even if the changes are accomplished by one of the various configuration programs (linuxconf, yast, etc.), it is still a hassle to have to do this on multiple machines. What makes matters worse is when changes are made to your network, such as changing routes or which machine is your name server, you have to go through everything again. Although such changes hopefully do not occur too often, obviously the more machines you administer, the longer it takes to make the changes by hand.

What is needed is a way to centrally configure and manage the network configuration of all of your systems. This is accomplished using the Dynamic Host Configuration Protocol (DHCP). Even if you are running a network with a handful of machines, then you may want to consider DHCP. It is generally plug-n-play, in that a machine can be added to the network with basically no additional configuration effort on your part, saving your hours of configuration time.

The version of DHCP provided with most Linux distributions is maintained by the Internet Software Consortium (ISC). The package the ISC provides not only includes the DHCP server, but also the DHCP client, as well as a DHCP relay, which allows you to have a central DHCP server which manages multiple networks. The current distributed release is version2, with version 3 already in the beta stage. For more details check out the DHCP page on the ISC web site at http://www.isc.org/products/DHCP/.

The most basic (and perhaps most commonly known) function of DHCP is to assign IP addresses to machines within a network. Although dynamically assigning the addresses is one of the key advantages of DHCP it is not a requirement. That is, you *could* configure DHCP to assign specific addresses to specific machines. For example, I have done it in some networks for the servers, as well as the clients. Each machine was configured to use DHCP, but the servers needed to have static addresses. Rather than configuring the servers themselves with a particular address, we did this using DHCP. That way, should routers, DNS servers, or whatever get changed, we did not need to reconfigure the servers.

DHCP is also useful in environments where you have a people with laptops working in multiple networks, such as people who regularly work out of the home, but still come into the office. Other cases are people who visit many of your branch offices and where these offices are on different networks. If the laptop is configured to use DHCP and there is a DHCP server at each location, the laptop is automatically configured for the local network. In fact, I use it at home on my four-node network, so I don't have to deal with configuring each machine individually (or re-configuring it).

When the DHCP server (**`dhcpd`**) starts it reads it configuration file, which defaults to **`/etc/dhcp.conf`**, but can be changed when the server is started using the -cf option. Using the configuration file, DHCP stores the list of addresses in memory for each of the subnets to which it provides services. When a DHCP client starts, it requests an address. The

server looks for an available address and assigns it to the client. Should the specific client be configured a static address, then it is this address, which is returned to the client.

The assignment of the IP address to the machine is referred to as a lease. Like leases in other contexts, DHCP leases are only valid for a specific period of time. The default is one day, but you can configure it to be most any value. In addition, it is also possible for the client to request a specific lease duration. However, to prevent any machine from holding on to the lease and not letting go, you can configure the server with a maximum lease time.

Depending on your network setup, it may be necessary to limit DHCP to just a single network segment. This could be a problem if the DHCP server is sitting on both segments. However, DHCP can be configured to listen for requests on just specific network interfaces. In addition, if for some reason your system cannot determine whether a specific interface can

Obviously, the DHCP server needs a means of keeping track of the leases across reboots of either the server or the clients. This is accomplished with the dhcpd.leases files, which is typically in the **/var/state/dhcp** directory. After reading the dhcpd.conf file at system startup, the server reads the dhcpd.leases file and marks accordingly those systems that have active leases.

One important thing to note is that unlike other system services, dhcpd does not re-read its configuration file while it is running. Therefore, you need to restart the server by hand each time you make a change. Also, the **dhcpd.leases** file is not rewritten each time the server is started. Instead, the file is maintained across reboots, so that the state of each lease is saved.

Basic Server Configuration

The **dhcpd.conf** file is very straightforward and easy to comprehend. At the top is a header, which contains the global configuration parameters for the server itself and which is applicable to each of the supported subnets, unless specifically overridden. Following that are the configuration declarations for all subnets accessible from the server, whether they are actually provided DHCP services or not.

Configuration is done by various statements in the dhcpd.conf file, which can be either a declaration or parameter. A declaration is used to describe the topology of your network, as it is these declarations that specify for which subnets or even individual host specific configuration is valid. Parameters define the various characters, such as how to do something (which route to take), how to behave (the length a lease is valid) or basic characteristics, such as the IP address.

In its simplest form a DHCP configuration entry consists of a subnet address, the netmask and the range of IP addresses. For example, you might have something that looks like this:

```
subnet 10.2.0.0 netmask
255.255.0.0 { range  10.2.3.0 10.2.3.200; }
>
```

This entry applies to the Class A network 10.2.0.0. However, only addresses in a much smaller network (10.2.3.0) are available. In fact, not all of the addresses in that range are available, since the highest address is 10.2.3.200. Note that each entry is followed by a

semi-colon. Hosts can also be configured individually using the host keyword, which is followed by the name of the host.

In this example, we used the hardware and fixed-address options to define the configuration for this specific host.

These have the general syntax:

```
option option-name option-data
```

What is valid as "option-data" will depend on the option you use. Some are IP addresses or hostnames; others are can be text strings or numbers, while others are simply Boolean values (true/false or on/off). Note that you actually need to include the word "option" to tell the DHCP server that what follows is an option and not a subnet declaration or something else. Keep in mind that if an option specified as a global parameter, it applies to all of the subnets. However as mentioned above, your can also override this value in one of the subnet definitions.

Table 1 gives you a list of the more common dhcpd options. However, there are several dozen more, many of which only apply to specific protocols or services such as NNTP, finger, IRC, and so forth. For a complete list of options with even more details, check out the dhcp-options man-page.

One of the host specific definitions is "hardware" which describes the specific hardware. As of this writing only the "ethernet" and "token-ring" hardware types are supported. Following the type is the physical address of the card (i.e. the MAC address). For example, you might have something like this:

```
host saturn { hardware ethernet 00:50:04:53:F8:D2; fixed-address 192.168.42.3: }>
```

This example says that the machine jupiter has an Ethernet card with the MAC address 00:50:04:53:F8:D2 and it to be assigned the fixed address 192.168.42.3.

Sometimes you need to specify options for a number of machines on your network without having to treat them as a separate subnet. For example, you could define a subnet for a group of machines and then apply specific options just to that subnet. However, this means that you will generally have specify all of the necessary configuration options, plus these special nodes obviously cannot have IP addresses in the same subnet as the others.

To overcome this limitation, you can group machines together using the group keyword. Then, all the options included within the group definition apply to that group. However, it is also possible to specify parameters for specifics. Like subnets, it is also to specify individual hosts with the group. For example:

```
group { default-lease-time 300000; option routers 192.168.42.1;
host  jupiter{ hardware ethernet 00:50:04:53:D5:57;
               default-lease-time 500000;       }
host  saturn { hardware ethernet 00:50:04:53:F8:D2;}
host  uranus { hardware ethernet 00:50:04:53:32:8F;}
}
>
```

In this example, we set the default lease time (how long the lease is valid) for the group to 500000 seconds (almost 6 days) and the router to be the machine 192.168.42.1. This definition applies to the three hosts listed. However, with the host jupiter we set the default lease time to a higher value, but the router definition would still apply.

Another consideration is when there are multiple networks on the same physical network segment. There are several reasons that such a configuration is necessary and the ISC DCHP provides you with the ability to configure your system accordingly. This is done by declaring a shared-network.

A shared network is basically nothing more than a container for a group of machines. One difference to the group declaration is that a shared-network declaration can contain subnets, as well as groups or individual hosts. It has the general syntax:

```
shared-network network-name {
shared-network-specific parameters
subnet {
subnet-specific parameters
    }
   group  {
      group-specific parameters
   }
}
>
```

Note that within either the group or subnet declarations, you can specify parameters for individual hosts, just as when they are not part of a shared-network.

Although the configuration of the DHCP server is pretty straightforward, having to administer a large number of systems by editing files can become tedious. One solution I found useful is Webmin (www.webmin.com). This provides a graphical, web-based interface to a large number of system functions (including DHCP). Figure 1, shows you the primary DHCP configuration page. At the top you see three subnets which this machine managers. Here, too, you would also see any shared networks that are configured.

Underneath are any machines that are specifically configure as well as groups of machines. When you select each object, you are can configure the same options as you can be editing the files. Figure 2, shows you the configuration for a specific host.

Troubleshooting the Server

With complex DHCP configurations it is often difficult to tell which parameter applies to any given host. When trying to figure out what is happening, there are two rules of thumb. First, host or group declarations can specifically override the global definition. In addition, host declarations can override the group declarations.

Second, definitions are not necessarily applied in the order in which they appear in the dhcpd.conf file. Instead, the values are applied starting from the specific moving to the more general. That is, the server first checks for a specific host configuration, then for a group configuration, then the subnet configuration, followed by a shared-network configuration,

followed by the global declarations. Configuration options are only added to and not overwritten. Therefore, the configuration for the smaller, more specific units (like hosts) overrides those of the more general units (like global parameters). So, when trouble shooting, start at the bottom, working your way up.

Perhaps the most basic troubleshooting technique is to look at the leases the server has assigned. This is done by looking at the leases file (**/var/state/dhcp/dhcp.leases**), which maintains the current state of all active leases. One thing to note is that this file is rewritten from time to time to keep the file from growing too large. First a temporary copy is made called and the old file is renamed dhcpd.leases~. Although fairly rare, it can happen that for some reason the server dies at this point. There is no dhcpd.leases file and thus server cannot restart. Rather than simply creating an empty dhcpd.leases file, you need to rename dhcpd.leases~ to get things to work correctly.

The contents of the **dhcpd.leases** file is fairly straight forward. Each lease declaration is identified with the keyword "lease" followed by the IP address and a block of configuration information within curly brackets. For example, you might have something like this:

```
lease 192.168.42.1 {
        starts 0 2000/01/30 08:02:54;
        ends 5 2000/02/04 08:02:54;
        hardware ethernet  00:50:04:53:D5:57;
        uid 01:00:50:04:53:D5:57;
        client-hostname "PC0097";
}
>
```

The starts and ends statements indicate the period when the lease is valid. Each entry is of the form:

```
weekday yyyy/mm/dd hh:mm:ss;
>
```

The weekday is the numerical value for the day of the week starting with 0 on Sunday, as in this case. One key aspect is that the date and time are Greenwich Mean Time and not local time.

The hardware entry is the same as from the dhcpd.conf file and list the hardware address of the card. The uid entry is unique identified for the client, using either a ASCII string client identifier supplied by the client or the hardware address preceeded by hardware type (in this example "01").

Often times the client wants to pick its own name. There are two ways a client can do this. One is using the "Client Hostname" option, as in this example. The other is using the "Hostname" option, which is used by many operating systems, such as Windows 95. In this cases, you would have just the keyword "hostname", which is also followed by the name of the host.

Just because the dhcpd.leases file contains a specific entry does not make it valid. One approach would be to remove any applicable entry (based on either the IP address or hardware address) and then re-start the server. If you end up with the same results, there may be a problem with the way the client is configured.

It may also be necessary to see what the server thinks it is doing as compared to looking at the client or dhcpd.leases file and guessing what the server thinks it is doing. This can be accomplished by running the dhcp server in the foreground (using the -f option) and having it output all of the error messages to stderr (using the -d option) rather than using the system logging daemon. You can then watch the server as it accepts and processes requests.

Client Configuration

Configuring the client-side is fairly straight forward, depending on your distribution. For example, if you are running a SuSE 6.3, then all you need to do is get into the network configuration portion of YAST and select the basic network configuration. Pressing F3 sets auto-IP configuration, which gives you the choice of configuring either DHCP or BOOTP. If you select DHCP, you end with something like figure 2. This makes changes to the file **/etc/rc.config** by setting the configuration parameters for the respective card simply to "dhcpclient". For example, without DHCP you might have an entry like this:

```
IFCONFIG_0="192.168.42.1 broadcast 192.168.42.255 netmask
255.255.255.0 up"
>
```

Once DHCP is configured the entry would look like this:

```
IFCONFIG_0="dhcpclient"
>
```

Note that you could have some of the interfaces configured to use DHCP and others with static addresses. When the system boots, the **/etc/rc.d/network** script is called (for example, as **/etc/rc.d/rc2.d/S05network**). If it finds that the IFCONFIG line for the respective card equals "dhcpclient", it simply skips the configuration of that interface, for the moment. Later in the boot process, the DHCP client script is started (for example, as **/etc/rc.d/rc2.d/S05dhclient**). It is here that the client then tries to get its configuration from the DHCP server.

On other systems, like Caldera or Redhat, have their own configuration tool and change the appropriate file in /etc/sysconfig/network-scripts/. For example, if you were configuring DHCP on your eth0 interface, the script ifcfg-eth0 would be changed. When done you end up with something like this:

```
DEVICE=eth0
IPADDR=0.0.0.0
NETMASK=255.255.255.0
NETWORK=
BROADCAST=0.0.0.255
GATEWAY=none
ONBOOT=yes
DYNAMIC=dhcp
>
```

Find the line labeled simply DYNAMIC= and change it to DYNAMIC=dhcp.

In most cases the default client configuration is sufficient (at least, in my experience). If not, the client has it's own configuration file: **/etc/dhclient.conf**. If you have more than one interface on your system with different network option, you need to group the options by interface. For example, you might have something like this:

```
interface eth0 {
send dhcp-lease-time 3600;
request subnet-mask, broadcast-address, time-offset, routers,
        domain-name, domain-name-servers, host-name;
require subnet-mask, domain-name-servers;
}
>
```

The send statement tells the client to send the particular option with the specified value. These can be any of the options the server understands and are defined in detail in the dhcp-options (5) man-page.

The request statement is a list of configuration options (not the values) that the client requests that the server send. The key word is "request" as the required statement, says that the particular configuration option must be sent by a server in order for the client to accept the server's response.

Security

At first glance, there may not seem to be much to talk about in terms of security and DHCP. However, considering the importance of DHCP, a few precautions are in order.

The first consideration is the machine itself. Although an outage of a couple of hours might be something you can deal with, any long outage means that they may be a number of machines without a valid configuration or even a valid IP address. Therefore, you need to look at what other services the machine with your DHCP server provides. Since there is very little computer power required to support DHCP, you can easily get away with smaller machines. This might be preferable to having the **dhcpd** running along side some other machines. On the other hand, I worked in one company where the DHCP server was also the DNS and mail server for the company.

Another issue to consider is a denial of service attack. If your DHCP server were accessible from the Internet, it would be possible for someone to gobble up all of your IP addresses, leaving nothing for your own machines. Therefore, you should block DHCP traffic through

your firewalls. If your firewall is running on a Linux machine, you can use the ipfwadm program to block port 67 (the DHCP listen port) and port 68 (the DHPC transmit port).

Table 1 - Common dhcpd.conf Configuration Options and Declarations

Parameter	Description	Datatype
default-lease-time	Default length in seconds the lease is valid	Numeric
domain-name	The name of the domain for the specified subnet	Text
domain-name-servers	A list of name servers for the specified subnet.	List of IP addresses
fixed-address	Static addressto assign to a host	List of IP addresses(supports multiple networks)
group	Starts a group declaration	N/A
hardware	The type of hardware the networkinterface has (currently only ethernet and toke-ring are supported)	Harware-type: text Hardware-address: Octets, colon seperated.
Host	Starts a hostdeclaration	N/A
host-name	Name to assign to the requesting host	Text
max-lease-time	Maximum time seconds the server will grant alease should the client request a specific lease	Numeric time
netbios-name-servers	Name of the WINS server	List of IP addresses
range	Range of IP addresses to assign on the specified network	Low and high IP address
routers	A list of routers to use	List of IP addresses
shared-network	Starts a shared network declaration	N/A
subnet	Starts a subnet declaration	N/A
subnet-mask	The subnet-mask of this network, group, host or IP address	

9.3 NFS

The Network File System NFS is an industry standard means of being able to share entire filesystems among machines within a computer network. As with the other aspects of networking, the machines providing the service in this case the filesystem are the servers and the machines utilizing the service are the clients. Files residing physically on the server appear as if they are local to the client. This enables file sharing without the hassle of copying the files and worrying about which one is the more current.

One difference that NFS filesystem have over "conventional" filesystem is that it is possible to allow access to a *portion* of a filesystem, rather than the entire one.

The term *exporting* is used to describe how NFS makes local directories available to remote systems. These directories are then said to be *exported*. Therefore, an exported directory is a directory that has been made available for remote access. Sometimes the term *importing* is referred to the process of remotely mounting filesystems, although *mounting* is more commonly used.

There are a couple of ways you can mount a remote filesystem. The first is automatically mounting it when the system boots up. This is done by adding an entry into /etc/fstab. You could also add a line in some rc script that does a mount command.

If the remote mount is a one-time deal, the system administrator can also mount it by hand. Potentially, the administrator could create and entry in /etc/fstab that does not mount the filesystem at boot time, but rather is mounted later on. In either event, the system administrator would use the mount command. If necessary, the system administrator can also allow users to mount remote filesystems.

A client machine can also be configured to mount remote filesystem on an "as-needed" basis, rather than whenever the system boots up. This is through the mechanism of the automount program. We'll get into a lot of details about how automount works later on.

The syntax for using the mount command to mount remote file system is basically the same as for local filesystems. The difference being that you specify the remote host along with the exported path. For example, if I want to mount the man-pages from jmohr, I could do it like this:

```
mount –t nfs [–o options] jmohr:/usr/man /usr/man
```

Here I told the mount command that I was mounting a filesystem of type NFS and that the filesystem was on the machine jmohr under the name **/usr/man**. I then told it to mount it onto the local **/usr/man** directory. There are a couple of things to note here. First, I don't have to mount the filesystem on the same place as it is exported from. I could have just as easily exported it to **/usr/doc** or **/usr/local/man**. If I want I can include other options like "normal filesystems" such as read only.

If you are a server, the primary configuration file is **/etc/exports**, which is a simple ASCII file and additions or changes can be made with any text editor. This is a list of the directories that the server is making available for mounting along with who can mount them and what permissions they have. In addition, the server needs a way to find the clients address, therefore mounting will fail if the name cannot be resolved either by DNS or **/etc/hosts**. Likewise, the clients depends on name resolution to access the server.

The **/etc/exports** file has one line for each directory you want to export. The left side is the path of the directory you want to export and the right side is options you want to apply. For example, you can limit access to the directory to just one machine or make the directory read only. On *junior*, the exports might look like this:

```
/pub              *
/                 jmohrrw
/usr/jmohr_root   jmohrrw
```

The first line says that I am exporting the /pub directory to the entire world. Since there are no options, this means that the filesystem is also writable. I wouldn't normally do this if I were connected to the Internet, even if there wasn't anything sensitive here. It is a matter of practice, that I know exactly what access I am giving to the world.

The next line says that I am exporting the entire root filesystem to the machine *jmohr*. Since this is a development environment, I have different versions and distributions of Linux on different machines. I often need to have access to the different files to compare and contrast them. Here, the filesystem is also writable as I explicitely said rw for read-write.

The last line takes a little explaining. When I mount the root filesystem from *jmohr*, I mount it onto **`/usr/jmohr_root`**, which is the name of the directory that I am exporting here. This demonstrate the fact that you can export a filesystem to one machine and then have it re-exported.

Keep in mind, however, that we cannot increase the permission during the re-export. That is, if the filesystem were originally made read-only, we could not make it writable when I re-export it. However, if it were writable, I *could* export it as read-only.

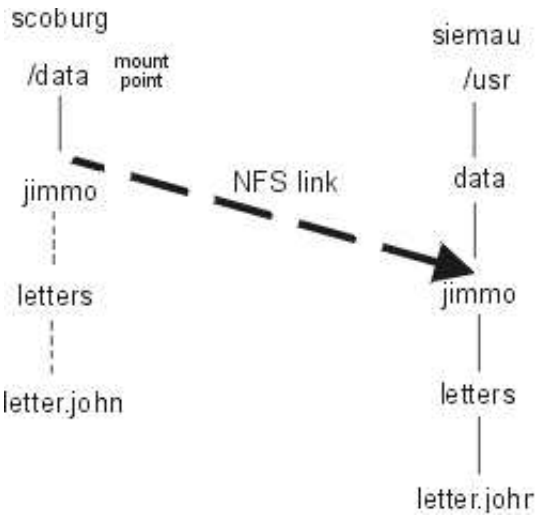


Image - An example NFS mount

A solution that many systems provide is amd, which is an automatic mounting facility for NFS filesystems. Once configured, any command or program that accesses a file or directory on the remote machine within the exported directory forces the mounting to occur. The exported directory remains mounted until it is no longer needed.

If you can access a filesystem under Linux, you can access it under NFS. including DOS filesystems This is because the access to the file is a multi-step process. When you first access a file say opening a text file to edit it. The local system first determines that this is an NFS

mounted filesystem. NFS on the local system then goes NFS on the remote system to get the file. On the remote system, NFS tries to read the file that is physically on the disk. It is at this point that it needs to go through the filesystem drivers. Therefore, if the filesystem is supported on the remote system, NFS should have no problem accessing it. Once a filesystem has been exported, the client sees the filesystem as an NFS filesystem and therefore what type it is, is *really* irrelevant.

There are a couple of limitations with NFS. First, although you might be able to see the device nodes on a remote machine, you cannot access the remote devices. Think back to the discussion on the kernel. The device node is a file that is opened by a device driver to gain access to the physical device. It has a major and minor number that point to and pass flags to the device driver. If you open up a device node on a remote system, the major and minor numbers for that device node point to drivers in the *local* kernel.

Remotely mounted filesystems present a unique set of problems when dealing with user access rights. Because it can have adverse effects on your system, it is necessary to have both user and group ID unique across the entire network. If you don't, access to files and directories can be limited, or you may end up giving someone access to a file that shouldn't. Although you could create each user on every system, or copy the passwd files, the most effect method is using NIS.

9.3.1 The Flow of Things

There are two daemon processes that provide the NFS services on the server. These are mountd and nfsd. **Mountd** is responsible for checking access permissions to the exported filesystem. When a clients tries to mount a filesystem, **mountd** returns a pointer to the filesystem if the client has permission to mount it.

The workhorse on the server side is the nfsd daemon. It has the responsibility of handling all filesystem requests from the clients. Once a filesystem has been mounted, all access to the data on that remote filesystem is made through nfsd. Remember that you could be exporting directories and not just entire filesystems. Therefore it's better to say that access to the *mount point* and below is made through nfsd.

Also key to this whole process is the portmapper, portmap. The portmapper converts TCP/IP port numbers to RPC program numbers. What this means is that when the NFS starts up, it registers its port with the local portmap process. The clients access the server by asking the portmapper on the server for the port number of nfsd and mountd. The port number is then used on all subsequent RPC calls.

In principle, mounting a remote filesystem is like mounting a local one. The general syntax is:

```
mount <options> <filesystem> <mountpoint>
```

One of the primary differences is that since we are an NFS filesystem, we have to explicitly tell mount, by using the '-t nfs option. We can also include other options such as '-r' for read only. Let's assume that we have our two machines jmohr and siemau. On siemau is an NFS filesystem that we want to mount from jmohr. Assuming that the proper entries exist in the /etc/exports file on siemau, the command on jmohr might look like this:

```
mount -t nfs junior:/usr/data /data
```

Like other filesystems, the local mount command parses the command into tokens and ensures that entries don't already exist in the mount table (**/etc/mnttab**) for either the filesystem or the directory. Realizing that this is a remote filesystem, mount gets the IP address for siemau (by whatever means are configured on the local machine) and gets the port number of mountd on siemau. The mount command then passes **mountd** the pathname of the requested directory (**/usr/data**).

Now it's the server's turn. To makes sure it can service the request, mountd must first check **/etc/exports** for the requested filesystem. In this case **/usr/data**. If jmohr is permitted, mountd passes back what is called a file handle, or pointer. Now the mount back on jmohr uses that file handle and the mount point (**/data**) as arguments the mount() system call. Finally, an entry is placed in the local mount table.

There are two primary NFS configuration files: **/etc/exports** and **/etc/fstab**. The **/etc/exports** file exists on the server and list those files and directories that can be accessed by remote hosts. It can also be configured to allow or deny access to specific hosts. Since each of these is a filesystem, you can manage anything mounted by NFS through /etc/fstab. This allows you to mount remote filesystems at boot or in any way you can with a "normal" filesystem. One advantage NFS has over local filesystems is that you can configure them to be mounted only when you need them. That is, if the files in the directories are not used, the connection is not made. However, if the files are needed, the connection is automatically made. This is the concept of automounting, which we will get into next.

9.3.2 When Things Go Wrong

There are a couple of tools that can be used to specifically check NFS connections. Because NFS relies on the same mechanism as other programs using TCP/IP, solving NFS problems start with understanding the tools used for TCP/IP. Rather than repeating myself, I would point to the section on TCP/IP and the various HOWTOs.

If you want see all the programs using RPC on a specific machine, I would run it as:

```
rpcinfo -p <hostname>
```

Which might give me something like this:

program	vers	proto	port	
100000	2	tcp	111	portmapper
100000	2	udp	111	portmapper
100068	2	udp	1027	cmsd
100068	3	udp	1027	cmsd
100068	4	udp	1027	cmsd
100068	5	udp	1027	cmsd
100083	1	tcp	1028	ttdbserver
100005	1	udp	692	mountd
100005	1	tcp	694	mountd
100003	2	udp	2049	nfs
100003	2	tcp	2049	nfs

The columns are:

```
Program-number  version-number protocol port program
```

The program number is the RPC number of that program. You can see what RPC number equates to what program number by looking in **`/etc/rpc`**. Here we see that all the NFS related daemons are running. If we look carefully, we see that there for each program (except walld and rusersd) there is a copy of the program using both UDP and TCP. If you find that one or more of these is not running, then stopping and restarting NFS might work. Otherwise, rebooting should correct the problem. Note that portmapper, mountd, nfs, and status are required.

If you want to check if a particular program is running on the server, this can also be done with rpcinfo. The general syntax for this command is:

`rpcinfo -u <server_name> <program_name>`

For example, if I wanted to check to see if nfsd was running on *jmohr*, I would *not* run it as:

`rpcinfo -u jmohr nfsd`

If I did, I would end up with the message:

```
rpcinfo: nfsd is unknown service>
```

This is because the name of the service in RPC's eyes is nfs. Therefore, the correct syntax would be:

rpcinfo –u jmohr nfs

Which should give you the response:

```
program 100003 version 2 ready and waiting>
```

If you don't see this, then run **rpcinfo** -p to find out what programs are registered.

If you want to find out about what filesystems are being mounted or can be mounted, you can use showmount. On the server, **showmount** -a will show you which filesystem have been mounted and by whom. This will be in the form host:filesystem. If you want to find out what filesystem are being exported and their permissions, use showmount -e. On *jmohr*, I get this:

export list for jmohr.siemau.de:

/usr/man	(everyone)
/usr/lib	(everyone)
/u1	access=siemau
/u2	(everyone)

Each of the filesystem listed is accessible from every other system with the expection of /u1 which is only accessible from siemau. This is essential what is contained in **/etc/exports**.

If a client simply stops responding, it may be because the server is down and you have configured a hard mount. If so, the client may wait indefinitely for the server to come back up. Once it does, the processes can continue as before. If the mount is soft, you will (should) get an error after the number of retries specified. (5 by default).

9.3.3 Automount

If you are used to later versions of Windows, you certainly know how nice it is to be able to access files on remote systems without having to explicitely mount them each time. Occassionally, I get comments from Windows fans that the lack of this feature in Linux is one of it's major shortcoming. Well, that would be true, if Linux was missing this functionality. In fact, this feature has been available for UNIX longer than Windows NT has been around and has been with Linux from the beginning. This is called "automount" or "automounter".

In order to be able to mount a remote filesystem using automount, you would first need to be able to mount it using normal NFS. That is to say that there are no flags that you can set on the remote side (where the filesystem is being exported) that says either to explicitly allow or deny access via automount. The remote filesystem simply makes a resource available and you access it with whatever tool you see fit. Therefore, for the purpose of this discussion, we are going to simply assume that in each case, the remote host has given us permission to access that filesystem. For more details, see the previous section on NFS configuration.

The amd program provides you with the ability to mount NFS filesystems only when you need them, automatically. They are automatically mounted by automount, hence the name. Actually, conventional NFS mounted filesystems can also be mounted automatically in the sense that you can configure them in **/etc/fstab** and they are automatically mounted as the system boots. Automount filesystems, on the other hand, are mounted when a user tries to access files or directories under the mount point. Also, if the files or directories are not accessed within a specific time, (five minutes by default) they are unmounted, thereby saving network resources. When booting you also save time since the system is waiting to connect to a remote machine that possibly could be down.

Keep in mind that the server side is oblivious to the fact that the request is coming from automount. As far as it knows it is just your normal every day NFS mount, therefore automounter can be used with systems that don't know about it.

The basic configuration unit with the automounter is one or more files called "maps." These map the filesystems you want to access to the directory where you want to mount them (the mount points). These map files are fairly easy to configure and can be edited with any text editor. All automount connections are based on references in the map file.

The amd program is command-line based, so you can specify the name of the map file directly. An example map file would look like this:

```
/defaults       opts:=rw;type=nfs jmohr_home    rhost:=jmohr;rfs:=/home
```

The first line specifies default parameters that are used during the mount. In this case, we say that the filesystem should be read-write and of type nfs.The next line specify the directories to be mounted. The syntax for these lines is:

```
directory rhost:=<remote_host>;rfs:=<remote_filesystem>
```

where directory is the name of the directory on the local machine where the filesystem should be mounted, <remote_host> is the name of the machine from which you are mounting the filesystem and <remote_filesystem> is the name of the remote filesystem.

To start amd the general syntax is:

/usr/sbin/amd -a <temp_dir> -- /<real_dir> <map_file>

Here, <temp_dir> is the name of a temporary directory where the remote filesystems are actually mounted. This directory is created by amd and should be removed if you start amd automatically through the rc scripts. A common convention is to define this directory as /amd.The two dash (--) is a common construct and tell amd that there are no more option to process. The /<real_dir> indicates the directory where the users will see the filesystems mounted. Finally, <map_file> is the name of the map file where amd should get its configuration information.

To run amd using the example configuration file, we might start it like this:

```
/usr/sbin/amd -a /amd -- /usr/jmohr/homes /etc/amd.junior
```

Here we mount the remote filesystems under **/amd** and the users see them under **/usr/jmohr/homes**. So, if my home directory on *junior* was **/usr/jmohr/homes/jimmo** every time I log into junior, amd kicks in and mounts the /home directory from *jmohr*.

Don't think of automount as your only means of mounting NFS filesystem just because of it's advantages. If you are constantly accessing certain filesystems, then you gain nothing by making them automounted. In fact, you might lose something since each time the connection is made, you need to wait. If mounted in the conventional manner, then you only need to wait once. If you have filesystems that are accessed regularly, but others that are accessed only on occasion, you simply mount some at boot and the rest via automount.

A common use of automount is with NIS. NIS is used to distribute configuration files from the NIS server across the net to the NIS clients. Why not include the automount maps in the set of files that is being distributed? This could be useful if you wanted to have all the documentation on a single machine to save space and access to the doc is made through automount. Since doc is not being constantly access, this saves the problem of having the filesystem containing the doc be continually mounted.

Another use is when you want each user to have the same home directory no matter where they are. Something similar to what we had in the example above. If mounted by automount and distributed via NIS, every time they logged in, no matter on what machine, they would have the same home directory. Granted, there is the problem of not being able to access their home directory if the server is down. However, the problem of not being able to access the home directory still applies when login into a single machine. For example, if the home directory was on a filesystem that wasn't mounted.

In reality, automount behaves very similarly to traditional NFS mounts. The system knows that the specified directory is an NFS mount point. When something is accesses on the other side of the mount point, the automount daemon reacts to the request basically the same way nfsd does with a normal NFS filesystem. The automount daemon then checks the mount table (**/etc/mnttab**) to see if the filesystem is already mount and mounts it if it isn't. Once the file system is mounted, requests are handled normally.

Like other filesystems, an entry is maintained in the system mount table (**/etc/mnttab**) for all filesystem that have been mounted with automounter. When the timeout has expired (five minutes by default), automounter removes the entry from /etc/mnttab, but still maintains a copy in its memory. This copy is updated whenever mounting or unmounting a filesystem.

The version of amd that I installed with Caldera OpenLinux has some really nifty features. (These feature may be available in other releases, I just didn't test them). The most exciting feature for me required me to reconsider what amd was there for.

I had always used amd (as well as the automounter of other UNIXes) to automatically mount files systems from other machines. That is, mounting them via NFS. On the surface, one thinks of UNIX system administrators as being lazy. They spend all day trying to get out of work. That is, they develop programs that save them work. Never mind that the work that is

saved is much less than the work that they invested in trying to save that work.

This is not a bad thing, for two reasons. First, the mentality of most UNIX administrators is that this *really* isn't an attempt to get out work. There should be a better way of doing something, so they set out to find it. Just like the great inventors and explorers of the past, they try it because "it's there."

The reason that this is not a bad thing is because like other kinds of inventions, the results are there for others. Once the program exists, then other administrators can use it. In keeping with the Linux mentality, you create a program that *you* want to write, share it with others and they share theirs with you. You now have an entire tool chest of utilities that didn't exist a year ago.

So, what does this have to do with amd? Well, how often have you wanted to access a floppy, CD-ROM or DOS partition on your machine? First, you need to find the device that it's on, then find a place to mount it and then finally you get around to mounting it. Wouldn't it be nice if you could access these filesystem without having to go through all of this? You *can*!

That's the nice thing about amd. You can use it to mount all sorts of filesystems, including those on floppy, CD-ROMs and DOS partitions. In fact, that's what I did. I have a handful of applications that do not run under VMWare, so I am forced to switch to Windows 95 when I want to run them. Therefore, I cannot make my entire system Linux. Often, I need to transfer data from one system to other. That means finding the device that it's on, then finding a place to mount it and then finally getting around to mounting it.

What a waste of time!

I noticed that amd was running on my system, using the configuration file /etc/amd.localdev. Seeing that I thought, "localdev? local devices? Hmmm...." Checking in this file, I discovered that you can, in fact, mount local filesystem via amd. So, let's take a look at the file that had on my system:

```
# /etc/amd.localdev : automounter map for local devices

# Don't forget to unmount floppies with "amq -u /auto/floppy" before
# ejecting them, especially when they are mounted read-write !!!

/defaults opts:=nodev,nosuid,ro;dev:=/dev/${key};type:=msdos;

# floppy        -dev:=/dev/fd0;type:=msdos opts:=rw opts:=ro
floppy          dev:=/dev/fd0;type:=msdos;

# floppy95 -dev:=/dev/fd0;type:=vfat; opts:=rw opts:=ro

floppy95 dev:=/dev/fd0;type:=vfat;

cdrom           type:=iso9660
c_drive         dev:=/dev/sda1;type:=vfat; opts=rw
d_drive         dev:=/dev/sda5;type:=vfat; opts=rw
e_drive         dev:=/dev/sdb1;type:=vfat; opts=rw
```

```
f_drive         dev:=/dev/sdb5;type:=vfat; opts=rw
g_drive         dev:=/dev/sdb6;type:=vfat; opts=rw
h_drive         dev:=/dev/sdb7;type:=vfat; opts=rw
*               dev:=/dev/${key}
```

In principle the options are the similar to those for the fstab and exports files. I leave it to you to check amd(8) man-page for the details.

The thing I want to first point out is the last line:

```
*    dev:=/dev/${key}
```

As one might guess from the asterisk, this is a wild card and represents every filesystem that we have not specified previously. Since we did not specify any options, the options used are those in the /default line. The nifty part of this is the ${key} part of the device name. This is translated to mean the name of the sub-directory under /auto. For example, my C: drive is /dev/sda1. If I did cd /auto/sda1, amd would see the name of the sub-directory as sda1, then it would look for a matching device. Since it finds one, it can mount that device. Therefore, if you have mountable filesystem, you do *not* need to explicitely define them.

In this file are two lines for floppies (floppy and floppy95). The type of filesystem in the first line is msdos and this second is vfat, which allows you to mount a floppy and still have access to the long filenames. The next line is one for a cdrom. These were the lines that were originally in my amd.localdev file.

When you start amd, you tell it the name of where it should mount these filesystems. By default, on my system, this is the directory, /auto. So, when I do a cd **/auto/cdrom**, the cdrom is automatically mounted and I find myself in the root directory of the cdrom without going through the hassle of mounting it first.

Once I discovered this, I add the lines that look like this:

```
c_drive         dev:=/dev/sda1;type:=vfat; opts=rw
```

As you might guess, this automatically mounts my DOS filesystems. I originally had them all in /etc/fstab, so they would automatically mount at boot up. Since I only use them occasionally, this didn't make sense. I changed the entries so that they weren't mounted automatically, all I had to do was run "mount /dev/sda1! (or whatever) and they were mounted.

Note in this case that the filesystem type is actually VFAT, and not the standard DOS FAT filesystem. Now, when I do cd **/auto/c_drive**, I find myself on the C: drive of my Windows 95 system. (/dev/sda1) When I do cd **/auto/g_drive**, I am then on the G: drive.

Being a normal Linux system administrator, this was too much for me. Not that the work was too much, I had too many entries in there that were basically all the same. In essence, the default behavior that I wanted was that I would cd into a directory under /auto and I would be on one of my Windows 95 drives. Therefore, I could change the /default like to look like this:

```
/defaults opts:=nodev,nosuid;dev:=/dev/${key};type:=vfat;
```

Note that I not only changed the filesystem type to vfat, but I also removed the options to say that this was a read-only filesystem (ro). Now when I do a cd /auto/sda1 I am on the C: drive or /auto/sdb6 I am on the G: drive.

Hmmmm. How do I know that /dev/sdb6 is the G: drive? Trying to figure that out each time is as much work as mounting it by hand. (Well, not quite.) To save me some work, I simply created a handful on links in /dev that look like this:

```
lrwxrwxrwx   1 root     root            4 Jan 26 10:46  /dev/c_drive -> sda1
lrwxrwxrwx   1 root     root            4 Jan 26 10:46  /dev/d_drive -> sda5
lrwxrwxrwx   1 root     root            4 Jan 26 10:46  /dev/e_drive -> sdb1
lrwxrwxrwx   1 root     root            4 Jan 26 10:46  /dev/f_drive -> sdb5
lrwxrwxrwx   1 root     root            4 Jan 26 10:46  /dev/g_drive -> sdb6
lrwxrwxrwx   1 root     root            4 Jan 26 10:46  /dev/h_drive -> sdb7
>
```

Now, I all need to do is cd /auto/g_drive and I end up on the right drive.

One might ask what advantage this has over traditional mounts via /etc/fstab. When dealing with NFS mounts across the network, there is the issue of bandwidth and the problems that occur when the server side goes down. Since you are accessing a local drive, there are no issues of bandwidth and if the server goes down, the client goes with it.

Well, that last part is the point. If I only need to access the filesystem for a short time, it is safer to umount when I am done. By using amd, I don’t need to worry about forgetting it. After specific period of time (default: 5 minutes), the system will unmount any unused filesystems.

File	Function
/etc/amd.local	Automount configuration file for local devices
/etc/exports	NFS files systems being exported
/usr/sbin/rpc.portmap	Portmapper. Converts DARPA ports to RPC program number.
/usr/sbin/rpc.mountd	NFS mount request server.
/usr/sbin/rpc.nfsd	NFS daemon to handle client requests.
/usr/sbin/amd	The automount program.
/usr/sbin/rpc.rusersd	Network user name server.
/usr/sbin/rpc.rwalld	Network wall server.
/usr/bin/rpcinfo	Reports RPC information.
/usr/bin/rusers	Reports information on network users.
/usr/bin/rwall	Write to all users on the network.
/usr/bin/showmount	Shows information on remotely mounted filesystems.

9.4 SAMBA

Due to the incredibly large number of Microsoft applications, it is almost expected that a server be able to provide some kind of access to the Windows machines. If you have TCP/IP running on your machines, then you have connectivity from your Windows machines to a Linux server through telnet and ftp. However, you do have access to the file and print services provided by an NT server. Or do you?

The network protocol used by Windows NT networks is the Session Message Block or SMB. This is the same protocol that Microsoft has been using for years with their Lan Manager product. Therefore, anything that can access a Lan Manager server, will be able to access an NT server.

Recently, Linux has been able to support SMBs through the SAMBA package. This not only means that a Linux machine can provide file and print service to Windows machines, but they can also provide the same services to a Linux machine.

Because Linux and Windows have a different understanding of security and approach it in a different way, you have to be careful in what you make available. However, in keeping with the UNIX tradition of configurability, you have a wide range of choices on how to configure your system. The are many different options to define your security as well as what you make

available and how.

SAMBA has both a server and a client component and as one might expect the client side is fairly straightforward. Fortunately, to get the server side working, you don't have to do too much. On the other hand, there *are* a lot of options that you can use to configure the system to fit your needs.

There are five primary files that SAMBA uses. The SMB daemon is **smbd**, which is the SMB server. This is what is what is providing the services. The client side is the program smbclient. This is what allows you to access Windows machines from your Linux workstations and servers.

The **nmbd** program is the NetBIOS name server daemon.

SAMBAs configuration file is **smb.conf** which you should find in the **/etc** directory. This file has a format that is similar to Windows INI files, in that it is broken up into section. Each section has a particular function. For example, the section that describe the printer configuration would start like this:

```
[Printers]
```

Like the Windows ini files, smb.conf is line based, with one option per line. In general, the configuration options have the format:

```
option = value
```

Finally, there is the **testpararm** program, which is used to test the SAMBA configuration.

The first section of smb.conf sets up the global configuration and is called, [global]. The default looks something like this:

```
[global]
   printing = bsd
   printcap name = /etc/printcap
   load printers = yes
;  Uncomment this if you want a guest account
;  guest  account = pcguest
   log file = /var/log/samba-log.%m
   lock directory = /var/lock/samba
   share modes = yes
```

The printing option determines how the printer status information is interpreted. The default is bsd, which is the most common used. For information on the others, see the smbd man-page.

The printcap name option defines the name of the printcap file. In this example, we are using the standard printcap file. If you use a different printing option, the printcap file could be different.

The load printers options determines whether or not all the printers listed in printcap are loaded for browsing.

The guest account is used for enabling access somewhat anonymously. By default this is disabled, as in this example. To enable it, uncomment the lines by removing the semi-colons. The actual effects of this account we'll get to later

The log file option defines what file to write diagnostic messages and other information. Note that there is a %m at the end of the file. This variable is the netbios name of the client machine. This can be used in other places, as well. In this case, there will be one log file for each client.

The lock directory options define the directory where lock files are put. The share modes option is used to enable/disable the "share modes" when files are opened. This allows clients to get exclusive read or write of files. Since this is something not directly supported by Linux, smbd uses locks in the lock directory.

There are primarily two types of services that you access via smb: file and print services. File services are directories that have been made available. Normally, all sub-directories under the exported or *shared* directory are accessible. Lets assume that we want to make the users home directory available. To do this we *share* them. The section might look like this:

```
[home_dirs]
   comment = Home Directories
   path=/home
   read only = no
```

Here the section name is [home_dirs] but we specify the path of **/home**. So when we are on Windows machine (e.g. Windows 98) we would see this directory as homes. (Since that is the share name) In this example, we specified the directory as not being read only. This would be the same as specifying the directory as writable. (Note that the section [homes] is a special section, which we will get to shortly)

We can also define access for specific users. For example, if I wanted to give each user access to their own home directory and no other, I could create shares for each directory. So, the share for jimmo's home directory might be:

```
   [jimmo]
   comment = Jimmo's Home Directory
   path=/home/jimmo
   valid users = jimmo
   writable  = yes
```

Here we added the valid users option, which, as one might expect, indicates which users can have access to this directory. Since the only user specified is jimmo, only I can access this directory. Note that if I were to add the option browseable = no and use the Windows browser (not the same as a Web browser), I would not not be able to see the share, even though I could explicitely connect to it. If it were browseable, but I was not in the list of valid users, I could see it, but would then be asked for your password. Since I was not in the list of valid users, no password I enter would work. (At least this is the behavior from Windows 98)

Setting up a printer is the same basic process. We define a resource that we share and specify what access is permitted. An example, might look like this:

```
[deskjet]
   comment = Deskjet on Junior
   path = /tmp
   printer = lp
   public = no
   writable = no
printable = yes
```

Here the path is not the path to the resource, but rather a world-writable directory that has the sticky-bit set. This is the spool directory and allows us to write to it, but no one can remove our files. In this example, I just used **/tmp**. However, you could create a special directory for this purpose, for example **/var/spool/public**. We could also specify a list of valid users for this printer.

The [homes] section is used to connect to users home directories on the fly, without having to create individual shares as I suggested above. When you try to connect to a service with the same name as a user, **smbd** will do one of two things. If there really is a service of that name, smbd will connect to it. If there is no such service, smbd will treat it like a user name and look for that users home directory in the passwd file. Next, smbd will create a "virtual" share by making an internal copy of the [homes] section, but using the users name as the share name. An example [homes] section might look like this:

```
[homes]
    comment = Home Directories
    browseable = no
    read only = no
    create mode = 0750
```

Note that there is no path specified in this example. This is because, the path for the share is the home directory specified in **/etc/passwd**. However, we could have specified a path and although the name of the share would be that of the user, the path would be whatever we specified.

We could still specify the path using the %S variable. This is the name of the current service, which, in this case, is the name of the current user. This could be done like this:

```
path=/data/pcusers/%S
```

This might be useful if we give the users one home directory when accessing from a PC and another home directory when directly logging into the Linux machine.

The [printers] section behaves similarly to the [homes] section. If you try to connect to a service, smbd will first look for a service with the specific name. If it finds one, that's what it will connect to. If there is none, it will look through the printcap file. If there is a matching printer name in the printcap file, this is the one that will get used. Otherwise, an error is returned. An example, [printers] might look like this:

```
[printers]
   path = /var/spool/public
   writable = no
   public = yes
   printable = yes
```

Note that this service is defined as not writable. However, it is defined as being printable. This is necessary for correct operation.

To a limited extent, your Linux machine can be configured as an SMB client. This is done using the smbclient program. This provides a functionality similar to ftp in that you can copy files from one machine to another. The general syntax would be:

smbclient \\\\server\\service</P>

Note that there are twice as many back-slashes as normal. This is because the shell will first try to interpret the slashes, so we have to escape each one with a back slash. Alternatively, we could enclose the complete service name inside of single quotes, like this:

smbclient \\server\service

At this point we have configured SAMBA to allow us to connect from a Windows machine, or any other machine that understand SMBs. The good news is that's not all. SAMBA has many more configuration options. The bad news is that we don't have space to cover them all. The first place to start looking is the **smbd** and **smb.conf** man-pages. These both go into a lot of detail about how you can configure SAMBA.

There are a few configuration options that I would like to address. The first set have to do with security. SAMBA allows you to specify access based on both user (as described above) and by host. This is done with the allow hosts option. Like the valid user list, the list of authorized hosts is comma delimited values. In this case, the IP address or host name. You can globally define access by including the list in the [global] section or you can define it for specific services.

Using the IP address, you could specify entire networks by simply including just the network portion of the IP address. For example:

```
hosts allow = 192.168
```

We could also exclude specific addresses. For example:

```
hosts allow = 192.168 EXCEPT 192.168.42
```

This allows access from the class B net 192.168, but denies access from any machine on the class C network 192.168.42.

Supplemental to hosts allow is hosts deny, which, as you might guess, is used to specifically deny access. This has the same syntax as hosts allow. Note that access defined here is the first phase. If you deny access to a particular machine, that's it. They cant get any further. However, if you allow access they can still be restricted at the user level.

You could also use a list of invalid users to restrict access. For example, you could define root as an invalid user. This would prevent root from accessing any service, which is helpful if you leave a security hole open. You can also include users that are part of Linux group by preceeding that name with an at-sign@. For example, to keep root and members of the adm group out, the line might look like this:

```
invalid users = root,@adm
```

Note that any place that you have a list of users, you can use the @ construct to include group names.

On some cases, there are directories that you don't want users to have access to, although you allow them access to their parent. This is first accomplished by setting the permissions on the directory. However, this is not always a valid alternative. For example, you may not want users to have access to the /dev directory, but you cant change the permissions. However, you can use the don't descend option to prevent them from accessing through SAMBA. This options takes a common separated list. For example:

```
don't descend = /proc,/dev
```

The guest ok option (and its synonym public) are used to give access to services without requiring a password. Once connected, access will be the same as the guest account. This is a boolean option with a default of no.

Another useful option give you the ability to define a configuration file based on the user or machine that is connecting. This is done with the config file option, which appears in the global section. Setting the configuration file based on the machine name might look like this:

```
config file = /etc/smb.conf.%m
```

When you connect, smbd will look for a matching configuration file. If it finds one, it will *reload* the new one. If there is no match, the original configuration stays in effect. This is useful for defining a configuration for some machines (or users) and not for others.

To make management easier, especially if you have a lot of shares for a lot of different machines, using the include option you can have one main configuration file and include the configuration files for specific machines and users. By using variable substitution, you can create files based on machine or user name and then include them. For example, to include files based on machine name, the directive might look like this:

```
include /etc/smb.com.%m
```

Also helpful to manage your services is the copy option, which allows you to copy services (what else?). This way you can create services with different names but have the same basic configuration options. The syntax is simply:

```
copy = service
```

Where service is the name of the service that you want to copy.

Two things to note. First, the "original" must appear first in the configuration file. Second, you can change parameters in the copy and they will override anything in the original. This would be useful to create a kind of template, which , for example, defines the access permissions, but each service would define its own path.

The guest account option is used in conjunction with the guest ok and defines a user name that will be used during the connection. This user *must* exist in the password file, but it is safest that they don't have a valid login (i.e. have **`/bin/false`** as their shell).

You can limit access further by defining a service as guest only. This means that although a user has a valid password to access a particular service, they are limited to whatever the guest account has access to.

If you have been working with UNIX and DOS/Windows machines at the same time, you probably have experienced something that is affectionately called file name *mangling*. This results from the fact that UNIX file names can be longer than the standard DOS 8.3 names.

You can configure Samba so that it will " mangling" the DOS names for you, so they are accessible by DOS and Windows. (Note that you don't need to do this with Win98 or WinNT). Depending on what your needs are, you can mangle names in different ways.

One difference between UNIX and DOS file names that always give me trouble is that DOS doesn't see a difference between upper a lowercase. Therefore, the files LETTER.TXT and letter.txt are the same in DOSs eyes. The mangled names option (yes/no) is used to turn this functionality on and off. If enabled (yes), names are mangled according to the rules you define. Otherwise, they are invisible. (See the sbm.conf man-page for details on how names are mangled.)

To mangle the name correctly, you first need to define a default case with the default case option which can either be upper or lower. Using the mangle case option (yes/no) you tell smbd to mangle the characters that are not the default case.

You can also set the case sensitive option (yes/no), which control whether the name are case sensitive or not. If not (the default), smbd must do a name search to access the appropriate file. When you create a new file, the preserve case option (yes/no) defines how smbd will behave. The default is no.

The mangled map option allows you to define specific changes to files. This is useful when mangling of file names is not the best choice. The classic example is Web pages. On UNIX machines, these normally have the extension .html. However, this does not fit with the 8.3 convention of DOS file names. Normally, DOS would come up with some weird, mangled name. You can keep the name somewhat normal by using a mangled map. For example:

```
mangled map = (*.html *.htm)
```

This means that whenever smbd files a file with an extension of .html it will automatically convert the extention to .htm.

The good and bad news is that's not it. The smb.conf file contains dozens more options. Many of which you require a very strong imagination to come up with uses for. However, this is in keeping with the Linux tradition that you are only limited by your imagination.

Command	Function
/usr/sbin/smbd	SMB Daemon
/usr/sbin/nmbd	Netbios name server daemon
/usr/bin/smbclient	ftp-like SMB client
/usr/bin/testprns	Check printer name for use with SAMBA
/usr/bin/testparm	Check/test SAMBA configuration files

Table - Key SAMBA files

9.5 Accesing the Web

It was difficult to decide where to put this topic. You can't have access to the Web without networking, however, it looses much of it's impact unless you are using a graphical interface like X. Because the Web is a network of machines accessed in a common manner, I figured the networking chapter would be the best place to talk about it. I think this is a good choice since, there are character based programs that do not require X.

So what *is* the Web. Well, as I just mentioned, it is a network of machines. Not all machines on the Internet are part of the Web, but we can safely say that all machines on the Web are part of the Internet. The Web is the shortened version of World Wide Web, and as its name implies it connects machines all over the world.

Created in 1989 at the internationally renowned CERN research lab in Switzerland, the Web was originally begun as a means on linking physicists from all over the world. Because it is easy to use and integrate into an existing network, the Web has grown to a community of tens of thousands of sites with millions of users accessing it. With the integration of Web access software, on-line services have opened the Web up to millions of people who couldn't have used it before.

What the Web really is, is a vast network of inter-linked documents, or resources. These resources may be pure text, but can include images, sound and even videos. The links between resources are made through the use of the concept of hypertext. Now, hypertext is not something new. It has been used for years in on-line help systems, for example, like those in MS-Windows' programs. Certain words or phrases are presented in a different format often a different color or maybe underlined. These words or phrases are linked to other resources. When you click on them, the resource that is linked is called up. This resource could be the next page, a graphics image, or even video.

Resources are loaded from their source by means of the hypertext transfer protocol, HTTP. In principle, this is very much like ftp, in that resources are files that are transferred to the requesting site. It is then up to the requesting application to make use of that resource, such as display and image or playing an animation. In many cases, files are actually retrieved using ftp instead of HTTP and the application simply saves the file on the local machine.

The application that is used to access the Web is called a Web browser. Web resources are provided by *Web Servers*. A Web Servers is simply a machine running the HTTP daemon: **`httpd`**. Like other network daemons, httpd receives requests from a Web client such as Mozilla or Konqueror and sends it the requested resource.

Like the **`ftp`** daemon, **`httpd`**, is a relatively secure means of allowing anonymous access to your system. You can define a root directory, which, like ftp, prevents users from going "above" the defined root directory. Access to files or directories can be defined on machine basis and you can even provided password control over files.

When httpd starts, it reads its configuration files and begins listening for requests from a document viewer one that uses the HTTP protocol. When a document is requested, httpd checks for the file relative to the DocumentRoot defined in srm.conf.

Web pages are written in the Hypertext Markup Language HTML. This is "plain-text" file that can be edited by any editor, like vi. Recently, as a result of the increasing popularity of the Web, dozens, if not hundreds of commercially available HTML editors have become available. The HTML commands are similar, and also simpler, that those used by troff. In addition to formatting commands, there are build in commands that tell the Web Browser to go out and retrieve a document. You can also create links to specific locations labels within that document. Access to the document is by means of a Uniform Resource Locator URL.

There are several types of URLs that perform different functions. Several different program can be used to access these resources such as ftp, http, gopher, or even **`telnet`**. If you leave off the program name, the Web browser assumes that it refers to a file on your local system. However, just like **`ftp`** or telnet you can specifically make references to the local machine. I encourage using absolute names like that as it makes transferring Web pages that much easier.

All that you need to access the Web is an Internet connection. If you can do **`ftp`** and **`telnet`**, then you can probably use the Web. So, assuming you have a Web browser and an Internet connection. The question is where do you go? The question is comparable to "Given a unlimited value plane ticket, where do you go on vacation?" The sky is the limit.

As I mentioned, the convention is that the Web server's machine name is *www.domain.name*. To access their home page, the URL would be *http://www.domain.name*. For example, to get to your home page, the URL is *http://www.yourdomain.com*. In order to keep from typing so much, I will simply refer to the domain. name and you can expand it out the rest of the way. In some cases, where the convention is not followed, I'll give you the missing information.

I remember when comet Schumaker-Levy 9 was making history by plowing into the backside of Jupiter. The Jet Propulsion Laboratory has a Web site, on which they regularly updated the images of Jupiter. I still remember my friends asking me if I had seen the "lastest" images. If they were more than three hours old, I would shrug them off as ancient history.

If you are interested in *free* software did I say the magic word?, check out *www.cdrom.com.* You can download gigabytes worth of games and utilities and GIFs and ray-traces and source code and full-text copies of Alice in Wonderland. Most of these are available from sites spread out over the Internet. It's really nice to have them all it one place. The machine www.cdrom.com is the Web server for Walnut Creek CD. Although you could spend weeks downloading their archives, you don't need to. The CDROMs they offer are very reasonably priced. Use the Web site or you can even ftp to ftp.cdrom.com and get access to many of the CDs that they offer. I liked some of those that I found so useful, I subscribed. This saves you about $10 per CD and you get quarterly updates of many of them.

Another place that contains similar information is InfoMagic. While their offering is similar to Walnut Creek CD, InfoMagic does provide a few that Walnut Creek doesn't. They can be reached at www.infomagic.com.

One of the CDs that I found very useful was the Internet Info CDROM. This contains a wealth of information about the Internet. This includes standards that are applicable to the Internet like IEEE and RFCs. There are also Frequently Asked Questions FAQ from some of the Usenet newsgroups.

The issue of Usenet *newsgroups* opens up a whole can of worms. Without oversimplifying too much, we could say that Usenet was the first, nation-wide on-line bulletin-board. Whereas the more commercial services like CompuServe store their messages in a central location, Usenet is based on the "store and forward" principle. That is, messages are stored on a message and forwarded to the next at regular intervals. If those intervals are not all that often, it may be hours or even days before messages are propagated to every site.

Messages are organized into a hierarchical, tree structure, very much like many things in UNIX. although you don't have to be running a UNIX machine to be accessing Usenet. Groups range from things like rec.arts.startrek.fandom to alt.sex.bondage to comp.unix.admin.

Although I would love to go into more details, this really goes beyond the scope of this book. Instead, I would like to recommend *Using UUCP and Usenet* by Grace Todino and Dale Dougherty, and Managing UUCP and Usenet by Tim O'Reilly and Grace Todino, both from O'Reilly and Associates. In addition, there is a relatively new book that goes into more details about how Usenet is organized, what newsgroups are available and some general information about behavior and interaction with other when participating in a Usenet sendmail. This is *Usenet Netnews for Everyone* by Jenny Fristrup, from Prentice Hall.

Note that the browser that is provided on the CD does not support many of the advanced features of some of the commercials ones, such as Netscape. Because there is a Netscape version that runs on Linux, I would suggest that you get it before you get too involved in the Web. You can get the Netscape Navagator via anonymous FTP from ftp.netscape.com.

9.6 Firewalls

If you are connecting yourself to the Internet at all, then you need to consider protecting your internal network. If you set up an Internet server that has only one network interface, that being to the Internet, then your internal network is safe from intruders. This solution provides

the ultimate in security (short of not connecting at all), but requires more work on your part to transfer data to and from that machine.

The alternative is to implement a system by which someone from the outside has the least chance of getting into your network, but you have the ability to get to and through that gateway machine. This is the concept of a *firewall*.

Like a firewall in a building, a network firewall functions more or less as "damage control". Should a fire break out, the firewall prevents it from spreading further. With a network firewall, should an intruder break into the gateway machine, the potential for further damage is limited.

In the following section, we will be talking about the basic issues involved with implementing a firewall on your machine. Keep in mind that these are the *basics*. There is more to it if your want to make your system as safe as possible. In comparison to car security, this is analogous to telling you about locking your door or installing an anti-theft device like "The Club", but we won't go into details like electronic devices that cut the gas flow if the car is "hot-wired".

First let's briefly, talk about the firewall itself. Your firewall is a machine that has routing capabilities. Technically, it does not need to be a computer, but can be something as simple as a router. As it's name implies, a router is a machine that routes packets. Most routers allow you to not only configure where packets are allowed to go to, but also from where they are allowed to come. In addition, many can be even more finely tune it, such as limiting the type of packets and to/from what ports.

If you use a computer as your router, it needs to have routing capabilites such as Linux machine. Because of the features such as FTP and HTTP services, a Linux machine is well suited to the task. This explains why approximately 10% of all Internet servers are running Linux.

One way of making the firewall safe is to prevent all packets from going *through* it. This means that packets are not *forwarded*. Certain routers will allow you to configure them in such a way that incoming packets are blocked, but outgoing packets are let through.

Note that this does not mean that you can only send packets to the outside, but never hear their answer. Instead that means that packets that *originate* on the outside are blocked. When you send a request from an internal machine, the packet that you receive is a response or acknowledgment. Such packets are allowed through, but those that are sending the initial request are not.

This has a slight danger in that acknowledgment packets can be intercepted and manipulated. This requests detailed knowledge of both the TCP/IP protocols and the application involved, so it is not something that the casual hacker is going to try. However, it is *possible*.

To increase the security of your system, you turn off packet routing altogether. This means that no packet is let through. In such as system, you need to connect first to the firewall machine and *then* to the Internet. At this point, you now have two distinct networks. Although the firewall can see *both* of them, neither can see the other. Since nothing gets through, not even email, your internal network is (fairly) safe.

This means that an intruder cannot reach your internal network, he must first reach the firewall and use it as a "springboard" to get inside. Unfortunately, this also means that people on the inside cannot reach the Internet without first getting to the gateway machine. If this is your choice, then the security is more important than the slight inconvenience that the users encounter.

The alternative is what is called a *proxy server*. In essence, this functions like a translator. When packets reach the proxy server, they are redirected to another connection on the other side of the firewall. For example, **`httpd`** (your web server daemon) normally listens on port 80. However, this has been disabled. Instead, I connect to port 1080 (or something greater than 1024). The services that is listening on that port makes a connection to port 80 on the destination machine to complete the HTTP connection. This middleman, or proxy, does the translation for my application. Other than knowing that I need to connect to the proxy server first, the functionality is the same as if there were no firewall.

There are a couple of ways of implementing the proxy under Linux, which we will get to later.

9.6.1 Securing the Server

A key aspect of a firewall is the security of the firewall itself. If it is not secure, it is comparable to locking all the doors to your house, but leaving the key under the mat. Therefore, taking the key with you, or for that matter throwing it away is a much safer alternative.

So what do I mean when I say "throwing away the key"? Well, what I mean is that you eliminate all potential routes that an intruder can use to get through the firewall or from the firewall to the internal network. Granted, your security should be sufficient enough that the intruder cannot get into the firewall in the first place. However, you need to plan for that possibility.

The question is not whether I am too paranoid, but rather whether I am paranoid enough. To me, not making the firewall machine secure is comparable to locking all the doors and writing your safe combination on the wall next to it. Okay, the house is secure, but *should* someone break in, they have free run of all your valuables.

So, let’s first talk about locking the doors.

The purpose of your Internet gateway is to provide a gateway to the Internet. Sounds simple enough, but what that means is not always clear. The question you need to answer is "What is the purpose of the Internet connection?" The answer to which will define what steps you take to secure the firewall.

For example, let’s assume that you have a Internet connection so that your employees can connect to the Internet, but do not provide any services yourself. In this case, you do not need to enable services *on the firewall* such as FTP or HTTP. All that happens is packets are routed through the firewall. Therefore, you can remove the daemons themselves (i.e. **`telnetd`**, **`ftpd`**, **`httpd`**, etc) and the programs (**`telnet`**, **`ftp`**, etc.). To simply disable them, place a pound sign in front of the appropriate entry in /etc/services.

This is where a lot of controversy occurs. In a article I wrote for a number of years ago, I describe the procedures as "throwing away the key", in that the programs and daemons were physically removed from the machine. Some people disagree and say to move them to a place that the normal user would not have access to. The reason being the difficultly in administering the system should these programs be needed.

Despite the respect I have for their technical capabilities, I have to disagree with him on this point. Although he is correct that it does make the administration more difficult, there are two important security issues involved. First it makes hacking more difficult. If these programs are not on the machine, they *cannot* be compromised. You have, in essence, thrown away the key.

The other issue is that hackers are *not* normal users. They are sneaky, plodding and patient. Maybe you have hidden the file on the system, but a good hacker isn't thwarted by such simple tricks. I wouldn't be and I am not even a good one. If a program was not in it's original location, the first thing I would do is to see it was anywhere on the system. I have seen administrators simple rename the file to something like telnet.orig so simply starting telnet, does nothing.

My attitude is that it is better to err on the side of inconvenience than on security. If you discover that access is too inconvenient, you can always change it. However, if you discover that the security is too lack, it's too late.

I apply this same attitude when it comes to the services that I make available. If you remember for the networking chapter, the available services are defined in /etc/services. My suggestion is to first comment out *every* service. Then, one-by-one, uncomment those that you want. Yes, you may forget one and cause a certain amount of inconvenience. However, doing it this way you know exactly what services you are enabling. If your forget to enable one, you have inconvenienced someone. If you do it the other way, by disabling the ones you do not want, if forget to disable one, you may have let the would-be hacker into your system.

Another means of securing your system is to limit access to the machine. There is, of course, the issue of the physical security of the machine. If someone has physical access to your firewall all the network security in the world is of little value.

What you turn on depends on what your needs are. For example, if you ware providing FTP and HTTP services these two entries should be uncommented. (Note this assumes that httpd is running from inetd and is not stand-alone). I would definitely say that on the firewall, you should *not* uncomment **`netstat`**, **`systat`**, **`tftp`**, **`bootp`**, **`finger`**, and **`ntp`**. There is no reason that I have ever heard of to make these services available across the Internet. Personally, I think you should also leave out telnet and login (for rlogin). The key is to only give what you *have* to.

To set this all up there needs to be a couple of changes in the kernel. Obviously, the basic networking functionality needs to be turned on, but there are two other options that you will find useful. The first is IP_FORWARDING. This needs to be turned *off* so that packets are not passed through the firewall.

The next is IP_FIREWALLING. This is necessary for the basic firewall functionality. If you want to keep track of the IP packets, then you need to turn on IP accounting. I recommend doing this because it allows you to keep track of from where the packets are coming and where they are going.

Another concept is the idea of IP masquerading. With this enabled, the internal network is "invisible" to the rest of the world. What happens is the firewall server converts the IP addresses so that machines on the Internet think they are communicating with the firewall and not an Internal machine. You can then assign IP address to your internal network that reside with the private ranges as defined by RFC 1918. IP_FIREWALLING must be enabled for this to work.

If you plan to use your firewall as springboard, then the configuration is basically complete. However, if you are planning to go from your workstations through the firewall, then there is more that you need to do do. This is where the concept of a proxy comes in.

There are currently two well known proxy packages available for Linux. The first is "socks", which operates as a full proxy and can work with any program. There is a single configuration file and a single daemon that need to be configured.

The other is the TIS firewall toolkit, which requires a new version of each daemon that will be going through the firewall. If you do not want users to have access to a particular service, then you just don’t provide them with the necessary programs. However, with socks, it is easier to unintentionally allow access.

Although not really a firewall, the TCP Wrapper program can be used to control access to and from a *specific* machine. That is, it must be installed on each machine individually. In contrast, a firewall works for all machines that are trying to get passed it to the Internet.

The socks proxy server is available on many of the newer distributions. If you don’t have it you can get as a gzipped tar archive from ftp://sunsite.unc.edu/pub/Linux/system/Network/misc/socks-linux-src.tgz.

It is here that we have come to the point where we have locked the door and put "The Club" into our steering wheel. The Firewall HOWTO goes into details of the actual configuration of the proxy server and the associated files.

9.6.2 Securing the Internal Network

How secure the Internal network should be is another issue that I have had "heated discussions" with my co-workers about. They argue that if we "make sure" that the firewall is secure, then we don’t need to worry about the security on the internal network. To me this is the same issue as locking the front door, but writing the safe combination on the wall. Based on my hacking experiences, I think that it is unwise to take anything for granted.

Here again, you need to weigh security with convenience. In most cases, the inconvenience of slightly slower connections or an extra two seconds to login is negligible compared to the damage cause by a malicious intruder. The best approach is to address those issues that we talked about earlier, including implementing the private IP address as defined in RFC 1918.

In addition, you should very much be considering implementing the same security on the Internal machines as you would on your gateway. The reason is security. *If* any intruder breaks into the gateway and *if* they can then get into the internal, how safe of the other machines. If you left holes open on the gateway, the odds are the holes are on the internal machines as well.

9.7 Network Technologies

9.7.1 Ethernet

Linux supports two of the major network types: Ethernet and token-ring. Ethernet could be labeled as the great grand-father of all the other network types. It was developed in the 1970s by Xerox for linking computers to printers. Although not very wide spread at first, Ethernet has since expanded to be (perhaps) the most widely spread type of network.

The principle behind Ethernet is called Carrier Sensing, Multiple Access with Collision Detection (CSMA/CD). What this means is that every machine on the net sits quietly listening for messages. When one of the machines needs to talk, it waits for a pause and jumps in to send its message. What if two machines simultaneously see the pause and start to send? Well, a collision occurs. This is detected by both machine which wait a random amount of time before they will try again. Although the random amount of time could be the same for both machines, it doesn't happen too often and each machine eventually gets to send its message. The one that didn't get it's turn will see that the other one is talking and waits.

Because there is no guarantee that a specific machine will *ever* get a turn on the net, this type of mechanism is referred to as a probabilistic access system, since each machine will probably get access to the system someday. Keep in mind that the busier a network is, the greater the chance for collisions and the greater the likelihood that there will be more waiting. This does not mean that more machines mean more collisions. If I am sitting at my machine doing all of my work locally, then the traffic on the network cause by my machine is minimal. However, once I make a connection, the traffic increases.

Ethernet appears in several different forms, depending on it's physical characteristics. Primarily, these fall into the IEEE specification 802.3, with an average speed of 10MHz. One thing I need to point out is that the original specification developed at Xerox is not what most people think about when they think about Ethernet. Rather it is the IEEE 802.3 standard.

The most popular ways Ethernet appears is 10Base5 (Thicknet), 10Base2 (Thinnet) and 10Base-T (Twisted-Pair) and the 100-Mbit equivalents. The general format of these labels is *StypeL*, where *S* is the speed of the cable in megahertz, *type* is the transmission system, in this case baseband versus broadband and the *L* is the maximum length of the cable in 100 meters. I have also heard that the last number indicates the thickness of the cable in tenths of an inch. Thicknet, as one would guess, is thicker than thin net, but both are coax cable. Twisted pair is similar is format to normal phone cable, but may often have eight separate wires.

Often times, the topology (layout) of your network is dependent on what kind of cable you are using. Because it requires a central hub, twisted-pair is usually laid out in a star, with the hub at the center. This is a star topology. Thin- and thickwire are usually be spread out in a line, or

linear topology. This is also called a bus topology.

9.7.2 Token-Ring

Token-ring, developed by IBM, is embodied in the IEEE standard 802.5. The key concept in this type of network is the idea of a token. This is similar to a baton in a relay race when each machine must receive the token before it is allowed to go. If a particular machine has nothing to send, it simply passes the token on to the next machine. If it does have something to send, the message is "linked" with the token before it is send. By seeing the token linked to the message, the other machines know that the token is in use and pass it along to the destination. When the destination machine gets the entire bundle, it puts the token back on the network, with a tag to indicate that it received the packet. It is then passed to the originator as an acknowledgment. The originator then passes the token along to the next machine.

This scheme provides guaranteed access to the network since every machine will eventually get the token. Even if the originator of one packet has more to send, once it gets its acknowledgment back, it must pass the token along. If no others want to send anything, then it can come back to the first machine. However, the others were given the *chance* to send something. This method also provides reliability since the destination machine sends the packet back with an acknowledgment.

9.7.3 ATM

Although not provided in commercial distributions as of this writing, there are drivers for cards supporting the Asynchronous Transfer Mode (ATM). ATM functions like a never ending train. When an empty "car" comes along, the data can jump into it. These cars or *cells* are fairly short (58 octets total, 48 data, 5 header).

The basic idea behind ATM is that mixed bandwidth systems (such as data and voice) can use the same cable without loss of efficiency. ATM is the layer directly above the physical hardware, with the ATM adaptation layer (AAL) serving as the interface to the higher-level protocols. The ATM layer is actually broken up into two sub-layers. The transmission control (TC) is responsible for building the cells and the physical medium (PM) layer is the interface to the physical hardware.

As of this writing, ATM is supported in Linux, but not available in any of the standard distributions. An experimental (read pre-beta) driver is available from *lrcwww.epfl.ch*.

9.7.4 ISDN

For most of the life of electronic/electrical communication, the primary method of communication has been the telephone. As a result, there exists a network of cables and connection throughout the world that dwarfs the Internet in both number of connections and miles of wire. Wouldn’t it be wonderful if we could take advantage of the already existing network? Well, we can. This comes to us in the form of a system called Integrated Services Digital Network, or ISDN.

ISDN is one of the fastest growing technologies, particularly in Europe. Local telephone companies are offering it as a replacement or addition to conventional telephone lines. Until recently, the German phone company was offering cash incentives for businesses and private individuals to switch to ISDN. The primary benefit at least to most end users is that you can simultaneously send data across the same lines as your phone. For example, while you are speaking to a customer, you can be faxing them the spec sheets on your latest product *across the same phone line*. Although such functionality for both partners requires ISDN connections on both ends, your phone could be talking to their conventional phone and your fax could be talking to their conventional fax. However, from your office to the phone company is a single telephone connection.

If both sides are using ISDN, they need to be communicating in a fashion similar to a network like with TCP/IP or IPX/SPX. Therefore, both sides know who is calling. Image getting a call from a customer and having your database automatically call up the record on that customer, even before you pick up the phone! This ability to integrate all these different services from voice to fax to data communication gives ISDN its name.

The key concept in ISDN is the idea of a digital data pipe between the end device and the service provider. Note that I didn't say between the two participants. This allows the service provider the phone company to switch between the ISDN connection on one side to the analog connection on the other. At the receiving end your office will be something similar to a switch box. As the packets come in from the service provider, this switch box will route the packets to the appropriate device. Each device is set with a particular ID number. This works conceptually the same way as SCSI IDs.

As of this writing, three types of connections have been standardized. The first is often referred to as the "basic" rate as it provides the necessary service for basic users such as homes and small businesses. This provides two 64 kbps channels for voice or data and one channel for "out-of-band" signaling. In some cases, you could use these two channels simultaneously and get 128 kbps. However, this would be considered two phone calls.

The "primary" service provides 23 voices or data channels, instead of just two. In Europe, this is increased to 30 channels. The third type provides a 4 KHz analog phone channel along with a 8 or 16 kbps data channel. This allows you to use your old analog phone along side the new ISDN device.

ISDN is not just for data communication. As I mentioned the German phone company subsidized the transfer from the old analog system to ISDN. I have an ISDN connection at home. Using the same wires as my old phone line, the ISDN line comes into a telecommunications box, which converts the signal so that my normal analog phones can work.

ISDN support is provided by isdn4linux, which is a set of kernel modules. The main module isdn communicates with the driver for the particular card. As of this writing, there is a limited number of cards that are supported. However, many cards are supported that don't have "official" drivers for them. For example, the AVM A1 Fritz card is supported using the Teles driver. For more information, check out the /pub/isdn4linux directory on *ftp.franken.de*. This not only has many of the drivers, but also a *very* extensive FAQ.

If you have a 2-0 or later kernel, then you are in luck. ISDN support is included by default. When you run 'make config', you are prompted to incldue/activate it along with several different options. There are only a few cards mentioned by name. However, I know that many other cards will work with the drivers that are included.

9.7.5 Network Hardware

The saying that the chain is only as strong as it's weakest link definitely applies to the network. For a network operating system like Linux, the network hardware can become a deciding factor in terms of how well it performs (or at least how the performance is perceived). It is therefore essential that your network hardware can not only handle the load now, but also as you network grows.

One of the problems I encountered when researching this material is that there is much material available on so many different products. In addition, networking covers such a wide range of products, you could write an entire book just on the networking aspects. In fact, there is a number of good books that do just that.

Since I cannot talk about every aspect, I decided that I would limit my coverage to the network interface card (NIC) which is the first piece of hardware in the long journey between workstation and server. In addition, the most common pieces of hardware on this journey are routers, bridges, hubs and switches (if you have a twisted pair network).

As its name implies a router routes the traffic along the network. However, it more than just deciding what path to take. Instead, modern routers have the ability to determine if the packet should be sent at all. This can be determined by which port as well as which machine is to send or receive the packet. For example, it is common to have a router that only allows connections to a specific machine using only the HTTP or SMTP (email) protocols. Other protocols or even these protocols to other machines are blocked. This is the basic functionality of a firewall.

Typically, routers are a connection between two separate networks. Depending on the router itself, you could have several different networks connected to the same router. In fact, it is possible to have different kinds of physical networks connected to the routers, such as having both serial (to connect to modems, for example), twisted pair and optical.

A hub is often called a repeater, because it serves as a hub the network cables as well as "repeats" the signal, allowing you to transmit over greater distances. A hub is needed when you are using twisted pair cables and every node (client and server) must be connected to a hub. Since a hub sits at the bottom of the protocol stack, it transmits every type of packet.

Typically, hubs are used to organize the nodes on your network into physical groups. However, they do not perform any logical functions, such as determining routes to take (that's what a router does). Despite this, most hubs are capable of doing collision detection.

A modification of a hub is a bridge. Bridges allow you to physically separate network segments and can extend the length of your cables. The difference lies in the fact that the bridge determines if a packet is intended for a machine on the same segment or not. If it is, it can be ignored and not passed through to other segments.

The key lies in what is called a collision domain. In essence, this is the set of nodes that send out packets, which collide with each other. The more collisions you have, the worse your network performance because it means you have more network traffic and other machines need to wait. By grouping machines into groups that communicate with each other, you reduce the collisions with unrelated machines.

Because bridges block the packets for the local collision domain, each domain has fewer collisions. Keep in mind that this only works when there is a lot of traffic between the nodes, such as in a work group. If you have a strict client-server model, a bridge may not bring you much advantage.

Another way of significantly reducing collisions is using a switch. The difference is that the switch analyzes packets to determine the destination and makes a virtual connection between the two ports, thus reducing the number of collisions. Using the store-and-forward method, packets are stored within the switch before being sent along. The cut-through method reads just the header to determine the destination.

An important aspect to look at is obviously the transfer speed of the card. One common problem I have seen in companies without a dedicated IT organization (as in some cases with one) is forgetting the saying about the weakest link. This happens when they buy 10Mbit cards for their workstations (or are perhaps using older models), but install a 100Mbit card in their server. The problem is that the server can only send at 10Mbit, because that's what the clients can handle.

As we discussed previously, the two most common Ethernet types are twisted pair and thin-wire. Traditional Ethernet was limited to only 10Mbit and has been essentially replaced by FastEthernet, which can handle 100Mbits. The problem is that you may not be able to use other existing network components such as cables if you were using thin-wire. The reason is simply that thin-wire is unable to transmit at the higher speed. On the other hand twisted pair can handle it.

One place this is commonly noticed is the connectors on the network cards themselves. You will often find many cards designated 10/100 or something in their name. As you might guess, this indicates they can handle either 10 or 100Mbits, depending on the speed of the hub to which they are connected. I have seen some cards that require you to set the speed either in software or hardware.

However, my 3Com cards detect the speed the hub uses and adjust automatically. In my office at home, I have three computers all hooked through a 10Mbit hub. Since very little data is going through the network, this was sufficient as well as less expensive. Even so, my 3Com cards are all 10/100 and adjust to the slower speed. When I upgrade to a faster HUB, I do not need to replace the cards or do any configuration. I just plug the cables into the new hub and go.

This may sound like a minor point and it is for my three node network. However, at work, with hundreds of nodes it becomes a major issue. Imagine having to change the hardware settings on hundreds of PCs. That means opening the cases, pulling out the card, setting the jumper, putting the card back in, and then closing the case. Granted most newer cards are plug

and play, but are you sure yours is.

Some cards like my 3Com Fast EtherLink XL 3C905B-COMBO have connectors for thin-wire, thick-wire and twisted pair, only the twisted pair connector allows you to use the 100Mb connector. Note also that most of the 3Com Fast EtherLink 10/100 cards, just have the twisted-pair connector.

Keep in mind that even if you do use the twisted pair connector, you are limited by the speed of the other hardware. I chose a 10Mbit hub because I did not want or need to spend the extra money for a 100Mbit hub. Even in a business, you may not need more. If all of your applications are installed locally, with only the data on the server, you probably won't even come close to needing even the 10Mbit. This is especially true if you break down your network into sub-nets, which are separated by routers or you are using switches.

However, speed is not the only consideration, particularly in a server. Take the analogy of a 100 mile race between a Ferrari and a Geo Metro. The winner is fairly obvious, unless you take a Ferrari loaded with bricks and has to refuel every mile. In some cases, you might have a Ferrari network card which is slowed down by other things.

There are several things your card can do, such as what my 3Com 3C980-TX Fast EtherLink Server NIC does. The first is the ability to combine multiple cards into a single virtual interface. One card is processing the packet while the other is receiving, for example. The load is balanced between the cards to ensure that one is not overburdened.

The next feature is what 3Com calls self-healing drivers. Here the card is monitored and action is taken based on what it finds. One simple example would be shutting down one card in a virtual set if it appeared to be causing to many errors.

Throughput (the true measure of speed) is increased by using 3Com's Parallel Tasking. Traditionally, network cards transfer data between the card and memory in one direction at a time. 3Com cards can transmit in both directions. In addition, there was a previous limitation with PCI cards that could transmit a maximum of 64 bytes at once. The newest 3Com cards have increased this to 1514, the maximum for a standard Ethernet packet. This meant that with previous cards, it might need up to 24 bus cycles to transmit the data, the 3Com card can do it in a single cycle.

A moment ago, I mentioned cases where people would install 100Mbit cards in their server and 10Mbit cards in their clients. In those cases, they actually had 10 Mbit hubs, so the problem was as much an issue with the hub as with the speed of the client cards. In some cases, it actually makes sense to configure your system like that, but you need a hub that can handle the job.

One solution to the problem is the 3Com SuperStack II Dual Speed Hub. The key is part of the name: "dual speed.". As its name implies it can actually handle both 10Mbit and 100Mbit connections. It is able to sense the speed on the port and adjust itself for that port. This means that the connection between the server could be running at 100Mbit, with the connection between the hub and clients running at 10 Mbit (or maybe just some of the clients).

This ends up increasing overall performance since the hub can operate in duplex mode. That is, it can send and receive at the same time. 10 Mbit data is being sent to the hub as it is sending 100Mbit data to the server.

Some vendors try to save a little by making hubs that "pretend" to run at both 10 and 100Mbits. This is done by having a single port that can handle the 100Mbits, which is typically connected to the servers. However, this means that if you ever upgrade a single client, you have to upgrade the hub as well. The 3Com solutions automatically make the change for you.

One thing to keep in mind here is the cabling. FastEthernet requires what is referred to as category 5 cabling. However, 10Mbit can be handled by category 3 or 4. Although you can certainly connect your network using category 3 cable, the number of errors increases dramatically. Packets need to get resend and it can actually turn out to be slower than running at 10Mbit. The 3Com SuperStack addresses this issue by monitoring the frequency and type of errors. Should the errors be too high, it will automatically lower the speed to 10Mbit.

In principle, routers have the same limitations as hubs, in that they can limit, well as are limited by, the other network components. However there are several features that we ought to take a look at.

One feature provided by 3Com’s NETBuilder routers is what is referred to as bandwidth grooming. Among other things, this allows you to prioritize the traffic on your network, based on a number of different criteria. For example, you define higher priority to specific protocols or specific ports (or both). This is useful when defining priority based on a specific application, type of connection and many other cases.

In addition, the NETBuilder series features dual processors. While one processor is handling tradition routing functions such processing the packets, the second processor concerns itself with the "grooming" functions, which greatly increases the overall performance.

There is also the issue of security. Many people think of router security only in terms of connections to the Internet. However, some companies are concerned with internal security as well. For example, it is possible with the NETBuilder routers to disallow connections from the warehouse to the main server, except for specifically defined ports. This might give them access to the main database application, but prevent them from poking around the file system.

One thing to keep in mind is that there are a number of differences between the behavior of a Wide Area Network (WAN) and a Local Area Network (LAN). In my opinion, the two most significant aspects are the fact that a WAN has slower speeds and the routing of the packets is the dominant behavior as compared to fast speeds and switching for the LAN. Even if your internal network only runs at 10Mbps, it is still 160 times faster than a typical 64Kbps WAN connection.

The result of all of this is that you typically have different kinds of equipment for both. In addition, because of the slower speeds, a WAN has less bandwidth and your are "encouraged" to reduce unnecessary traffic. This is where routing comes in. You want to limit unnecessary and even unwanted traffic. For example, we talked above about the ability of 3Com routers to direct traffic based on specific ports. In some cases, you may want to turn off specific ports to

certain network segments to reduce the traffic, although other ports (and therefore other protocols) are allowed. One common thing is to restrict broadcast traffic, which the 3Com routers can do.

Another thing we discussed was the ability of the 3Com routers to prioritize the packets. In most cases, applications always use the same range of ports to access other machines. For example, an Oracle database is usually accessed using port 1521. To ensure proper response times, port 1521 could be given priority over something like file data transfer. Files going across the WAN can be typically given lower priority than the database application. The 3Com router thus allows you to manage the performance on each network segment.

A off-shoot of this is "protocol reservation." As its name implies, a certain portion of the bandwidth is reserved for specific protocols. That means that no matter what other traffic is on the link, the reserved portion will always be available for that protocol.

Another thing to consider is how the routing information is transferred between routers. Many routers use what is called "distance vector routing" where the router can determine the shortest path between two nodes. However, you may not want the router to choose the shortest path, since "short" means the number of nodes it goes through (or hops) and not the length of the cable or the speed. Often such routers will exchange information even though the network has not changed. In essence, this wastes bandwidth.

Instead, to limit bandwidth you want all packets going to a particular subnet to always use a pre-defined route. This is a capability of "link state" routing. Although this requires more computational power than distance vector routing, it also requires a lot less bandwidth. Since routes are calculated, less data is transferred, so when a link goes down, the updated information reaches the effected routers more quickly and the new route in effect more quickly as the computation is faster than thenetwork.

Another core aspect of the vendor your choose is the after sales service. For most companies, the primary concern is the warranty. That is, what happens when a card malfunctions. Most warranties last a year, which is normally long enough to identify any manufacturing defects. However, even within the warranty period, you will generally find that you will either have to return the card to the reseller or return it directly to the manufacturer. Therefore, it is a good idea to have enough spares on hard. Although you might be able to work out an arrangement with either the vendor or reseller to send you a replacement before they receive the defective card, you are still out of work for a couple days, so spares are still a good idea.

Thin Wire versus Twisted Pair

The fact that twisted pair cabling is less expensive than thin wire is deceiving. For a given length of cable, the cable itself and the connectors are cheaper. However, you must keep in mind that there will be a cable from the hub to each node, including the server. In contrast, thin wire cables are laid between the nodes, forming a "loop".

Let’s take an example with a server and four computers, spaced evenly every ten feet. You could get away with just forty feet of thin wire cable, as you need ten feet from the server to the first machine, another ten feet from the first to the second, and so on.

With twisted pair, let's assume that the hub is right next to the server, so the cable length can be ignored. You need ten feet of cable to the first computer, but twenty feet to the second, thirty feet to the third, and forty feet to the fourth. This means a total of 100 feet. The more computers you have the greater the difference in cable lengths.

In addition, there is more work. You cannot just move from computer to computer, adding cable as you go. You lay the cable from the hub to the first computer, then go back to the hub. You lay the cable from the hub to the second computer, then go back to the hub, and so forth.

One the other hand, twisted pair is a lot safer. As I mentioned, if the connection to one computer goes down, the rest can still work.

Well enough for the theory. Reality today is a lot different than it was when both of these technologies where fairly young. Today, most installations have switched to twisted pair and every new installation I know does so as well. For the system administrator or network technician any perceived disadvantage of twisted pair is easily countered by the advantages.

The problems that thin-wire cabling has, such the "messy" physical connections at the back, the "loop" nature of the cabling, plus the slower speeds make thin-wire far less attractive than five years ago. Because of the loop, a problem anywhere means problems for everyone. This goes beyond finding connection breaks. For example, if one of the NICs has a problem and is causing interference on the network, all nodes are affected. Added to this is the fact that it is often difficult to determine which NIC is causing the problem. Each node needs to be examined individually, which means much higher costs.

On the other hand with twisted pair, the cabling is easier to manage, problems are easier to troubleshoot and the system is generally easier to manage. As an example, take the company where I currently work. We did some renovations on an existing building, but insisted on double floor. Within the floor we laid cabling for both the telephone and the LAN. At several places in each office we installed a "well", with receptacles for the telephone and LAN. Each was then connected to a central location, which then provided the connection to other areas of the company. For our LAN, each node is connected to a 100 Mbit switch which is then connected via optical fiber to other parts of the building and even to another office across town.

For the network technicians, this means all they need to do is plug one end of the cable into the back of the computer and the other into a plug in the nearest floor well. Therefore, they don't need to worry about ensuring the loop is complete. Just plug and go.

As I mentioned, all of the cables lead to a central location, which is initially just a patch panel. From here the connection is made to the switch. Since it is the physical connection from the patch panel to a switch that determines which segment a computer is on, we can easily patch computers from one segment to another without re-wiring. This allows you to have completely seperate networks within the same room. For example, my work station needs access to customer machines, so I am on one segment. The test machines do not need that access so they are on a different segment. However, they not only can be in the same room, but can be plugged into connections in the same floor well.

Another nice thing about this is that the physical connections for both the phone and LAN are the same. Although physically seperate within the patch cabinet, the wires are the same, the patch cables are the same and so forth. Since the LAN and phone patch panels are physically seperate within the cabinet, it is much each for our network technicians.

Because of the explosion in computer use during the past few years, you will be able to find many motherboards with a NIC built-in. Needless to say, this will be twisted-pair and not thin-wire. These NICs, as well as new ones that you can buy seperately typically do autoswitching duplex 10/100 Mbit.

Chapter 10 System Monitoring

Monitoring your system means more than just watching the amount of free hard disk space or the number of users running a certain application. Many aspects of your system are static over fairly long periods of time, such as the layout of your hard disk. However, such information is as important to really knowing your system as how much free memory there is at any given time.

Linux provides a wide range of tools to not only monitor your system as it runs but to find out how it's configured. In this chapter, I am going to talk about the tools you need and the files you can look in to find out anything about your system that you need. In the next chapter, we will take some of these tools and see how you can use them to help you achieve the goal of administering your system.

10.1 Finding Out About Your System

One challenging aspect of tech support is that before you talk to the customer, you have no way of knowing what the problem will be. It can be anything from simple questions that are easily answered by reading the manual to long, drawn-out system crashes.

Late one Monday afternoon, when I had been idle the longest, my phone rang as the next customer came into the queue. The customer described the situation as simply that his computer would no longer boot. For some reason, the system rebooted itself and now it would not boot.

When I asked the customer how far the system got and what, if any, error messages were on the screen, his response indicated that the root file system was trashed. At that point, I knew it was going to be a five-minute call. In almost every case, there is no way to recover from this. On a few rare occasions, fsck can clean things up to be able to boot. Because the customer had already tried that, this was not one of those occasions.

We began discussing the options, which were very limited. He could reinstall the operating system and then the data, or he could send his hard disk to a data recovery service. Because this was a county government office, they had the work of dozens of people on the machine. They had backups from the night before, but all of that days work would be lost.

Because no one else was waiting in the queue to talk to me, I decided to poke around a little longer. Maybe the messages we saw might indicate to us a way to recover. We booted from the emergency boot/root set again and started to look around. The fdisk utility reported the partition table as valid but it looked as though just the root file system was trashed, which is bad enough.

I was about ready to give up when the customer mentioned that the fdisk table didn't look right. Three entries in the table had starting and ending blocks. This didn't sound right because he only had two partitions: root and swap. So I checked **`/etc/fstab`** and discovered that another file system was being mounted.

Because the data was probably already trashed, there was no harm in continuing, so we decided to try running **fsck** on it. Amazingly enough, fsck ran though relatively quickly and reported just a few errors. We mounted the file system and, holding our breath, we did a listing of the directory. Lo and behold, *there* was his data. All the files appeared to be intact. Because this was all in a directory named **/data**, he simply assumed that there was no **/usr** or **/home** file system, which there wasn't. However, there was a *second* file system.

I suggested backing up the data just to be safe. Because it was an additional file system, however, reinstalling the OS could preserve it. Within a couple of hours, he could be up and running again. The lesson learned: Make sure you know the configuration of your system! If at all possible, keep data away from the root file system and do a backup as often as you can afford to. The lesson for me was to have the customer read each entry one-by-one.

Being able to manage and administer your system requires that you know something about how your system is defined and configured. What values have been established for various parameters? What is the base address of your SCSI host adapters? What is the maximum UID that you can have on a system? All of these are questions that will eventually crop up, if they haven't already.

The nice thing is that the system can answer these questions for you, if you know what to ask and where to ask it. In this section, we are going to take a look at where the system keeps much of its important configuration information and what you can use to get at it.

As a user, much of the information that you can get will be useful only to satisfy your curiosity. Most files that I am going to talk about you can normally read. However, you won't be able to run a few of the utilities, such as fdisk. Therefore, what they have to say will be hidden from you.

If you are an administrator, there are probably many nooks and crannies of the system into which you never looked, many you probably never knew existed. After reading this section, I hope you will gain some new insight into where information is stored. For the more advanced system administrator, this may only serve as a refresher. Who knows? Maybe the gurus out there will learn a thing or two. Table 0-1 gives you a good overview of the various files configuration files on your system.

The command **getconf** will display the maximum allow value of various system configuration parameters. It is useful if you run into problems and you think you might have reached some predefined limit. For example, you might try to create a filename or username which is to long. The **getconf** command will show you what the maximum is. For example:

```
getconf LOGNAME_MAX
```

will show you the maximum length of the user name. This can also show you to things that are less meaningful to users, such as the size of a page in memory.

A list of the essential system files can be found here.

10.1.1 Hardware and the Kernel

The part of the system that probably causes the most problems and results in the greatest number of calls to Linux hotlines is hardware, For those of you who have watched the system boot, you may already be familiar with what I call the hardware or boot screen (which actually consists of several screens). This gives you a good overview to what kind of hardware you have on your system and how it is configured. Because many hardware problems are the result of misconfigured hardware, knowing what the system thinks about your hardware configuration is very useful.

Fortunately, we don't need to boot every time we want access to this information. As the system boots (as well as at other times), the kernel writes to system log files. In the case of the messages you see at boot, this file might be **`/var/log/messages`**, **`/usr/adm/messages`**, or **`/usr/var/messages`**. You can also get a dump of these messages using the **`mesg`** command.

Supplementary to this are the files under **`/proc`**. If you look at your mount table (using the mount command), you will see that this is a mounted file system. However, you will also note that no physical device is associated with it. However, it behaves just like a normal file system, with a number of directories and files, even though none of the files actually exist. When someone accesses this "file system," the appropriate VFS structures are filled and the appropriate information is delivered.

You find two basic types of information via the files in /proc. First, you can get a lot of kernel information by reading the files in this directory. On many systems, the only way to access this information is to read the device /dev/kmem (the kernel memory device) directly. Here, all you need to do is run

```
cat file_name
```

to be able to read the various information.

In addition to these files, there is one subdirectory for each process on the system. The name of the directory is the process ID of the respective process.

- cpuinfo: Various information about the CPU, such as the manufacturer, model (SX/DX), integrated NPU, etc.
- devices: A list of block and character devices configured into the kernel, along with their major number.
- dma: Devices and their dma channels.
- filesystem: File system drivers in the kernel.
- interrupts: A list of interrupts and the total number of interrupts on that interrupt vector.
- kcore: The physical memory of the system. Same format as a core file. The size of this file is the same as physical memory, plus a 4Kb header to make it look like a core file.
- kmsg: Often used instead of the syslog() system call to log kernel messages. To access this file, a process must have root privileges. If the syslog process is running, this file should *not* be read.
- ksysms: A list of kernel symbols.

- loadavg: Load average. This is the same information produced by uptime.
- meminfo: Memory information in bytes including free memory, user memory, and swap.
- module: An ASCII list of kernel modules.
- net: Subdirectory containing information about the network layer.
- self: The process currently accessing the /proc FS and is linked to directory with the applicable PID.
- stat: Various statistics about the system, including time CPU spent in user mode, pages brought in and out from the disk, number of swap pages brought in and out, number of interrupts since boot, number of context switches and the boot time.
- uptime: Amount of time system has been up and time spent in the idle process.
- version: Version information.

In each process subdirectory are several files and one directory.

- cmdline: Complete command line for the process.
- cwd: Current working directory of the process.
- environ: Environment for the process.
- exe: A link to the executable that started that process.
- fd: A subdirectory that contains one entry for each file that the process has opened. Each entry i s the file descriptor that the process is using.
- maps: Memory mappings the process uses.
- mem: Memory of the process.
- root: Pointer to the root directory of the process.
- stat: Status information about the process. This file is mostly just numbers (with the exception of the executables name and the status). In general, this is the same information you would get from running ps, with various options:
- pid: The process ID.
- comm: The commands (executables) filename.
- state: Process state, such as R, running; S, sleeping in an interruptible wait; D, sleeping in an uninterruptible wait or swapping; Z, zombie; and T, traced or stopped (on a signal).
- ppid: PID of the parent.
- pgrp: Process group ID of the process.
- session: Session ID of the process.
- tty: The tty the process uses.
- tpgid: Process group ID of the process that currently owns the tty to which the process is connected.
- flags: Flags of the process.
- minflt: Number of minor faults (i.e., those that have not required loading a memory page from disk).
- cminflt: Number of minor faults that the process and it's children have made.
- majflt: Number of major faults (i.e., those that require loading a memory page from disk).
- cmajflt: Number of major faults that the process and it's children have made.
- utime: Number of jiffies (10 milliseconds) that this process has been scheduled in user mode.
- stime: Number of jiffies that this process has been scheduled in kernel mode.

- cutime: Number of jiffies that this process and it's children have been scheduled in user mode.
- cstime: Number of jiffies that this process and it's children have been scheduled in kernel mode.
- counter: Current maximum size in jiffies of the process's next time slice, of what is currently left of its current time slice, if it is the currently running process.
- priority: Standard nice value, plus 15. The value is never negative in the kernel.
- timeout: Time in jiffies of the process's next timeout.
- itrealvalue: Time in jiffies before the next SIGALRM is sent to the process due to an interval timer.
- starttime: Time the process started in jiffies after system boot.
- vsize: Virtual memory size.
- rss: (Resident set size) Number of pages the process has in physical memory, minus three for administrative purposes.
- rlim: Current limit in bytes on the RSS of the process (usually 2,147,483,647).
- startcode: Address above which program text can run.
- endcode: Address below which program text can run.
- startstack: Address of the start of the stack.
- kstkesp: Current value of esp (32-bit stack pointer), as found in the kernel stack page for the process.
- kstkeip: Current EIP (32-bit instruction pointer).
- signal: Bitmap of pending signals.
- blocked: Bitmap of blocked signals (usually 0; 2 for shells).
- sigignore: Bitmap of ignored signals.
- sigcatch: Bitmap of catched signals.
- wchan: The wait channel of the process.
- statm: Additional status information that takes longer to gather and is used less often.
- size: Total number of pages that the process has *mapped* in virtual memory (whether or not they are in physical memory).
- resident: Total number of pages the process has in memory; should equal the RSS field from the stat file.
- shared: Total number of pages that this process shares with at least one other process.
- trs: (Text Resident Size) Total number of text pages the process has in physical memory. (Does not include any shared library.)
- lrs: (Library Resident Size) Total number of library pages the process has in physical memory.
- drs: (Data Resident Size) Total number of data pages the process has in memory.
- dt: Number of library pages that have been accessed.

Access to this information is useful when you run into problems on your system. System slowdowns, for example, are difficult to track down. Using ps, you can get a feel for what is on your system. Sometimes, **`ps`** will show you what process is running but not what it is doing. With a little practice, the files and directories in **`/proc`** can.

10.1.2 Terminals

Each line in **/etc/inittab** that refers to a terminal device points to an entry in the **/etc/gettydefs** file. The entry for the COM1 port (/dev/ttyS1) might look like this:

```
S1:234:respawn:/etc/getty  ttyS1  M19200
```

From our discussion of the **/etc/inittab** file in the section on starting and stopping the system, you see that this entry starts the **/etc/getty** command. Two arguments are passed to getty: the terminal on which it should run (**ttyS1**) and the gettydefs entry that should be used (m). The **/etc/gettydefs** file defines such characteristics as the default speed, parity, and the number of data bits. For example, the m entry, to which the previous inittab entry points, might look like this:

```
M19200 # B19200 CS8 CLOCAL # B19200 SANE -ISTRIP CLOCAL #@S login: # M19200
```

The fields are

```
label # initial_flags # final_flags #login_prompt # next_label
```

The label entry is what is being pointed to in the inittab file. The initial_ flags are the default serial line characteristics that are set, unless a terminal type is passed to getty. Normally, the only characteristic that needs to be passed is the speed. However, we also set HUPCL (hang up on last close).

The final_flags are set just before getty executes login. Here again, set the speed and HUPCL. However, we also set the terminal to SANE, which is actually several characteristics. (Look at the **gettydefs** man-page for more details.) We also set TAB3, which turns tabs into space; ECHOE, which echoes the erase character as a backspace-space-backspace combination; and IXANY, which enables any character to restart output if it is stopped by the XOFF character.

10.1.3 Hard Disks and File Systems

A common problem that has caused long calls to support is the layout of the hard disk. Many administrators are not even aware of the number of partitions and file systems they have. This is not always their fault, though, because they often inherit the system without any information on how it's configured.

The first aspect is the geometry, which includes such information as the cylinders, heads, and sectors per track. In most cases, the geometry of the hard disk is reported to you on the hardware screen when the system boots. You can also get this information from **fdisk**.

To find how your hard disk (or hard disks) is laid out, there are several useful programs. The first is **fdisk**, which is normally used to partition the disk. Using the -l option, you can get fdisk to print out just the partition table. On my system, I get output like this:

```
Disk /dev/sda: 255 heads, 63 sectors, 1106 cylinders
Units = cylinders of 16065 * 512 bytes

   Device Boot    Start       End    Blocks   Id  System
```

```
/dev/sda1             1       653   5245191   83  Linux
/dev/sda2           654      1106   3638722+  83  Linux

Disk /dev/hda: 255 heads, 63 sectors, 826 cylinders
Units = cylinders of 16065 * 512 bytes

   Device Boot    Start       End    Blocks   Id  System
/dev/hda1             1       192   1542208+   c  Win95 FAT32 (LBA)
/dev/hda2   *       193       199     56227+  83  Linux
/dev/hda3           200       250    409657+  82  Linux swap
/dev/hda4           251       826   4626720   83  Linux

Disk /dev/hdc: 16 heads, 63 sectors, 39770 cylinders
Units = cylinders of 1008 * 512 bytes

   Device Boot    Start       End    Blocks   Id  System
/dev/hdc1             1      8739   4404424+  83  Linux
/dev/hdc2          8740     17478   4404456   83  Linux
/dev/hdc3         17479     27881   5243112   83  Linux
/dev/hdc4         27882     39770   5992056   83  Linux
>
```

One my system I currently have three hard disks. The first one listed is a SCSI disks (/dev/sda) although this is not the first one booted. The second and third are both EIDE drives (/dev/hda, /dev/hdc). The first EIDE is what I boot from, and as you can see fdisk says there is a Windows 95 partition (although Windows 98 is actually installed). Followed by a small Linux partition (50 MB), the Linux swap space and another Linux partition. All of the other partitions on all drives are Linux.

If you look carefully and compare the ending blocks with the starting blocks of the next physical partition, you see that, in this case, there are no gaps. Small gaps (just a few tracks) are nothing to have a heart attack over because you are only loosing a couple of kilobytes. However, larger gaps indicate that the whole hard disk was not partitioned, and you may be loosing some useful space.

If you have multiple hard disks on your system, your messages file (i.e., **/var/log/messages**) may show you this. Every version of Linux I have seen will report all the SCSI devices it sees, and because hard disks are all standard devices, they should all be reported.

If you have multiple hard disks, you can specify the devices as an argument to fdisk. For example, to print the partition table for the second SCSI hard disk, the commands would be

fdisk -l /dev/sdb

Unlike other dialects of UNIX, Linux cannot have multiple file systems in each partition. Therefore, you cannot have more file systems than you have partitions. (I'll ignore NFS, etc., for the time being.) However, here we are talking about both primary and logical partitions. In my case, all of my partitions are primary, which limits me to only four partitions per drive. However, if I created an extended partition, I could have many more logical partitions(I have

created 8 logical partitions for testing and have been told you can have more). Theoretically, each partition (whether primary or logical) can have a different Linux distribution version.

To find out what file systems are on your disks, first use the mount command. However, this command only tells you which file systems are currently mounted. Using this command on a running system is useful to determine whether a directory is part of one filesystem or another. Although the df command (more on that later) will tell you which file systems are mounted, it doesn't tell you what options were used, such as whether the file system is read-only. On a few occasions, I have had customers call in reporting file systems problems because they could write to them, but found out they were mounted as read-only.

What if you suspect that there are more file systems than are mounted? The first thing to do is check fdisk for all the hard disks on your system. If you have only one hard disk and it only has one partition, then only one file system can be mounted.

Maybe the file systems exist but aren't mounted. To check, first run the mount command to see what is currently mounted. Then check the **/etc/fstab** file to see what file systems are known and what the options are. A noauto in the options column means that file system should not be mounted automatically when the system boots. Therefore, it's possible that a filesystem was created on a partition, but it is not mounted automatically when the system boots.

10.1.4 User Files

The **/etc** directory contains the all-important passwd file, which gives important information about which users are configured on the system, what their user ID numbers are, what their default groups are, where their home directories are, and even what shells they use by default.

The default group is actually a group ID number rather than a name. However, you can easily match the group ID to the group name by looking at **/etc/group**. This also gives you a list of users broken down into the groups to which they belong. Note that "groups" is plural because a user can belong to more than one group.

A list of the essential system files can be found here.

10.1.5 Network Files

If you are running TCP/IP, you can look in a couple of places for information about your system. First, check out the file **/etc/resolv.conf**. If you don't find it and you know you are running TCP/IP, don't worry! The fact that it is missing tells you that you are not running a nameserver in your network. If it is not there, you can find a list of machines that your machine knows about and can contact by name in **/etc/hosts**. If you are running a nameserver, this information is kept on the nameserver itself.

The content of the **/etc/hosts** file is the IP address of a system followed by its fully qualified name, and then any aliases you might want to use. A common alias simply uses the node name and omits the domain name. Each line in the **/etc/resolv.conf** file contains one of a couple different types of entries. The two most common entries are the domain entry,

which is set to the local domain name, and the nameserver, which is followed by the IP address of the name "resolver." See the section on **TCP**/IP for more information on both of these files.

It's possible that your machine is the nameserver itself. To find this out, look at the file **/etc/named.conf**. If this exists, it is probably a nameserver. The /etc/named.conf file will tell you the directory where the name server database information is kept. For information about the meaning of these entries, check out the **named** man-page, as well as the section on TCP/IP.

Another place to look is the start-up scripts in **/etc/rc.d**. Often, static routes are added there. One likely place is **/etc/rc.d/init.d/network**. If these static routes use tokens from either **/etc/networks** or **/etc/gateways** that are incorrect, then the routes will be incorrect. By using the -f option to the route command, you can flush all of the entries and start over.

Although not as often corrupted or otherwise goofed up, a couple other files require a quick peek. If you think back to our telephone switchboard analogy for TCP, you can think of the /etc/services file as the phone book that the operator uses to match names to phone numbers. Rather than names and phone numbers, **/etc/services** matches the service requested to the appropriate port. To determine the characteristics of the connection, inetd uses **/etc/inetd.conf**, which contains such information as whether to wait for the first process to finish before allowing new connections.

Other common places for confusion are incorrect entries and the inevitable calls to support deals with user equivalence. As I talked about in the section on TCP/IP, when user equivalence is set up between machines, many remote commands can be executed without the user having to produce a password. One more common misconception is the universality of the **/etc/hosts.equiv** file. Though this file determines what user equivalence should be established with what other machine, it does not apply to one user: root. To me, this is rightly so. Though it does annoy administrators who are not aware of this, it is nothing compared to the problems that might occur if it did apply to root, and this is not what you would expect.

To allow root access, you need to create a **.rhosts** file in roots home directory (usually /) that contains the same information as **/etc/hosts.equiv** but instead applies to the root account. The most common mistake made with this file is the permission. If the permission is such that any user other than root (as the owner of the file) can read it, the user-equivalent mechanism will fail. See /etc/hosts.equiv and **$HOME/.rhosts** to see which remote users have access to which user accounts.

A list of the essential system files can be found here.

10.1.6 Important System Files

File	Purpose	Where to Find More Information
User and Security Files		

/etc/group	User group information	**group**, **chmod**
/etc/npasswd	npasswd configuration file	**npasswd**
/etc/shadow	shadow password file	**password**, **npasswd**
/etc/passwd	User account information	**password**, **chmod**
Networking Files		
/etc/bootptab	Internet Bootstrap Protocol server database	**bootptab**
/etc/exports	Directories to export to NFS clients	**exports**
/etc/gateways	List of gateways	**routed**
/etc/hosts	Hostname to IP address mapping file	**route**
/etc/hosts.equiv	Lists of trusted hosts and remote users	**hosts.equiv**
/etc/inetd.conf	inetd configuration file	**inetd**
/etc/named.conf	named default initialization file	**named**
/etc/networks	Known networks	**route**
/usr/lib/named or **/etc/named.d**	**named** configuration files	named
/etc/smb.conf or /etc/samba/smb.conf	SAMBA configuration file	**smb.conf**
/etc/snmpd.conf	SNMP daemon configuration file	**snmpd.conf**
/etc/ftpaccess	FTP configuration file	**ftpaccess**
/etc/httpd/access.conf	HTTP access configuration file	
/etc/httpd/httpd.conf	HTTP daemon configuration file	
/etc/httpd/srm.conf	HTTP server resource management configuration file	
/etc/services	Network services list	services(5)
X-Windows Files		
/etc/XF86Config or **/etc/X11/XF86Config**	X-Server configuration file	**XF86Config**, **xf86config**
/etc/X11/xinit/xinitrc	xinit configuration file	**xinit**
$HOME/.xinitrc	User-specific xinit configuration file	**xinit**
$HOME/.fvwmrc	fvwm configuration file	**fvwm**, **X**

/usr/lib/X11/system.fvwmrc	System default MWM configuration file	**fvwm**, **X**
/usr/lib/X11/app-defaults	Application-specific defaults	**X**
$HOME/.Xdefaults-hostname	Host-specific defaults	**X**
System Start-Up Files		
/etc/inittab	init configuration file	**inittab**
/etc/lilo.conf	Lilo configuration file	**lilo.conf**, **lilo**
/etc/rc*	System start-up scripts	**init**, **initscript**
System Log Files		
/etc/syslog.conf	System login configuration file	**syslog.conf**
/var/log/message	General system log file	**syslogd**
Miscellaneous Files		
/etc/profile /etc/bashrc /etc/cshrc	Systemwide shell configuration files	man-page for respective shell
$HOME/.bashrc $HOME/.chsrc $HOME/.kshrc	User-specifc shell configuration files	man-page for respective shell
/etc/sysconfig	Miscellaneous configuration files	

10.2 What the System Is Doing Now

At any given moment, there could be dozens, if not hundreds, of different things happening on your system. Each requires systems resources, which may not be sufficient for everything to have an equal share. As a result, resources must be shared. As different processes interact and go about their business, what resource a process has and the amount of that resource that it is allocated will vary. As a result, performance of different processes will vary as well. Sometimes the overall performance reaches a point that becomes unsatisfactory. The big question is, what is happening?

Users might be experiencing slow response times and tell you to buy a faster CPU. I have seen many instances in which this was the case, and afterward, the poor administrator is once again under pressure because the situation hasn't changed. Users still have slow response times. Sometimes the users tell the administrator to increase the speed on their terminal. Obviously 9600 isn't fast enough when they are doing large queries in the database, so a faster terminal will speed up the query, right?

Unfortunately, things are not that simple. Perhaps you, as the system administrator, understand that increasing the baud rate on the terminal or the CPU speed won't do much to speed up large database queries, but you have a hard time convincing users of this. On the other hard, you might be like many administrators who, because you were "unlucky" enough

to have worked with a computer before, was thrown into the position, as often is the case. What many of us take as "common knowledge," you may have never experienced before.

The simplest solution is to hire a consultant who is familiar with your situation (hardware, software, usage, etc.) to evaluate your system and make changes. However, computer consultants can be like lawyers. They may charge enormous fees, talk in unfamiliar terms, and sometimes in the end, you still haven't gained anything.

Not all computer consultants or lawyers are like that. It's simply a matter of not understanding what they tell you. If you do not require that they speak in terms that you understand, you can end up getting taken to the cleaners.

If you feel you need a consultant, do two things. As with any other product, you must shop around. Keep in mind that the best consultant to get is not necessary the cheapest, just as the best one is not necessarily the most expensive. The second key aspect is to know enough about your system, at least, conceptually to understand what the consultant is saying.

In this section, I am going to combine many of the topics and issues I discussed previously to find out exactly what your system is doing at this moment. By knowing what the system is doing, you are in a better position to judge if it is doing what you expect it to do, plus you can make decisions about what could and/or should be changed. This knowledge also has a side benefit of helping you if you should need to call a consultant.

So, where do we start? Well, rather than defining a particular scenario and saying what you should do if it happened, lets talk about the programs and utilities in terms of what they tell you. Therefore, I am going start with general user activity and proceed to the specifics.

10.2.1 Users

Its often useful to know just how many users are logged onto your system. As I mentioned earlier, each process requires resources to run. The more users who are logged on to your system, the more processes there are using your resources. In many cases, just seeing how many users are logged in rings bells and turns on lights in your head to say that something is not right.

The easy way to figure out how many users are logged in is with the **who** command. Without any options, who simply gives you a list of which users are logged in, the terminal into which each user is logged, and the time each user logged in. If you use the -q option (for "quick"), you get just a list of who is logged on and the user count. For example

root root root jimmo # users=4

For every user logged in, there is at least one process. If the user first gets to a shell and starts its application that way, it probably has two processes: the login shell and that application. Therefore, the minimum is the number of users times two (assuming the application isn't the login shell). Granted, the shell is sleeping, waiting for the application to finish, though it is still taking up system resources. Plus, dozens of system process are running, also taking up system resources.

Although I rarely use who with any option except -q, it does have several other options that I have used on occasion. One is the -w option, which tells you whether the user has the mesg flag set to yes or no. A + (plus sign) means you can contact him or her with write, for example, and a - (minus sign) means you cant. Another option, the -i option, tells you the users idle time.

10.2.2 Processes

The **ps** command gives you a process status. Without any options, it gives you the process status for the terminal on which you are running the command. That is, if you are logged in several times, ps will only show you the processes on that terminal and none of the others. For example, I have four sessions logged in on the system console. When I switch to one and run ps, I get

```
  PID TTY          TIME CMD
 1762 pts/3    00:00:00 bash
 1820 pts/4    00:00:00 bash
 1858 pts/5    00:00:00 bash
23172 pts/4    00:00:00 ps
>
```

This shows those processes running as the same user which started the ps command.

If I want to see the processes on a particular terminal, you can use the -t option. A nice aspect of the -t option is that you don't have to specify the full device name or even the "tty" portion. You can just give the tty number. For example:

```
ps –t pts/4
```

which shows:

```
  PID TTY          TIME CMD
 1799 pts/4    00:00:00 bash
 1805 pts/4    00:00:00 su
 1806 pts/4    00:00:00 bash
 1819 pts/4    00:00:00 su
 1820 pts/4    00:00:00 bash
>
```

Note that there are a number of other processes than in the previous example. The reason is that the first example, shows us all of the processes for the current user and the second example shows us all of the processes running on that terminal. The reason that there are more processes on that terminal than the user has is that there are a couple of su command which switch users. Therefore the processes are running as a different user.

Keep in mind that if I was on a pseudo-terminal, the terminal number also includes the pts/ and not just the number. The pts shows us that we are running on a "pseudo" terminal that does not physically exist.. If I have a console or serial terminal, then the pts isn't used because it isn't part of the tty name. For example, if I wanted to check processes on tty4, I would enter the following:

ps -t4

Note also that on older system you did *not* specify the **/dev** portion of the device name, even if you specify the tty portion. For example, this works

ps -t pts/0

but this doesn't work:

ps -t /dev/pts/0

Also be careful with spaces after the -t. Some versions require that there be *no* spaces. New versions of ps accept, both the older style single-letter option as well as the longer, GNU-style options. Typically I use ps aux to show me all of the processes on the system and grep for a specific process. Assuming I know which one I am looking for.

If you are curious about what a particular user is running, use the -u option to find out every process that user owns.

Although running **ps** like this shows who is running what, it tells you little about the behavior of the process itself. In the section on processes, I showed you the -l option, which shows you much more information. If I add the -l long option, I might get output that looks like this:

```
  F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
100 S     0  1258     1  0  69   0 -   324 read_c tty1     00:00:00 mingetty
100 S     0  1259     1  0  69   0 -   523 wait4  tty2     00:00:00 login
100 S     0  1260     1  0  69   0 -   324 read_c tty3     00:00:00 mingetty
100 S     0  1261     1  0  69   0 -   324 read_c tty4     00:00:00 mingetty
100 S     0  1262     1  0  69   0 -   324 read_c tty5     00:00:00 mingetty
100 S     0  1268     1  0  69   0 -   324 read_c tty6     00:00:00 mingetty
100 S     0  1730  1682  0  69   0 -   550 wait4  pts/3    00:00:00 su
000 S     0  1731  1730  0  69   0 -   684 wait4  pts/3    00:00:00 bash
100 S     0  1805  1799  0  69   0 -   550 wait4  pts/4    00:00:00 su
000 S     0  1806  1805  0  69   0 -   682 wait4  pts/4    00:00:00 bash
100 S     0  1843  1838  0  69   0 -   550 wait4  pts/5    00:00:00 su
000 S     0  1844  1843  0  69   0 -   682 wait4  pts/5    00:00:00 bash
100 S     0  2100  1858  0  69   0 -   550 wait4  pts/5    00:00:00 su
000 S     0  2101  2100  0  69   0 -   684 read_c pts/5    00:00:00 bash
100 S     0  6779  3357  0  69   0 -   550 wait4  pts/7    00:00:00 su
000 S     0  6780  6779  0  72   0 -   694 wait4  pts/7    00:00:00 bash
100 R     0 23217  6780  0  78   0 -   797 -      pts/7    00:00:00 ps
>
```

When problems arise, I quite often want to see which process is running the longest, so I use the TIME column, which tells me the total time that this process has been running. Note that the time for all bash processes is zero seconds, though I actually logged into many of these sessions for several hours. The reason for this is because the shell spends most of its time waiting either for you to input something or for the command that you entered to finish. Nothing out of the ordinary here. Because I am the only one on the system at the moment, this value is actually low. A database that is running might have times that are in hours.

On one web server I was on, there was a perl script started from a web page that had averaged about 85% of the CPU time over the last 10 days. This is typically not normal, since pages are usually loaded and then process stops.

Unless I knew specifically on which terminal a problem existed, I would probably have to show every process to get something of value. This would be done with the -e option for "everything". The problem is that I have to look at every single line to see what the total time is. So, to make my life easier, I can pipe it to sort.

Figuring out what is a reasonable value is not always easy. The most effective method I have found is to monitor these values while they are behaving "correctly." You then have a rough estimate of the amount of time particular processes need, and you can quickly see when something is out of the ordinary.

Something else that I use regularly is the PID-PPID pair that occurs when I use the -l option. If I come across a process that doesn't look right, I can follow the PID-to-PPID chain until I find a process with a PPID of 1. Because process 1 is init, I know that this process is the starting point. Knowing this is often useful when I have to kill a process. Sometimes, the process is in an "unkillable" state, which happens in two cases. First, the process may be making the transition to becoming defunct, in which case I can ignore it. Second, it be stuck in some part of the code in kernel mode, in which case it won't receive my kill signal. In such cases, I have found it useful to kill one of its ancestors such as a shell. The hung process is inherited by init and will eventually disappear. However, in the meantime, the user can get back to work. Afterward comes the task of figuring out what the problem was.

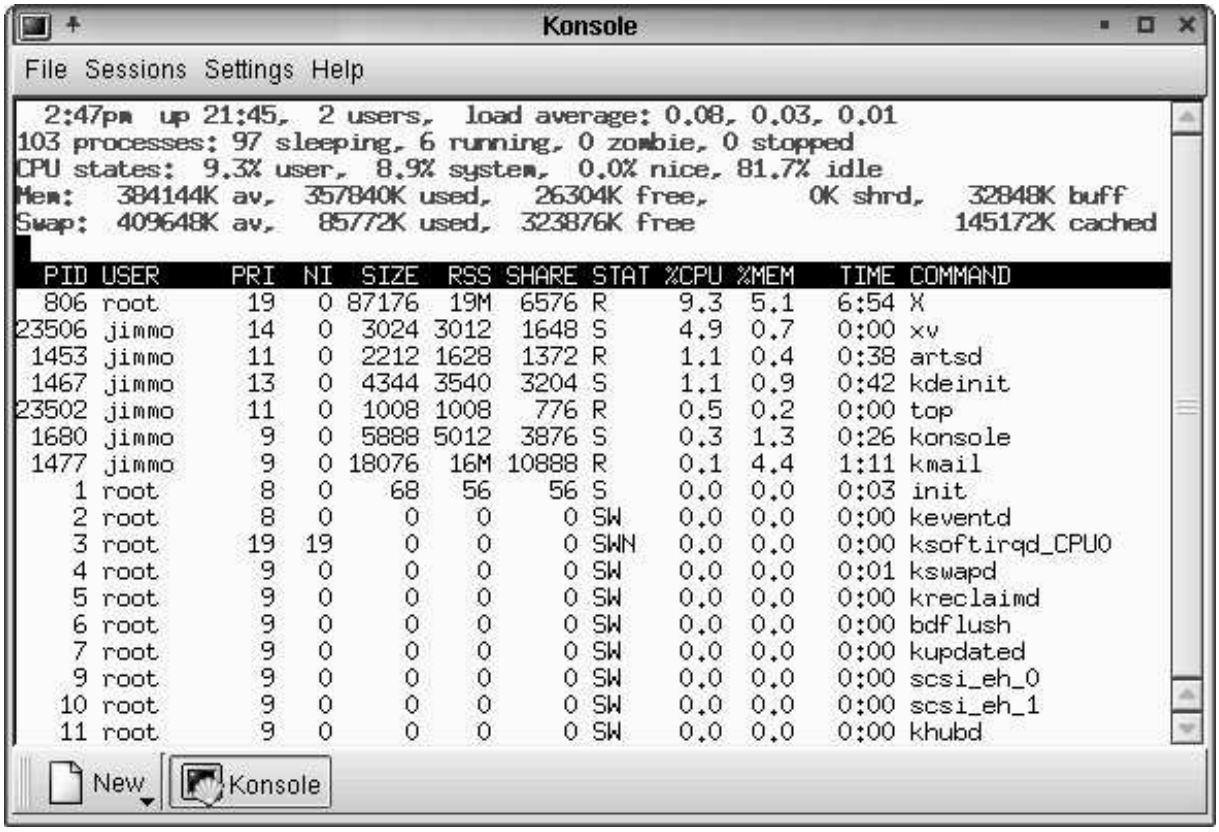
However, you don't need to follow the PID-PPID chain yourself. By using -f, you can get ps to print the output in "forest" modes, whereby the family tree of each process is shown. This might look like this:

```
# ps au --forest
USER       PID %CPU %MEM   VSZ  RSS TTY      STAT START   TIME COMMAND
jimmo     3357  0.0  0.0  2796    4 pts/7    S    Mar27   0:00 /bin/bash
root      6779  0.0  0.0  2200    4 pts/7    S    Mar27   0:00 su
root      6780  0.0  0.2  2776  872 pts/7    S    Mar27   0:00  \_ bash
jimmo     1838  0.0  0.0  2780    4 pts/5    S    Mar27   0:00 /bin/bash
root      1843  0.0  0.0  2200    4 pts/5    S    Mar27   0:00 su -
root      1844  0.0  0.0  2728    4 pts/5    S    Mar27   0:00  \_ -bash
linux-tu  1857  0.0  0.0  2148    4 pts/5    S    Mar27   0:00      \_ su - linux-tu
linux-tu  1858  0.0  0.0  2732    4 pts/5    S    Mar27   0:00          \_ -bash
root      2100  0.0  0.0  2200    4 pts/5    S    Mar27   0:00              \_ su -
root      2101  0.0  0.2  2740  876 pts/5    S    Mar27   0:00                  \_ -bash
root     23470  0.0  0.4  2764 1744 pts/5    R    14:41   0:00                      \_ ps au --forest
jimmo     1799  0.0  0.0  2780    4 pts/4    S    Mar27   0:00 /bin/bash
root      1805  0.0  0.0  2200    4 pts/4    S    Mar27   0:00 su -
root      1806  0.0  0.0  2728    4 pts/4    S    Mar27   0:00  \_ -bash
linux-tu  1819  0.0  0.0  2148    4 pts/4    S    Mar27   0:00      \_ su -l linux-tu
linux-tu  1820  0.0  0.1  2736  600 pts/4    S    Mar27   0:00          \_ -bash
jimmo     1682  0.0  0.0  2784    4 pts/3    S    Mar27   0:00 /bin/bash
root      1730  0.0  0.0  2200    4 pts/3    S    Mar27   0:00 su -
root      1731  0.0  0.0  2736    4 pts/3    S    Mar27   0:00  \_ -bash
linux-tu  1761  0.0  0.0  2148    4 pts/3    S    Mar27   0:00      \_ su - linux-tu
linux-tu  1762  0.0  0.1  2740  760 pts/3    S    Mar27   0:00          \_ -bash
jimmo     1681  0.0  0.0  1536   60 pts/2    S    Mar27   0:00 tail -f /var/log/httpd/error_log
jimmo     1482  0.0  0.0  1524    4 pts/0    S    Mar27   0:00 /bin/cat
root      1268  0.0  0.0  1296    4 tty6     S    Mar27   0:00 /sbin/mingetty tty6
root      1262  0.0  0.0  1296    4 tty5     S    Mar27   0:00 /sbin/mingetty tty5
root      1261  0.0  0.0  1296    4 tty4     S    Mar27   0:00 /sbin/mingetty tty4
root      1260  0.0  0.0  1296    4 tty3     S    Mar27   0:00 /sbin/mingetty tty3
root      1259  0.0  0.3  2092 1208 tty2     S    Mar27   0:00 login -- jimmo
jimmo    23199  0.0  0.3  2708 1504 tty2     S    14:00   0:00 -bash
root      1258  0.0  0.0  1296    4 tty1     S    Mar27   0:00 /sbin/mingetty --noclear tty1
>
```

Here you see a number of different sets of processes that are related. In a couple of cases, one did an su to root su - and then to another user. In one of these cases, the ps au --forest command that make this output can been seen.

One nice thing is that ps has its own sort options, so you don't need to pipe it through sort. Also, you can select which information you want displayed and even on which order.

Another very useful tool is **top**. Without any option it shows you something like the following image.



In the upper portion of the screen is some general system information about how many users are on the system, how long the system has been running, the number of process, memory usage and so forth. In the lower portion you see a process list. This is sorted so that the most active processes are at the top of the list. One advantage of top is that it constantly refreshes, so you get update information every five seconds this can be changed, if you want.

As you might expect, there are many different configuration options to choose from. You can even configure top, so that you cannot interrupt it keeps on running. That way you can start it on a terminal to give you constantly updated information about your system without the danger of someone breaking out of it an having access to a shell.

10.2.3 Files and File Systems

Another thing you should monitor is how much space is left on your file systems. I have seen many instances in which the root file system gets so close to 100 percent full that nothing more can get done. Because the root file system is where unnamed pipes are created by default, many processes die terrible deaths if they cannot create a pipe. If the system does get that full, it can prevent further logins (because each login writes to log files). If root is not already logged in and can remove some files, you have problems.

Fortunately, by default, when **mke2fs** creates a file system, a little space is reserved for root. This prevents the system from becoming completely full, so you have a chance to do something about it.

So the solution is to monitor your file systems to ensure that none of them get too full, especially the root file system. A rule of thumb, whose origins are lost somewhere in UNIX mythology, is that you should make sure that there is at least 15 percent free on your root file system. Although 15 percent on a 200MB hard disk is one-tenth the amount of free space as 15 percent on 2Gb drive, it is a value that is easy to monitor. Consider 10-15Mb as a danger sign, and you should be safe. However, you need to be aware of how much and how fast the system can change. If the system *could* change 15Mb in a matter of hours, then 15Mb may be too small a margin. When you consider that a 100 GB drive costs about $100 (early 2005), then their really is little reason *not* to go out and get a new drive before the old one gets too full.

Use **df** to find out how much free space is on each mounted file system. Without any options, the output of **df** is one file system per line, showing how many blocks and how many inodes are free. Though this is interesting, I am really more concerned with percentages. Very few administrators know how long it takes to use 1,000 blocks, though most understand the significance if those 1,000 blocks mean that the file system is 95 percent full.

Because I am less concerned with how many inodes are free, the option I use most with **df** is -v, which shows the data block usage. On an older system I had, I got something like this:

```
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/hda4              4553936    571984   3750616  14% /
/dev/hda2                54447      5968     45668  12% /boot
/dev/hdc1              4335064   3837292    277556  94% /data
/dev/sda2              3581536    154920   3244680   5% /home
/dev/sda1              5162796   1783160   3117380  37% /opt
/dev/hdc2              4335064   1084964   3029884  27% /oracle
/dev/hdc3              5160416   2542112   2356152  52% /usr
/dev/hdc4              5897968   1532972   4065396  28% /usr/vmware
shmfs                   192072         0    192072   0% /dev/shm
>
```

The shortcoming with df is that it tells you about the entire hard disk but can't really point to where the problems are located. A full file system can be caused by one of two things. First, there can be a few large files, which often happens when log files are not cleaned out regularly.

The other case is when you have a lot of little files. This is similar to ants at a picnic: individually, they are not very large, but hundreds swarming over your hotdog is not very appetizing. If the files are scattered all over your system, then you will have a hard time figuring out where they are. At the same time, if they are scattered across the system, the odds are that no single program created them, so you probably want (if not need) them all. Therefore, you simply need a bigger disk.

If, on the other hand, the files are concentrated in one directory, it is more likely that a single program is responsible. As with the large log files, a common culprit are the files in **/var/log**.

To detect either case, you can use a combination of two commands. First is find, which, you already know from previous encounters, is used to find files. Next is **du**, which is used to determine disk usage. Without any options, du gives you the disk usage for every file that you specify. If you don't specify any, it gives you the disk usage for every file from your current directory on down.

Note that this usage is in blocks because even if a block contains a single byte, that block is used and no longer available for any other file. However, if you look at a long listing of a file, you see the size in bytes. A 1 byte file still takes up one data block. The size indicated in a long directory listing will usually be less than what you get if you multiple the number of blocks by the size of the block (512 bytes). To get the sum of a directory without seeing the individual files, use the -s option.

To look for directories that are exceptionally large, you can find all the directories and use du -s. You also need to be sure that you don't count multiple links more than once, so include the -u option as well. Then, sort the output as numerical values and in reverse order (-nr) to see the larger directories first, like this:

```
find / -type d -exec du -us {} \; | sort -nr > /tmp/fileusage
```

I redirected the output into the file **/tmp/fileusage** for two reasons. First, I have a copy of the output that I can use later if I need to. Second, this command is going to take a *very* long time. Because I started in /, the command found this directory (/) first. Therefore, the disk usage for the entire system (including mounted file system) will be calculated. Only after it has calculated the disk usage for the entire system does it go on to the individual directories.

You can avoid this problem in a couple of ways. First, use -print instead of -exec in the find and then pipe it first to grep -v. This strips out /, and you can then pipe that output to **xargs**. This way, you avoid the root directory.

Personally, this is not very pretty, especially if I were going to be using the command again. I would much rather create a list of directories and use this as arguments to du. That way I can filter out those directories that I don't need to check or only include those that I do want to check. For example, I already know that **/var/log** might contain some large files.

On occasion, it's nice to figure out what files a process has open. Maybe the process is hung and you want some details before you decide to kill it.

10.2.4 Checking Other Things

UNIX performance tuning is often considered a black art. I've talked with many customers who call in to tech support, expecting that we will say a few magic words, wave our wizards wand, and then abracadabra, their systems will run better. This misconception is often compounded by the fact that support engineers often don't have the time to go into long, detailed explanations and instead quickly look over output from various system utilities and

tell the customer to increase kernel parameter X or change setting Y. Miraculously, the system instantly runs better. From the customers stand point, this is "magic."

Well, not really. Some customers do express their frustration at not being able to improve the situation themselves. This is not because they aren't smart enough, but it is the same reason why many people take their cars to a mechanic for a tune-up. By comparison to replacing the block, a tune-up is a relatively simple procedure. However, many people don't have the skills to do it themselves.

This applies to system tuning as well. Because many customers do not have the skills, they turn to the mechanic to do it for them. I remember when I was about 18 and had a couple of friends who were real car enthusiasts. When their cars suddenly started making strange sounds, I can still remember them saying that the franistan had come loose from the rastulator. Well, at least that's what it sounded like to me at the time. The reason why I couldn't figure that out myself was that I didn't have the training or experience. However, they had the experience and could tell what the problem was just by listening to the car. My not being able to tell what was wrong with a car just by listening to it, is the same as many system administrators, who don't have the training or experience to tune an Linux system. However, you can.

Although a site like this one cannot provide the experience, it can provide some of the training. Keeping with the car analogy, I've talked about the transmissions, the breaks, the drive shaft, the electrical system, and even the principles of the internal combustion engine. With that knowledge, you can now understand why it is necessary to have clean spark plugs or the proper mixture of air and gasoline.

With a cars engine, you often get a "feeling" for its proper behavior. When it starts to misbehave, you know something is wrong, even though you may not know how to fix it. The same applies in principle to a Linux system, though many garages can afford the high-tech equipment that plugs your into you car and shows you what the car is doing. From there, it is a simple step for the mechanic to determine the proper course of action. What you need for a Linux system is a tool that does the same thing.

10.3 Big Brother

Winston sat in the darkened room. All around him lights blinked and flashed. Regardless of what it did, Big Brother knew. There was nothing he could do, without Big Brother being aware. There was also the chance that should Winston do something that Big Brother did not like, alarms would sound and people would come storming into the room. Thank goodness , Winston just sat quietly.

Although this sounds like it comes right out of George Orwell's 1984, we're talking about a different Winston and different Big Brother. In fact, Winston can go by many names. However, regardless of what name is used, it is under the ever present watch of a real-life Big Brother, who watches and waits for one false move.

Fortunately, this Big Brother is truly your friend. However, like its Orwellian namesake, it is constantly on the look-out for things it doesn't like, waiting to sound an alarm. What we are talking about is a systems monitoring tool developed by Sean MacGuire of The MacLawran Group (www.maclawran.ca//bb-dnld/).

In essence, there is not much that Big Brother does, which you cannot find in commercial monitoring tools. Although many commercial products are available with more features, Big Brother has a number of advantages which makes it ideal for many environments. One of the most compelling reasons to use Big Brother is its simplicity. It is composed of just a handful of scripts and programs, which the clients use to gather information and report it to a central server where the information is collected and displayed in an extremely accessible format (see figure 1).

The configuration and scripts themselves are very easy to change which allows you to fit Big Brother to your current needs, while at the same time giving you room to grow, without putting a strain on your budget. Although it is not covered directly by the General Public License, you can download for free from the MacLawran Group's web site. It is covered by a "fair use" license however to redistribute it, you need written permission from the MacLawran Group. See their web site for more details.

What Big Brother is Made of

Big brother is a made up of five key components. There is a central monitoring station or display server, which receives incoming messages from the clients, processes them and makes them available in the form of Web pages. This means that although the display server currently run only on UNIX or Linux machines, you can monitor from anywhere. (Shell scripts refer to this server using the BBDISPLAY variable.)

Network connectivity is monitored from the server using a shell script (bb-network.sh), which tries to contact the various clients using a simple ping. In addition, you can configure Big Brother to check for connectivity to specific services on the client, such as HTTP, FTP, and SMTP. The machine that checks connectivity does not necessarily need to be the display server and is referred to in the scripts as BBNET. In some cases, it might make sense to have the BBNET machine sit on a machine acting as a router and direct the information it gathers to a server inside a firewall, for example, which would allow you to check connectivity to the Internet, as well.

It some cases, checking connectivity to the machine is sufficient. If so, you do not need to do any further configuration as all the work is done by the server. However, if the machine needs to report information to the display server (such as disk space, running processes, etc.), you need to configure the client. This is accomplished by the "Local System Monitor" component which is embodied in the shell script bb-local.sh.

Big Brother can also be configured to send pager messages to the system administrator, based on any monitored event. When one of those events occur , the machine where the event occurred uses the bb-page.sh shell script to send a message to the BBPAGER server, which then uses Kermit to contact the pager via modem. Like the BBNETserver, the BBPAGER server does not need to be the same machine as the display server. Note that Kermit is not

included as part of the big brother package. Finally, there are a handful of support programs, such as the Big Brother daemon (bbd) which sits on the various servers and clients program (bb), which sends the appropriate messages to the display and pager servers.

Keep in mind that the exact same event on two different machines can have different meanings depending on your environment. For example, a database server, which stores all of the data in a single file may take up an entire filesystem and from the operating system's standpoint, the size never changes. Therefore, the filesystem might be constantly at 99% full. As a result, you probably won't want report it. However, a file server that reaches 80% full might generate a warning and then a "panic" when the file system reaches 95% full.

Another important aspect is that monitoring is not just limited to UNIX and Linux machines. Instead clients are available for Windows NT and NetWare. Note that only the binaries are provided for Windows NT and NetWare clients.

The primary configuration file is etc/bb-hosts, which sits under the directory where you install Big Brother. By default, this is /usr/local/bb, but can be changed when you compile the source for your specific system. The bb-hosts file has a similar structure to the /etc/hosts, but also include information about what whether or not connectivity should be checked, and if so, specifically what network services should be looked at.

Since all of the pages are prepared using HTML, your BBDISPLAY server needs to have a an HTTP server running on it in order to server the pages to the clients. Big Brother updates the page index.html in regular intervals so the information is always current. However, this does not mean you have to keep pressing the refresh or update button to see the current information, as each page is automatically updated every 60 seconds.

At the top of the page in the figure below is a legend, describing what each of the different colors mean, along with the time the page was last updated and links to various sources of information. Clicking on the picture of Big Brother (actually Sean MacGuire), brings you to the Big Brother home page.

Big Brother also has the ability to group machines, so that they are displayed in a separate table. Each column represents one specific aspect being monitored, with the rows being the particular system. If something is not monitored on a particular system, there will be a dash in that cell of the table. Otherwise, there will be one of the colored balls listed in the legend. If you had it up in a browser, you would see that the yellow and red balls are blinking, which gives them an additional visual effect.

Note that the column headings (what is being monitored) are links to a help page. If you click on them, you are brought to the exact spot for that particular aspect, which gives you details about what is being monitored, as well as what script is used to do the monitoring and in some cases specific information about how the monitoring is being done.

Setting up Display Server

The first step is to get Big Brother and extract the files. Configure it for your system by changing into the ./doc directory and running ./bbconfig. , where is the name of the your OS. Leave off the OS name for a list of supported systems. However, you need to make sure you

get this right or Big Brother will not compile correctly.

Next, change to the ./src under the Big Brother root. Run "make" to compile all of the components, then "make install" to install them in their default location. If you need to change the location, you can change the location in the Makefile.

After the binaries are installed, edit runbb.sh in the Big Brother root directory and set the BBHOME variable to the directory where you installed Big Brother. This is extremely important as the entire functionality centers around this variable. Next,change to ./etc and edit the bb-hosts file and determines what aspects f the system will be monitored. It has a structure similar to a tradition hosts file, but is broken into three parts:

```
IP-ADDR HOSTNAME        # DIRECTIVES>
```

If you have turned on fully qualified domain names (the FQDN variable), then the HOSTNAME **must** also contain the domain. DIRECTIVES is essentially a list of what needs to be monitored on the remote site. If this machine is going to be one of the servers, then you should set the appropriate variable(s) in the list of directives (BBDISPLAY, BBPAGER or BBNET depending of what kind of server this is. Table 1 shows you a list of the more common directives. For some examples, check the bb-hosts file provided for you. (You will to have to change it anyway). Finally, run ./bbchkcfg.sh and ./bbchkhosts.sh to check for errors in the configuration files.

Remember that the Big Brother information is displayed by the HTTPD server. Therefore you need to tell your web server where to get the Big Brother pages. This is normally done by making a symbolic link between the Big Brother root directory and somewhere underneath your DocumentRoot for your web server. Commonly this is just the DocumentRoot/bb, so all you have to do is enter and you're in. Note that you must make sure that the server is configure to follow the symbolic links.

When you think you have this configured correctly, move back to the Big Brother root directory and start Big Brother by running the script ./runbb.sh. Like many script which starts services, you can use the arguments start, stop and restart. Note that by default, Big Brother is not configured to run when the system boots, so you probably need to add something to your /etc/rc.d directory (or where the start-up scripts are on your machine). If BB fails to start, check the file BBOUT for any errors. At this point, the server display server should be ready to go.

Setting up the clients

Configuring the UNIX clients is very easy if they have the same operating system and hardware as the server. If you have different operating systems, then you have seen that the same program will deliver a different output, sometimes with the same options. Therefore, you need to have each program deliver the information in a format that Big Brother can understand. That's why you need to configure Big Brother at the very beginning.

Even if your machines are different, you should configure your bb-hosts file to list all of the clients. Although you **could ** have a different bb-hosts file on each machine, I find it is easier to use the same file on each machine, although it may require the bb-hosts file to be

edited if you want to monitor different things on each client.

Once the bb-hosts file is configure correctly, you create a "tarball" for the client using the script doc/bbclient . This tarball contains the necessary programs and files. Copy the tarball into the BBHOME directory on the client and unpack it. (Note you could have different BBHOMES on the clients, but I think that just makes things harder to manage.)

If you have different platforms, you will need to install a client on one machine for each different platform and then create a tarball, which is then copied to the other clients of that type. Make sure to copy your master bb-hosts file (it has a common format across platforms) and check your configuration using bbchkcfg.sh and bbchkhosts.cfg.

Although the steps are fairy simple, configuring a Windows NT clients is completely different that for UNIX. As of this writing, the Big Brother client is only available in binary form as a zip archive, which you need to unpack on a local drive. Both Alpha (axp) and Intel (x86) versions are available and you need to rename the appropriate one for your system. For example, on an Intel machine you would run:

ren bbnt-x86.exe bbnt.exe

You next run bbnt.exe to install the program.

bbnt [-y] -install BBDISPLAY FQDN IPPORT

Here BBDISPLAY is the IP address of the Big Brother display server. FQDN is either Y or N depending on whether or not Big Brother should return the Fully qualified domain name of the machine. IPPORT is the port used for communication between the local Big Brother client and the Big Brother server. It is important that you use the exact same port as the server has, otherwise the server will not get updated. Note that the -y option simply tells Big Brother to install without prompting for confirmation.

Note that that Big Brother runs as a system service under windows NT. Once installed it can be managed from either the Control Panel -> Services, Server Manager->Services, or the command line using the net command (e.g. net stop, net start, and so on). Because it is running as a service, there is a potential security problem should the bbnt.exe program get replaced. Therefore, you need to make sure that it is readable, executable and writable by an administrative account **only**.

The NT client also has two addition options to either upgrade (-upgrade) or remove Big Brother completely (-remove). In order to avoid any problems it is a good idea to stop the service before you try to remove or upgrade it.

Configuring Big Brother for your site

The NT clients has a very comfortable GUI configuration interface (bbcgf.exe). In general, the available configuration options are self-explanatory. However, detailed configuration instructions are available in the included README file.

I have managed several machines where the default configuration on UNIX machines is sufficient. However, Big Brother has a number of different parameters which you can use. The primary configuration file is etc/bbdef.sh (the BB definitions file), which bbrun.sh reads when it starts up. Here you define not only basic parameters (like whether or not to display the fully-qualified names), but also specific behavior such as how full the disk needs to be before it is reported.

By default, Big Brother monitors your filesystems and reports when they get too full (90% warning, 95% panic). These levels can be changed globally by setting the DFWARN and DFPANIC variables to different values, depending on your needs. One useful aspect is the ability to define these levels on a per filesystem basis. For example, many sites have databases which take up entire partitions. Although the information in the database changes, how much space is taken up doesn't. If DFPANIC is set to 95%, for example, but the filesystem is **always** at 99%, you will get a lot of unnecessary messages. The trick is to copy the file etc/bb-dftab INFO to etc/bb-dftab and edit to suit your needs.

Similarly, the CPUWARN and CPUPANIC are used to report on CPU activity. These are based on the load average as reported by the uptime command (and then multiplied by 100). By default, these are set to 150 and 300 respectively, and can also be changed in the bbdef.sh file.

You can also monitor specific processes to see if they are running on the client. The PROCS variable defines which processes to monitor and report as a warning if they are not running, whereas the PAGEPROC defines which are defined as a panic. The intent here is that should the specified process not be running, Big Brother will page the administrator.

The MSGS and PAGEMSGS are used to monitor log files. By default, MSGS is set to "NOTICE WARNING". This means that only messages containing NOTICE or WARNING are looked at. PAGEMSG is set to "NOTICE", which means that should such a message appear, it generates a red condition (panic) and would page an administrator if the system is so configured. There is also the IGNMSGS variable, which tells Big Brother which messages to specifically ignore.

Keep in mind that the runbb.sh script reads the configuration information from bbdef.sh file when it is started. Therefore, unlike other Linux daemons, changes are not recognized until Big Brother is restarted. Therefore, after each change, you will have to restart Big Brother.

Although you probably will not need to change it, another important file is etc/bbsys.sh. This contains information about your specific operating system, such as the location of programs on your system, and which arguments to use. My suggestion is that you do not mess with this file unless you know what you are doing. It is generated for you when you install Big Brother and there is normally no need to change it.

Another important variable is BBTMP which defines the location of the directory, which Big Brother users to store temporary files. By default this is $BBHOME/tmp and there is generally no reason to change it. However, if you do want it somewhere else, you need to make sure that normal users do not have write access as it represents a potential security hole.

Configuration of Big Brother goes beyond just setting variables. Obviously, because you have the source code and it is mostly scripts, you can define the behavior anyway you want. The problem with changing the scripts and programs is that you will probably have to re-do a lot of work, when Big Brother gets updated. Instead, Big Brother provides a mechanism whereby you can define your own tests. That is, you can configure Big Brother to run addition or "external" tests. These are located in the $BBHOME/ext directory and a template is provided to show you the general syntax.

Next Steps

The good news **and** the bad news is that’s not all. It’s good for system administrators because there is a lot more you can do with Big Brother than we discussed here. It’s bad for me because we don’t have the space here to go into all of the details. You already have the basics to install Big Brother and be able to monitor the key aspects of your system.

One thing we did not cover much is the notification features. As I mentioned, Big Brother can be configured to send email, pages (including the Short Message Service - SMS), and other kinds of notification to administrator when certain events occur. Unfortunately, being as powerful as it is, discussing this topic requires more space that we have.

My suggested is that once you have it running, you should investigate the scripts and configuration files. Once you have Big Brother running (maybe even before you install it), I would suggest you subscribe to the Big Brother mailing list. (send email to majordomo@taex001.tamu.edu with the **body** of the message "subscribe bb"). This provides a great forum for asking your questions, as well as getting some neat tips and tricks from more advanced users. Both the scripts and the mailing list provide a wealth of information about what Big Brother can already done and what more you can do with it.

Table 1 - Common bb-host directives

BBDISPLAY Central server where information is displayed.
BBPAGER Server which sends pages to administrators.
BBNET Host which check connectivity to the network services.
http://www_path Host and path to check
for http connections. Multiple paths can be specified by separating using a pipe (’|).
ftp Check the ftp service
smtp Check the smtp server
pop3 Check the pop3 server
telnet Check the telnet service
ssh Check the ssh server
nntp Check the nntp server
dns Checks for name resolution server
noping Don’t do ping test for this host
dialup If host is down then display clear button

NOTE: The service name must be written exactly the was it does in /etc/services. On some system the POP3 server is written ’’pop-3’’.

Chapter 11 Solving Problems

Using "Solving Problems" as the title for this chapter was a very conscious decision. I intentionally avoided calling it "Troubleshooting" for several reasons. First, troubleshooting has always seemed to me to be the process by which we look for the causes of problems. Although that seems like a noble task, so often finding the cause of the problem doesn't necessarily mean finding the means to correct it or understanding the problem.

The next reason is that so often I find books in which the troubleshooting section is just a list of problems and canned solutions. I find this comparable to the sign "In case of fire, break glass." When you break the glass an alarm goes off and the fire department comes and puts out your fire. You may never know what really caused the problem nor what you can do to solve similar problems in the future.

The troubleshooting sections that I find most annoying list 100 problems and 100 solutions, but I usually have problem 101. Often I can find something that is similar to my situation, and, with enough digging through the manuals and poking and prodding the system, I eventually come up with the answer. Even if the answer is spelled out, it's usually a list of steps to follow to correct the problem. There are no details about what caused the problem in the first place or what the listed steps are actually doing.

Not all problems are alike, and not all problems can be approached in the exact same manner. However, there are some basic issues that are common to every problem and there are basics steps you can take, no matter what the cause.

11.1 Solving Problems Yourself

In this chapter, I am not going to give you a list of known problems and their solutions. A lot of ideas in the HOWTOs do that for you. I am not going to give you details of the system in which you need to find the solution yourself. I hope I did that in the first part of the book. What I am going to do here is to talk about the techniques and tricks that I've learned over the years to track down the cause of problems. Also, I'll talk about what you can do to find out where the answer is, if you don't have the answer yourself.

11.1.1 Preparing Yourself

Problem-solving starts before you have even installed your system. Because a detailed knowledge of your system is important to figuring out what's causing problems, you need to keep track of your system from the very beginning. One most effective problem-solving tool costs about $2 and can be found in grocery stores, gas stations, and office supply stores. Interestingly enough, I cant remember ever seeing it in a store that specialized in either computer hardware or software. I am talking about a notebook. Although a bound notebook will do the job, I find a loose-leaf notebook to be more effective because I can add pages more easily as my system develops.

In the notebook I include all the configuration information from my system, the make and model of all my hardware, and every change that I make to my system. This is a running

record of my system, so the information should include the date and time of the entry, as well as the person making the entry. Every time I make a change, from adding new software to changing kernel parameters, should be recorded in my log book.

In putting together your notebook, don't be terse with comments like, "Added SCSI patch and relinked." This should be detailed, like, "Added patch for Adaptec AIC-7xxx. Rebuild and reboot successful." Although it seems like busy work, I also believe things like adding users and making backups should be logged. If messages appear on your system, these, too, should be recorded with details of the circumstance. The installation guide should contain an "installation checklist." I recommend that you complete this before you install and keep a copy of this in the log book.

Something else that's very important to include in the notebook is problems that you have encountered and what steps were necessary to correct that problem. One support engineer with whom I worked told me he calls this his "solutions notebook."

As you assemble your system, write down everything you can about the hardware components. If you have access to the invoice, a copy of this can be useful for keeping track of the components. If you have any control over it, have your reseller include details about the make and model of all the components. I have seen enough cases in which the invoice or delivery slip contains generic terms like Intel 800Mhz CPU, cartridge tape drive, and 5GB hard disk. Often this doesn't even tell you whether the hard disk is SCSI, IDE, or what.

Next, write down all the settings of all the cards and other hardware in your machine. The jumpers or switches on hardware are almost universally labeled. This may be something as simple as J3 but as detailed as IRQ. Linux installs at the defaults on a wide range of cards, and generally there are few conflicts unless you have multiple cards of the same type. However, the world is not perfect and you may have a combination of hardware that neither I nor Linux developers has ever seen. Therefore, knowing what *all* the settings are can become an important issue.

One suggestion is to write this information on gummed labels or cards that you can attach to the machine. This way you have the information right in front of you every time you work on the machine.

Many companies have a "fax back" service in which you can call a number and have them fax you documentation of their products. For most hardware, this is rarely more than a page or two. For something like the settings on a hard disk, however, this is enough. Requesting faxed documentation has a couple of benefits. First, you have the phone number for the manufacturer of each of your hardware components. The time to go hunting for it is not when your system has crashed. Next, you have (fairly) complete documentation of your hardware. Last, by collecting the information on your hardware, you know what you have. I cant count the number of times I have talked with customers who don't even know what kind of hard disk they have, let alone what the settings are.

Another great place to get technical information is the World Wide Web. I recently bought a SCSI hard disk that did not have any documentation. A couple of years ago, that might have bothered me. However, when I got home, I quickly connected to the Web site of the driver

manufacturer and got the full drive specs, as well as a diagram of where the jumpers are. If you are not sure of the company's name, take a guess, as I did. I tried *www.conner.com*, and it worked the first time.

When it comes time to install the operating system, the first step is to read the release notes and installation HOWTO and any documentation that comes with your distribution. I am not suggesting reading them cover to cover, but look through the table of contents completely to ensure that there is no mention of potential conflicts with your host adapter or the particular way your video card needs to be configured. The extra hour you spend doing that will save you several hours later, when you cant figure out why your system doesn't reboot when you finish the install.

As you are actually doing the installation, the process of documenting your system continues. Depending on what type of installation you choose, you may or may not have the opportunity to see many of the programs in action. If you choose an automatic installation, many of the programs run without your interaction, so you never have a chance to see and therefore document the information.

The information you need to document are the same kinds of things I talked about in the section on finding out how your system was configured. It includes the hard disk geometry and partitions (**`fdisk`**), file systems (**`mount`** and **`/etc/fstab`**), the hardware settings (**`/var/log/messages`**), and every patch you have ever installed. You can send the output to all of these commands to a file that you can print out and stick in the notebook.

I don't know how many times I have said it and how many articles (both mine and others) in which it has appeared, some people just don't want to listen. They often treat their computer systems like a new toy at Christmas. They first want to get everything installed that is visible to the outside world, such as terminals and printers. In this age of "Net-in-a-box," often that extends to getting their system on the Internet as soon as possible.

Although being able to download the synopsis of the next Deep Space Nine episode is an honorable goal for some, Chief O'Brien is not going to come to your rescue when your system crashes. (I think even he would have trouble with the antiquated computer systems of today.)

Once you have finished installing the operating system, the very first device you need to install and configure correctly is your tape drive. If you don't have a tape drive, buy one! Stop reading right now and go out and buy one. It has been estimated that a "down" computer system costs a company, on the average, $5,000 an hour. You can certainly convince your boss that a tape drive that costs one-tenth as much is a good investment.

One of the first crash calls I received while I was in tech support was from the system administrator at a major airline. After about 20 minutes, it became clear that the situation was hopeless. I had discussed the issue with one of the more senior engineers who determined that the best course of action was to reinstall the OS and restore the data from backups.

I can still remember their system administrator saying, "What backups? There are no backups."

"Why not?" I asked.

"We don't have a tape drive."

"Why not?"

"My boss said it was too expensive."

At that point the only solution was data recovery service.

"You don't understand," he said. "There is more than $1,000,000 worth of flight information on that machine."

"Not any more."

What is that lost data worth to you? Even before I started writing my first book, I bought a tape drive for my home machine. For me, it's not really a question of data but rather, time. I don't have that much data on my system. Most of it can fit on a half-dozen floppies. This includes all the configuration files that I have changed since my system was installed. However, if my system was to crash, the time I save restoring everything from tape compared to *reinstalling*from floppies is worth the money I spent.

As technology progressed, CD writers became cheaper than tape drives. Current I make backups onto CD-ROMS of the most important data, so I can get to it quickly, but I use my tape drive to backup *all* of the data and system files, as it won't fit on a floppy.

The first thing to do once the tape drive is installed is to test it. The fact that it appears at boot says nothing about its functionality. It has happened enough that it appears to work fine, all the commands behave correctly, and it even looks as though it is writing to the tape. However, it is not until the system goes down and the data is needed that you realize you cannot read the tape.

I suggest first trying the tape drive by backing up a small subdirectory, such as **`/etc`**. There are enough files to give the tape drive a quick workout, but you don't have to wait for hours for it to finish. Once you have verified that the basic utilities work (like **`tar`** or **`cpio`**), then try backing up the entire system. If you don't have some third-party back-up software, I recommended that you use cpio. Although tar can back up most of your system, it cannot backup device nodes.

If the Linux commands are too cumbersome (and they are for many newcomers), a couple of commercial products are available. One such product is Lone-Tar from Cactus International. I have used Lone-Tar for years on a few systems and have found it very easy to use. The front end is mostly shell scripts that you can modify to fit your needs.

In general, Lone-Tar takes a differential approach to making backups. You create one Master Backup and all subsequent backups contain those files that have changed since the master was created. I find this the best approach if your master backup takes more than one tape. However, if it all fits on one tape, you can configure Lone-Tar always to do masters.

Cactus also produces several other products for Linux, including Kermit, and some excellent DOS tools. I suggest you check them out. Demo versions are available from the cactus Web site.

Like religion, it's a matter of personal preference. I use Lone-Tar for Linux along with their DOS Tar product because I have a good relationship with the company president, Jeff Hyman. Lone-Tar makes backups easy to make and easy to restore. There is even a Linux demo on the Lone-Tar Web site. The Craftworks distribution has a demo version of the BRU backup software.

After you are sure that the tape drive works correctly, you should create a boot/root floppy. A boot/root floppy is a pair of floppies that you use to boot your system. The first floppy contains the necessary files to boot and the root floppy contains the root file system.

Now that you are sure that your tape drive and your boot/root floppy set work, you can begin to install the rest of your software and hardware. My preference is to completely install the rest of the software first, before moving on to the hardware. There is less to go wrong with the software (at least, little that keeps the system from booting) and you can, therefore, install several products in succession. When installing hardware, you should install and test each component before you go on to the next one.

I think it is a good idea to make a copy of your kernel source (**`/usr/src/linux`**) before you make any changes to your hardware configuration or add any patches. That way, you can quickly restore the entire directory and don't have to worry about restoring from tape or the distribution CD-ROM.

I suggest that you use a name that is clearer than **`/usr/src.BAK`**. Six months after you create it, you'll have no idea how old it is or whether the contents are still valid. If you name it something like **`/usr/src.06AUG95`**, it is obvious when it was created.

Now, make the changes and test the new kernel. After you are sure that the new kernel works correctly, make a new copy of the kernel source and make more changes. Although this is a slow process, it does limit the potential for problems, plus if you do run into problems, you can easily back out of it by restoring the backup of the link kit.

As you make the changes, remember to record all the hardware and software settings for anything you install. Although you can quickly restore the previous copy of the kernel source if something goes wrong, writing down the changes can be helpful if you need to call tech support or post a message to the Internet.

Once the system is configured the way you want, make a backup of the entire installed system on a different tape than just the base operating system. I like to have the base operating system on a separate tape in case I want to make some major revisions to my software and hardware configuration. That way, if something major goes wrong, I don't have to pull out pieces, hoping that I didn't forget something. I have a known starting point from which I can build.

At this point, you should come up with a back-up schedule. One of the first things to consider is that you should backup as often as necessary. If you can only afford to lose one days worth of work, then backing up every night is fine. Some people back up once during lunch and

once at the end of the day. More often than twice a day may be too great a load on the system. If you feel that you have to do it more often, you might want to consider disk mirroring or some other level of RAID.

The latest kernel versions support RAID 0 (disk striping), which, although it provides an improvement in performance, has no redundancy. Currently, I am not aware of any software RAID solutions, though some hardware solutions might work with Linux.

The type of backup you do depends on several factors. If it takes 10 tapes to do a backup, then doing a full backup of the system (that is, backing up *everything*) every night is difficult to swallow. You might consider getting a larger tape drive. In a case where a full backup every night is not possible, you have a few alternatives.

First, you can make a list of the directories that change, such as **`/home`** and **`/etc`**. You can then use tar just to backup those directories. This has the disadvantage that you must manually find the directories that change, and you might miss something or back up too much.

Next, there are incremental backups. These start with a master, which is a backup of the entire system. The next backup only records the things that have changed since the last incremental. This can be expanded to several levels. Each level backs up everything that has changed since the last backup of that or the next lower level.

For example, level 2 backs up everything since the last level 1 or the last level 0 (whichever is more recent). You might do a level 0 backup once a month (which is a *full* backup of everything), then a level 1 backup every Wednesday and Friday and a level 2 backup every other day of the week. Therefore, on Monday, the level 2 will back up everything that has changed since the level 1 backup on Friday. The level 2 backup on Tuesday will back up everything since the level 2 backup on Monday. Then on Wednesday, the level 1 backup backs up everything since the level 1 backup on the previous Friday.

At the end of the month, you do a level 0 backup that backs up everything. Lets assume this is on a Tuesday. This would normally be a level 2. The level 1 backup on Wednesday backs up everything since the level 0 backup (the day before) and not since the level 1 backup on the previous Friday.

A somewhat simpler scheme uses differential backups. Here, there is also a master. However, subsequent backups will record *everything* that has changed (is different) from the master. If you do a master once a week and differentials once a day, then something that is changed on the day after the master is recorded on every subsequent backup.

A modified version of the differential backup does a complete, level 0 backup on Friday. Then on each of the other days, a level 1 backup is done. Therefore, the backup Monday-Thursday will backup everything since the day before. This is easier to maintain, but you may have to go through five tapes.

The third type, the simplest method, is where you do a master backup every day and forget about increments and differences. This is the method I prefer if the whole system fits on one tape because you save time when you have to restore your system. With either of the other methods, you will probably need to go through at least two tapes to recover your data, unless

the crash occurs on the day after the last master. If you do a full backup every night, then there is only one backup to load. If the backup fits on a single tape (or at most, two), then I highly recommend doing a full backup every night. Remember that the key issue is getting your people back to work as soon as possible. The average $5,000 per hour you stand to loose is much more than the cost of a large (8Gb) tape drive.

This brings up another issue, and that is rotating tapes. If you are making either incremental or differential backups, then you *must* have multiple tapes. It is illogical to make a master then make an incremental on the same tape. There is no way to get the information from the master.

If you make a master backup on the same tape very night, you can run into serious problems as well. What if the system crashes in the middle of the backup and trashes the tape? Your system is gone and so is the data. Also, if you discover after a couple of days that the information in a particular file is garbage and the master is only one day old, then it is worthless for getting the data back. Therefore, if you do full backups every night, use at least five tapes, one for each day of the week. (If you run seven days a week, then seven tapes is likewise a good idea.)

You don't necessarily always have to back up to tape. If the amount of data that changes is fairly small, you could backup to floppies. This is probably only valid if your system is acting as a Web server and the data change at irregular intervals. As with any backup, you need to weigh the time to recreate the data against the time to make the backup. If your data on the Web server is also stored elsewhere (like on the development machine), it may be easier to back up the Web server once after you get your configuration right, and then skip the backups. However, it's your call.

Other choices for backup media include WORM (Write Once/Read Many) drive and CD-Recordable. This is only effective if the data isn't going to change much. You could back up your Web server to one of these media and then quickly recovered it if your machine crashes. Copy the data to another machine on the network where a backup is done. (You could also mount the file system you want to back up via NFS.)

Although most people get this far in thinking about tapes, many forget about the physical safety of the tapes. If your computer room catches fire and the tapes melt, then the most efficient backup scheme is worthless. Some companies have fireproof safes in which they keep the tapes. In smaller operations, the system administrator can take the tape home from the night before. This is normally only effective when you do masters every night. If you have a lot of tapes, you might consider companies that provide off-site storage facilities.

Although some commercial products are available (which I will get into in a moment), you can use the tools on your system. For example, you can use **`tar`** or **`cpio`**. Although tar is a bit easier to use, cpio does have a little more functionality. The tar command has the following basic format:

```
tar options files
```

An example might be

```
tar cvf /dev/fd0 /home /etc
```

This example would back up **/home** and **/etc** and write them to the floppy tape device /dev/fd0. The c option says to create an archive, v is verbose mode in which all the files are output to stdout, and f says that tar should output to the following *file*. In this case, you are outputting to the device file /dev/fd0.

If you have a lot of directories, you can use the T option to specify file containing the directories to backup. For example, if you had file called file_list that contained the list of directories, the command might look like this:

```
tar cvfT /dev/rft0 file_list
```

To extract files, the syntax is essentially the same, except that you use the x option to extract. However, you can still use both the f and T options.

The GNU version of **tar** (which comes with most versions of Linux) has a very large number of options. One option I use is z, which I use to either compress or uncompress the archive (depending on which direction I am going). Because the archive is being filtered through gzip, you need to have **gzip** on your system. Although gzip is part of every Linux distribution, it may not be on your system. Also, if you want to copy the archive to another UNIX system, that system may not have gzip. Therefore, you can either skip the compression or use the Z (notice the uppercase) to use compress and uncompress.

Although I can imagine situations in which they might be useful, I have only used a few of them. The best place to look is the tar man-page.

If your backup media can handle more than one set of backups, you can use the mt command to manage your tape drive. Among the functions that the mt can do is to write a "file mark," which is simply a marker on the tape to indicate the end of an archive. To use this function, you must first back the backup to the no-rewind tape device (for example, /dev/rft0). When the drive has written all of the archive to the tape, write the file marker to indicate where the end is.

Normally, when tar is complete and the tape device is closed, it rewinds. When you use the no-rewind device, the tar process finishes, but the tape does not rewind. You can then use the **mt** command to write the file mark at the tapes current location, which is at the end of the tar archive. Even if there are multiple archives on the single tape, mt will find the specific location. Therefore, whenever you need to restore, you can access any of the archives. See the mt man-page for more detail.

11.1.2 Checking the Sanity of Your System

Have you ever tried to do something and it didn't behave the way you expected it to? You read the manual and typed in the example character for a character only to find it didn't work right. Your first assumption is that the manual is wrong, but rather than reporting a bug, you try the command on another machine and to your amazement, it behaves exactly as you

expect. The only logical reason is that your machine has gone insane.

Well, at least that's the attitude I have had on numerous occasions. Although this personification of the system helps relieve stress sometimes, it does little to get to the heart of the problem.If you want, you could check every single file on your system (or at least those related to your problem) and ensure that permissions are correct, the size is right, and that all the support files are there. Although this works in many cases, often figuring out which programs and files are involved is not easy.

Fortunately, help is on the way. Linux provides several useful tools with which you can not only check the sanity of your system but return it to normal. I've already talked about the first set of tools. These are the monitoring tools such as **ps** and **vmstat**. Although these programs cannot correct your problems, they can indicate where problems lie.

If the problem is the result of a corrupt file (either the contents are corrupt or the permissions are wrong), the system monitoring tools cannot help much. However, several tools specifically address different aspects of your system.

Linux provides a utility to compute a checksum on a file, called **sum**. It provides three ways of determining the sum. The first is with no options at all, which reports a 16-bit sum. The next way uses the -r option, which again provides a 16-bit checksum but uses an older method to compute the sum. In my opinion, this method is more reliable because the byte order is important as well. Without the -r, a file containing the word "housecat" would have the same checksum if you changed that single word to "cathouse." Although both words have the exact same bytes, they are in a different order and give a different meaning. Note that I have seen that newer versions of sum default to the -r behavior. If you want the older behavior (and get the same checksum in this example), use the -s option.

On many systems, there is the **md5sum** command. Instead of creating a 16-bit sum, md5sum creates a 128-bit sum. This makes it substantially more difficult to hide the fact that a file has changed.

Because of the importance of the file's checksum, I created a shell script while I was in tech support that would run on a freshly installed system. As it ran, it would store in a database all the information provided in the permissions lists, plus the size of the file (from an ls -l listing), the type of file (using the **file** command), and the checksum (using sum with the -r option). If I was on the phone with a customer and things didn't appear right, I could do a quick **grep** of that file name and get the necessary information. If they didn't match, I knew something was out of whack.

Unfortunately for the customer, much of the information that my script and database provided was something to which they didn't have access. Now, each system administrator could write a similar script and call up that information. However, most administrators do not consider this issue until it's too late.

We now get to the "sanity checker" with which perhaps most people are familiar: **fsck**, the file system checker. Anyone who has lived through a system crash or had the system shut down improperly has seen fsck. One unfamiliar aspect of fsck is the fact that it is actually several programs, one for each of the different file systems. This is done because of the

complexities of analyzing and correcting problems on each file system. As a result of these complexities, very little of the code can be shared. What can be shared is found within the fsck program.

When it runs, **fsck** determines what type of file system you want to check and runs the appropriate command. For example, if you were checking an ext2fs file system, the program that would do the actually checking would be **fsck.ext2** (typically in the **/sbin** directory).

Another very useful sanity checker is the **rpm** package manager (assuming that your system uses the RPM file format) that is the RPM program itself. As I talked about earlier, the rpm program is used to install additional software. However, you can use many more options to test the integrity of your system.

When the system is installed, all of the file information is stored in several files located in **/var/lib/rpm**. These are hashed files that rpm can use but mean very little to us humans. Therefore, I am not going to go into more detail about these files.

Assuming you know what file is causing the problem, you can use rpm to determine the package to which this file belongs. The syntax would be

rpm -q -f <full_path_to_file>

The -q puts **rpm** into query mode and the -f tells it to query the following file and tell me to what package it belongs. Once you know to what package a file belongs, you can verify the package. For example, lets say that you believe that there is something wrong with the **xv** file viewer. Its full path is **/usr/bin/X11R6/xv**, so to find out to what package it belongs, the command would be

rpm -q -f /usr/bin/X11R6/xv

This tells you that xv is part of the package

```
xv-3.10a-3>
```

Now use the -V option to verify the package:

rpm -V xv-3.10a-3

If rpm returns with no response, the package is fine. What if the owner and group are wrong? You would end up with an output that looks like this:

```
..UG.   /usr/bin/X11R6/xv>
```

Each dot represents a particular characteristic of the file. These characteristics are

5	MD5 checksum
S	File size
L	Symbolic link
T	Modification time
D	Device
U	User
G	Group
M	Mode (permissions and file type)

If any of these characteristics are incorrect, rpm will display the appropriate letter.

If you wanted to check all of the packages you could create a script that looks like this:

```
rpm -qa | while read pkg do echo "======== $pkg
================" rpm -V $pkg done
```

The first **rpm** command simply lists all of the package and pipes it into the read, which then loops through each package and verifies it. Since each file will be listed, you should have some seperator between each packages.

11.1.3 Problem Solving

System problems fall into several categories. The first category is difficult to describe and even more difficult to track down. For lack of a better word, I am going to use the word "glitch." Glitches are problems that occur infrequently and under circumstances that are not easily repeated. They can be caused by anything from users with fat fingers to power fluctuations that change the contents of memory.

Next are special circumstances in software that are detected by the CPU while it is in the process of executing a command. I discussed these briefly in the section on kernel internals. These problems are traps, faults, and exceptions, including such things as page faults. Many of these events are normal parts of system operation and are therefore expected. Other events, like following an invalid pointer, are unexpected and will usually cause the process to terminate.

Kernel Panics

What if the kernel causes a trap, fault, or exception? As I mentioned in the section on kernel internals, there are only a few cases when the kernel is allowed to do this. If this is not one of those cases, the situation is deemed so serious that the kernel must stop the system immediately to prevent any further damage. This is a panic.

When the system panics, using its last dying breath, the kernel runs a special routine that prints the contents of the internal registers onto the console. Despite the way it sounds, if your system is going to go down, this is the best way to do it. The rationale behind that statement is that when the system panics in this manner, at least there is a record of what happened.

If the power goes out on the system, it is not really a system problem, in the sense that it was caused by an outside influence, similar to someone pulling the plug or flipping the circuit breaker which my father-in-law did to me once. Although this kind of problem can be remedied with a UPS, the first time the system goes down before the UPS is installed can make you question the stability of your system. There is no record of what happened and unless you know the cause was a power outage, it could have been anything.

Another annoying situation is when the system just "hangs." That is, it stops completely and does not react to any input. This could be the result of a bad hard disk controller, bad RAM, or an improperly written or corrupt device driver. Because there is no record of what was happening, trying to figure out what went wrong is extremely difficult, especially if this happens sporadically.

Because a system panic is really the only time you can easily track down the problem, I will start there. The first thing to think about is that as the system goes down, it does two things: writes the registers to the console screen and writes a memory image to the dump device. The fact that it does this as it’s dying makes me think that this is something important, which it is.

The first thing to look at is the instruction pointer. This is actually composed of two registers: the CS code segment and EIP instruction pointer registers. This is the instruction that the kernel was executing at the time of the panic. By comparing the EIP of several different panics, you can make some assumptions about the problem. For example, if the EIP is consistent across several different panics, this indicates that there is a software problem. The assumption is made because the system was executing the same piece of code every time it panicked. This *usually* indicates a software problem.

On the other hand, if the EIP consistently changes, then this indicates that probably no one piece of code is the problem and it is therefore a hardware problem. This could be bad RAM or something else. Keep in mind, however, that a hardware problem could cause repeated EIP values, so this is not a hard -coded rule.

The problem with this approach is that the kernel is generally loaded the same way all the time. That is, unless you change something, it will occupy the same area of memory. Therefore, it’s possible that bad RAM makes it look as though there is a bad driver. The way to verify this is to change where the kernel is physically loaded. You can do this by rearranging the order of your memory chips.

Keep in mind that this technique probably may not tell you what SIMM is bad, but only indicate that you may have a bad SIMM. The only sure-fire test is to swap out the memory. If the problem goes away with new RAM and returns with the old RAM, you have a bad SIMM.

Getting to the Heart of the Problem

Okay, so we know what types of problems can occur. How do we correct them? If you have a contract with a consultant, this might be part of that contract. Take a look at it and read it. Sometimes the consultant is not even aware of what is in his or her own contract. I have talked to customers who have had consultant charge them for maintenance or repair of hardware, insisting that it was an extra service. However, the customer could whip out the contract and show the contractor that these services were included.

If you are not fortunate to have such an expensive support contract, you will obviously have to do the detective work yourself. If the printer catches fire, it is pretty obvious where the problem is. However, if the printer just stops working, figuring out what is wrong is often difficult. Well, I like to think of problem solving the way Sherlock Holmes described it in *The Seven Percent Solution* and maybe other places:

"Eliminate the impossible and whatever is left over, no matter how improbable, must be the truth."

Although this sounds like a basic enough statement, it is often difficult to know where to begin to eliminate things. In simple cases, you can begin by eliminating almost everything. For example, suppose your system was hanging every time you used the tape drive. It would be safe at this point to eliminate everything but the tape drive. So, the next big question is whether it is hardware problem or not.

Potentially, that portion of the kernel containing the tape driver was corrupt. In this case, simply rebuilding the kernel is enough to correct the problem. Therefore, when you relink, link in a new copy of the driver. If that is not sufficient, then restoring the driver from the distribution media is the next step. However, based on your situation, checking the hardware might be easier, depending on your access to the media.

If this tape drive requires its own controller and you have access to another controller or tape drive, you can swap components to see whether the behavior changes. However, just as you don’t want to install multiple pieces of hardware at the same time, you don’t want to swap multiple pieces. If you do and the problem goes away, how do you know whether it was the controller or the tape drive? If you swap out the tape drive and the problem goes away, that would indicate that the problem was in the tape drive. However, does the first controller work with a different tape drive? You may have two problems at once.

If you don’t have access to other equipment that you can swap, there is little that you can do other than verify that it is not a software problem. I have had at least one case while in tech support in which a customer called in, insisting that our driver was broken because he couldn’t access the tape drive. Because the tape drive worked under DOS and the tape drive was listed as supported, either the documentation was wrong or something else was. Relinking the kernel and replacing the driver had no effect. We checked the hardware settings to make sure there were no conflicts, but everything looked fine.

Well, we had been testing it using tar the whole time because tar is quick and easy when you are trying to do tests. When we ran a quick test using **`cpio`**, the tape drive worked like a champ. When we tried outputting tar to a file, it failed. Once we replaced the tar binary, everything worked correctly.

If the software behaves correctly, there is potential for conflicts. This only occurs when you add something to the system. If you have been running for some time and suddenly the tape drive stops working, then it is unlikely that there are conflicts; unless, of course, you just added some other piece of hardware. If problems arise after you add hardware, remove it from the kernel and see whether the problem goes away. If it doesn't go away, remove the hardware physically from the system.

Another issue that people often forget is cabling. It has happened to me a number of times when I had a new piece of hardware and after relinking and rebooting, something else didn't work. After removing it again, the other piece still didn't work. What happened? When I added the hardware, I loosened the cable on the other piece. Needless to say, pushing the cable back in fixed my problem.

I have also seen cases in which the cable itself is bad. One support engineer reported a case to me in which just pin 8 on a serial cable was bad. Depending on what was being done, the cable might work. Needless to say, this problem was not easy to track down.

Potentially, the connector on the cable is bad. If you have something like SCSI, on which you can change the order on the SCSI cable without much hassle, this is a good test. If you switch hardware and the problem moves from one device to the other, this could indicate one of two things: either the termination or the connector is bad.

If you do have a hardware problem, often times it is the result of a conflict. If your system has been running for a while and you just added something, it is fairly obvious what is causing the conflict. If you have trouble installing, it is not always as clear. In such cases, the best thing is to remove everything from your system that is not needed for the install. In other words, strip your machine to the "bare bones" and see how far you get. Then add one piece at a time so that once the problem re-occurs, you know you have the right piece.

As you try to track down the problem yourself, examine the problem carefully. Can you tell whether there is a pattern to when and/or where the problem occurs? Is the problem related to a particular piece of hardware? Is it related to a particular software package? Is it related to the load that is on the system? Is it related to the length of time the system has been up? Even if you cant tell what the pattern means, the support representative probably has one or more pieces of information to help track down the problem. Did you just add a new piece of hardware or SW? Does removing it correct the problem? Did you check to see whether there are any hardware conflicts such as base address, interrupt vectors, and DMA channels?

I have talked to customers who were having trouble with one particular command. They insist that it does not work correctly and therefore there is a bug in either the software or the doc. Because they were reporting a bug, we allowed them to speak with a support engineer even though they did not have the valid support contract. They kept saying that the documentation is bad because the software did not work the way it was described in the manual. After pulling some teeth, I discovered that the doc the customers used is for a product that was several years old. In fact, there had been three releases since then. They were using the latest software, but the doc was from the older release. No wonder the doc didn't match the software.

Collection information

Instead of a simple list, I suggest you create a mind map. Your brain works in a non-linear fashion, and unlike a simply list a mind map, helps you gather and analyse information the way your brain actaully works.

Work methodically and stay on track

Unless you have a very specific reason, don't jump to some other area before you complete the one you are working on. It is often a waste of time, not because that other area is not where the problem is, but rather "finding yourself" again in the original test area almost always requires a little bit of extra time "Now where was I?". Let your rest results in one area guide you to other areas *even if* that means jumping somewhere else before you are done. But make sure you have a reason.

Split the problem in pieces

Think of a chain that has a broken link. You can tie the end onto something, but when you pull nothing happens. Each link needs to be examined invidually. Also, the larger the pieces, the easier it is to overlook something.

Keep track of where you have been

"Been there done that." Keep a record of what you have done/tested and what the results where. This can save a lot of time whith complex problems with many different components.

Listen to the facts

One key concept I think you need to keep in mind is that appearances can be deceiving. The way the problem presents itself on the surface, may not the real problem at all. Especially when dealing with complex systems like Linux or networking, the problem may be buried under several different layers of "noise". Therefore, you should try not make too many assumptions and if you do, verify those assumptions before you go wandering off on the wrong path. Generally, if you can figure out the true nature of the problem then then finding the cause is usually very easy.

Be Aware of all limitation and restrictions

Maybe what you are trying to do is not possible given the current configuration or hardware. For example, maybe there is a firewall rule which prevents two machines from communicating. Maybe you are not authorized to use resources on a specific machines. You might be able to see machine using some tools e.g. ping but not with others e.g. traceroute.

Read what is in front of you

Pay particular attention to error messages. I have had "experienced" system administrators reports problems to me and say that there was "some error message" on the screen. It's true that many errors are vague or come from the last link in the chain, but more often than not they provide valuable information. This also applies to the output of commands. Does the command report the information you expect it to?

Keep calm

Getting upset or angry will not help you solve the problem. In fact, just the opposite is true. You begin to be more concerned with your frustration or anger and forget about the true problem. Keep in mind that if the hardware or software is as buggy as you now think it is, the company would be out of business. Obviously that statement does not apply to Microsoft products It's probably one small point in the doc that you skipped over if you even read the doc or something else in the system is conflicting. Getting upset does

nothing for you. In fact speaking from experience, getting upset can cause you to miss some of the details for which you're looking.

Recreate the problem

As in many branches of science, you cause something to happen and then examine both the cause and results. This not only verifies your understanding of the situation, it also helps prevent wild gooses chases. Users with little or no technical experience tend to over dramatize problems. This often results in in comments like "I didn't do anything. It just stopped working." By recreating the problem yourself, you have ensured that the problem does not exist between the chair and the keyboard.

Stick with known tools

There are dozens if not hundreds of network tools available. The time to learn about their features is not necessarily when you are trying to solve a business critical problem. Find out what tools are already available and learn how to use them. I would also recommend using the tools that are available on all machines or at least as many as possible. That way you don't need to spend time learing the specifics of each tool.

Don't forget the obvious

Cables can accidently get kicked out or damaged. I have seen cases where the cleaning crew turned off a monitor and the next day the user reported the computer didn't work because the screen was blank.

11.1.4 Crash Recovery

If your system crashes, the most important thing is to prevent further damage. Hopefully, the messages that you see on your screen will give you some indication of what the problem is so that you can correct it. (For example, timeout errors on your hard disk might mean it's time to buy a new hard disk.)

Emergency Floppies

It may happen that the system crash you just experienced no longer allows you to boot your system. What then? The easiest solution (at least the easiest in terms of figuring out what to do) is reinstalling. If you have a recent backup and your tape drive is fairly fast, this is a valid alternative, provided there is no hardware problem causing the crash.

In an article I once wrote, I compared a system crash to an earthquake. The people who did well after the 1989 earthquake in Santa Cruz, Ca were those who were most prepared. The people who do well after a system crash are also those who are best prepared. As with an earthquake, the first few minutes after a system crash are crucial. The steps you take can make the difference between a quick, easy recovery and a forced re-install.

In previous sections, I talked about the different kinds of problems that can happen on your system, so there is no need to go over them again here. Instead, I will concentrate on the steps to take after you reboot your system and find that something is wrong. It's possible that when you reboot all is well and it will be another six months before that exact same set of circumstances occurs. On the other hand, your screen may be full of messages as it tries to bring itself up again.

Because of the urgent nature of system crashes and the potential loss of income, I decided that this was one troubleshooting topic through which I would hold your hand. There is a set of common problems that occur after a system crashes that need to be addressed. Although the cause of the crash can be a wide range of different events, the result of the crash is small by comparison. With this and the importance of getting your system running again in mind, I am going to forget what I said about giving you cookbook answers to specific questions for this one instance.

Lets first talk about those cases in which you can no longer boot at all. Think back to our discussion of starting and stopping the system and consider the steps the system goes through when it boot. I talked about them in detail before, so I will only review them here as necessary to describe the problems.

As I mentioned, when you turn on a computer, the first thing is the Power-On Self-Test, or POST. If something is amiss during the POST, you will usually hear a series of beeps. Hopefully, there will be some indication on your monitor of what the problem is. It can be anything from incorrect entries in your CMOS to bad RAM. If not, maybe the hardware documentation mentions something about what the beeps mean.

When finished with the POST, the computer executes code that looks for a device from which it can boot. On a Linux system, this boot device will more than likely be the hard disk. However, it could also be a floppy or even a CD-ROM. The built-in code finds the active partition on the hard disk and begins to execute the code at the beginning of the disk. What happens if the computer cannot find a drive from which to boot depends on your hardware. Often a message will indicate that there is no bootable floppy in drive A. It is also possible that the system will simply hang.

If your hard disk is installed and it *should* contain valid data, your master boot block is potentially corrupt. If you created the boot/root floppy set as I told you to do, you can use fdisk from it to recreate the partition table using the values from your notebook. Load the system from your boot/root floppy set and run fdisk.

This is done from the hard disk. With the floppy in the drive, you boot your system. When you get to the Boot: prompt, simply press Enter. After loading the kernel, it prompts you to insert the root file system floppy. Do that and press Enter. A short time later, you are brought to a # prompt from which you can begin to issue commands.

When you run fdisk, you will probably see an empty table. Because you made a copy of your partition table in your notebook as I told you to do, simply fill in the values exactly as they were before. Be sure that you make the partition active as it was previously. Otherwise, you won't be able to boot, or you could still boot but you will corrupt your file system. When you exit fdisk, it will write out a copy of the master boot block to the beginning of the disk. When you reboot, things will be back to normal.

(I've talked to at least one customer who literally laughed at me when I told him to do this. He insisted that it wouldn't work and that I didn't know what I was talking about. Fortunately for me, each time I suggested it, it did work. However, I *have* worked on many machines where it didn't work. With a success rate of more than 90 percent, it's obviously worth a try.)

Table - Files Used in Problem Solving

Command	Description
/bin/pstat	Reports system information
/bin/who	Lists who is on the system
/bin/whodo	Determines what process each user is running
/etc/badblock	Checks for bad spots on your hard disk
/bin/rpm	Displays information about install packages (also used to install and remove software)
/etc/df	Calculates available disk space on all mounted file
/etc/fdisk	Creates and administers disk partitions
/etc/fsck	Checks and cleans file systems
/etc/fuser	Indicates which users are using particular files and file systems
/etc/ifconfig	Configures network interface parameters
/etc/ps	Reports information on all processes
/usr/adm/hwconfig	probe for hardware
/var/log/wtmp	Login records
/usr/bin/cpio	Creates archives of files
/usr/bin/last	Indicates last logins of users and teletypes
/usr/bin/lpstat	Prints information about status of print service
/usr/bin/netstat	Administers network interfaces
/usr/bin/sar	Reports on system activity
/usr/bin/tar	Creates archives of files
/usr/bin/w	Reports who is on the system and what they are doing

11.1.5 Hardware Diagnostic Tools

Since the world is not perfect you will eventually have to deal with a crashed system. In many cases, how the system behaves when it boots (or doesn't boot) will give you an indication of what is going on. However, it also will happen that there is nothing that specifically identifies the problem. It is also possible that your system boots fine, but exhibits odd behavior as it is running. The most common solution for this kind of problems on Windows machines is to re-install. However, this only corrects the problem if is related to the software. What about hardware problems?

There are a number of hardware diagnostic tools on the market. Some run under Windows, whereas others have their own "operating system" which you can boot, allowing you to directly access the hardware. Those that run as stand alone products, typically have a much wider range of tests they can conduct because they are not limited by the operating system. Keep in mind that this more that just reporting the IRQ or base address of the devices. These products actually test the various components of your system.

Personally, I think you should use tools which run under the operating system in conjunction with stand-alone products. It is possible that you might get incorrect results if you are running under any operating system as it often "interprets" the information for you. Although this is useful for "configuration" issues, defects and other problems are often missed.

There are also a few products that come with interface cards that are inserted to the bus, allowing you to diagnostic problems even when your system cannot boot. These have a small, digital display on the card which shows you the post code being sent across the bus. Based on the code, you can determine where the problem lies.

In general, the software products have a common set of tests they run through. The tests normally include:

- System Board
- Video Alignment Aids
- Video Adapter
- Parallel Port
- Serial Port
- Floppy Disk Drive
- Hard Disk Tests (0 & 1)
- Main Memory Tests

One of the key features to look at is the extent to which you can configure these tests. This might mean defining a specific set of tests to run, as well as how many times to run each test. Both are important aspects. If you already have an idea of where the problem is, you should not have to wait for the program to run through unnecessary tests. Also, with hardware you often have sporadic problems. Therefore, you might have to run the test continually for an extended length of time before the problem re-appears.

Another thing to look at is what values or configuration settings can be changed. Keep in mind that changing settings is not always a good thing. Particularly if a novice is running the tests.

TuffTEST

TuffTEST from Windsor Technologies is a powerful and very inexpensive stand-alone diagnostic tool. Although you could order it with all of the packaging, you can save time, money and trees by ordering and then downloading it from the web. As of this writing it is just $9.95, which is a fraction of most other products.

One key aspect is that it is designed specifically for user with less experience. Although it has most of the features of high-end tools, the emphasis is on ease of use, as well as providing the user with sufficient information to diagnose the problem.

This is a stand-alone product, in that it can be booted from a floppy. This sounds confusing at first, because you download it from the Internet. What you download is a program which allows you to create the bootable floppies. Once booted, TuffTEST "takes over" the computer, without the need for an operating system like DOS or Windows. As I mentioned before, often this yields more accurate results. TuffTEST has its own set of device drivers, which can access hardware directly.

Windsor boasts that TuffTEST is "safe for use by anyone." This is because none of the tests change data on the hard disk. In addition, the program is so configured that once it boots, it will wait 10 seconds for a menu selection and if no key is pressed it runs through the complete suite of tests.

Another advantage of TuffTEST is that it is complete written in assembly language which means more compact code, and faster execution. In addition, it take up just 125K of memory, which is actually relocated when then program runs. This ensures that every memory location is tested. In other cases, the program is actually too large to be able to check *all* of memory.

TuffTEST is not just a diagnostic tool as it can also display all of your hardware configuration information. This information can then be printed or saved to the disk. Each saved session contains the test results as well as the system configuration. Since you can save up to five previous sessions, you can compare the results from multiple tests.

Higher up on the scale is TuffTEST PRO, this is intended for the professional. This has the same basic functionality plus you can edit your configuration and make other changes to your system. Like TuffTEST. TuffTEST PRO is a stand-alone product, meaning you boot your operating system from the diskette and it becomes your operating system.

In addition, there are a number of tests that TuffTEST PRO has that are not included in TuffTEST. For example, TuffTEST PRO can report the switch positions on your motherboard, conduct I/O tests on your serial and parallel ports, determine the optimal interleave and low-level format your harddisk, and many other tests. Using the optional loopback test, you can do I/O tests on your serial and parallel ports.

One of the most interesting aspects of TuffTEST is sales approach. You can order a packaged version of the product, including a printed manual , if you feel it is necessary. However, there really is no need. The on-line manual contains all of the necessary information, plus the product is extremely intuitive.

Lifetime support is provided for *free.* However, the product is so easy to use it is hard to think of a reason why you would need to call them. In addition, updates range from free for minor changes to a slight fee for major new releases.

Micro2000

If you are concerned with diagnosing PC hardware problems, take a look at the wide range of products that Micro2000 has to offer. The products range from self-booting diagnostic tools to POST reader cards to remote diagnostics and beyond.

Micro-Scope is their self-booting diagnostic tool that can run on any PC. Regardless of the CPU manufacturer (Intel, Cyrix or AMD) or bus (ISA, EISA, MCA, PCI, and PCMCIA), Micro-Scope can identify problems on your PC. Version 7 (the newest, as of this writing) contains tests for your CD-ROM drive, without the need to load DOS-based CD-ROM drivers. Something which many other diagnostic tools do not have. In addition, the version 7 also contains support for the AMD K6-II and Intel Xeon processor, even those with a clock speed above 500Mhz. Upgrades for new processors are available for download from the internet.

Many tools simply report on the problems they find. However, Micro-Scope not only allows you to make changes, but also gives you detailed benchmarks of your system. This is useful when you "feel" something is wrong with your machine, but there is no identifiable hardware problem. With the report generated by the benchmark, you can see if the machine is performing as it should.

During the testing, Micro-Scope examines the CMOS and POST information. Anything that is inaccurate or questionable is flagged, allowing you to change it as needed. Part of this is being able to accurately identify your hardware, including brand name and model. This is extremely useful when buying brand name computers, which normally do not tell you exactly what components you have.

Microscope supports all common bus types including ISA, EISA, PCI and MCA. You can even display the POS registers on IBM PS/2 systems, including all slots, which adapters are in which slot, which ADF (adapter description file) to use and whether the ADF is loaded.

In addition to being able to diagnose CD-ROM problems, Micro-Scope can test many other multi-media components, such as DVD drives and sound cards. It has full synthesizer tests and can test the volume and left-right channels of your sound card.

Tests can be run once or repeatedly. The results of which can either be printed out or saved to disk (or just viewed on-screen if you want). In addition, you can use the printscreen capability to print directly from the application.

As with other products, Micro-Scope will thoroughly check your memory, using all of the common tests (checkerboard, walking-ones, etc.). Low memory is tested before the entire program is loaded, which is then relocated in memory to enable you to test all of your memory, regardless of how much you have. In addition, Micro-Scope will tell you exactly what bank is failing. This includes the ability to test internal and external system cache, as well as video RAM up to 64Mb.

Another bonus is the tools Micro-Scope has for data recovery. It can identify and correct many problems in the master boot record of your hard disk. It also includes an editor to allow you to make changes yourself anywhere on the disk (assuming you have the knowledge to do it).

In addition, to free download of patches, Micro-Scope comes with lifetime technical support. After using the program, I find it difficult to conceive of a reason why someone would need to call to support, as it is so intuitive, but the offer is nice. The product package contains both 3.5" and 5.25" disks, a uses manual, as well as 9 pin serial, 25 pin serial, and 25 pin parallel loopback connectors, to diagnose serial and parallel port problems.

Unfortunately, something like Micro-Scope cannot always do the job. This happens when your system just won't boot for any number of reasons. Using a diskette with its own operating system does no good, because the computer does not get that far to boot from anywhere. This is where Micro2000's product POST-Probe comes in handy.

As its name implies, POST-Probe monitors the POST codes being sent across your system bus as the computer is booting. It can fit into any ISA, EISA, PCU or MCA slot (although it requires the included adapter for the MCA). These codes are displaed on two seven-segment displays, indicating what the POST is testing at the moment. There are also four LEDs which monitor to the power, as well as four voltage pads (+5vdc, -5vdc, +12vdc, -12vdc and an additional 3.3V for PCI) to test the system using a voltmeter.

There is an additional LED which monitors clock signals, one for the RESET signal, and one for I/O reads and writes. You can therefore use POST-Probe after your system is running to identify other bus problems and possible problems with specific cards.

When your system stops, the last code displayed gives you an indication of what is wrong. Although the code does not always tell you the exact place where there is a problem, the included users manual lists each phase of the POST. By looking at the steps around where it stopped, I have never not found the problem. In one instance, I accidentally loosed up the cable to my hard disk. When I tried to boot, nothing happened. Using the POST-Probe I quickly found the problem.

As I will talk about in later chapters, I am a stickler for documentation. I am really impressed with the POST-Probe manual. It is written in an easy to understand language. POST failure codes are on the left side of each page, with the description of the device or chip that is causing the problem. This helps finding and understanding the problem.

For the true profession, Micro200 has combined Micro-Scope and POST-Probe into a single product, which they call the Universal Diagnostics Toolkit. Both products are combined in the full version within a case, which is not only large enough to hold both products, but tools and

many other things. Each product as the same lifetime technical support as the stand-alone versions.

Micro2000's product Burn-In takes much of the functionality of Micro-Scope to the next level. As its name implies, it is used to conduct "burn-in" tests of computers. This can be either new machines or ones that you have repaired. This is an extremely useful tool to prevent deploying products that will only cause you problems down the road. Particularly in cases where machines have multiple problems and only one is apparent, burn-in tests can save you a great deal of both time and money.

Like Micro-Scope, Burn-In is compatible with all CPU manufacturers and system buses. In addition, Burn-In performs all of the same tests that Micro-scope does.

Burn-In has a couple of very useful features for companies that install a larger number of PCs at once. First, the tests can be run without a monitor or keyboard. Therefore, you need a lot less space allowing you to simply stack up the PCs and run a large number of tests at once. Using the floppy drive light and speaker, the program send a few signals to the technician when it needs a "scratch" disk or the loopback plugs. Other than that, the program runs completely on its own, saving the results to disk.

As the tests are run, Burn-In writes a complete log to the scratch disk you provided. Since the log is ASCII, you can read it with any text editor. In addition, the log is being update the entire time. Therefore, if something should happen to the machine (like someone accidentally pulling the plug), Burn-In will be able to continue where it left off.

In addition, you only need to run the setup once. The test configuration is then saved and performed the same way each time the disk is booted. If the program determines that hardware is not present for a test is was selected to do, that test is simply skipped, without the need to configure the test for different hardware variations.

11.1.6 Netiquette

If you ever find yourself in a situation where you cannot solve the problem yourself and need to access any of the various resources on the Internet, there are a few ground rules that you need to adhere to. These are called "Netiquette" (net-etiquette). For a detailed discussion on netiquette, I would suggest the Netiquette Home Page. This is essentially the online version of the book Netiquette by Virginia Shea.

There are, however, a few important points that I would like to make. First, do your own research. In the sections on Linux documentation and Other linux resources we discussed many different places you can look for answers to your questions. You should always try these first before you go to a newsgroup or mailing list. Most of the time you will find the answer yourself. This is much more beneficial to your understanding of Linux than simply having someone spoon feed you your answer. Also, if you do not find your answer, or the answer is hard to understand, telling others where you look and what you found, keeps others from giving you the exact same answer, thereby saving time for everyone.

Second, list not only what you are trying to do, but how you *expect* it to behave. Don't simply say "it doesn't work right." While working in tech support, I have numerous calls from customers who assumed that something was not working correctly, when it was. They had incorrect ssumptions about the way things were supposed to work. Since their assumptions were wrong, the expected behaviour was not what was "correct".

11.2 Getting Help

If you're like me, you think the manual is for cowards. Any good computer hacker should not be afraid to open up the box and start feeding in disks without any regard for the consequences. You tear open the box, yank out the floppies, pop the first one in the drive, and start up the Software Manger and happily go through the thankless task of installing the software. After everything has been installed and your desktop icons have been created, you double-click the icon and start your new multimedia Web viewer. But wait! It doesn't work right. No matter how much you point and click and click again, nothing happens. In frustration, you get on the phone and frantically dial the 800-number from the back of the manual (making this the first time you opened it).

When you finally get through to support after waiting for two hours (it was actually only five minutes), you lash out at the poor tech support representative who was unlucky enough to get your call. You spend more time ranting about poorly written software than you spent on hold. When you finally finish insulting this persons ancestry, he or she calmly points out that on page 2 in the manual, where it describes the installation procedure, it says that to get the Web to work correctly, you have to have a network installed. Because you decided not to install TCP/IP when you first loaded the system, there is no way for the Web browser to work. You're embarrassed and the whole situation is not a happy thing.

Some people might want to say that things are not the same with Linux support as you don't have the same kind of support as with "commercial" products. Well, many Linux vendors offer 30 day installation support, as well as have fee-based support afterwards. So the issues are the same. Furthermore, just because you are posting to a "free" mailing list instead of calling support. There are a a number of things to do before you go looking for help.

One important aspect of solving problems yourself is actually trying to solve the problem yourself. This might seem too obvious to mention, but what really is not obvious is the extent to which you really should go to before turning to others for help. This is not because others are annoyed with "stupid questions", but rather the time is better spent working on the hard problems and not the ones that people **should** be able to solve on their own.

A way to do that is to read the HOWTOs and other documentation before, during, and after the installation. Doing so tends to limit the embarrassing calls to tech support or posts to newsgroups, but the world is not perfect and eventually something will go wrong. Programs are (still) written by human beings who can make mistakes, which we users call bugs. Perhaps the QA technician who was checking your SCSI host adapter sneezed at the very moment the monitor program reported an incorrect voltage. Maybe the manufacturer never tested that one, rare set of circumstances that causes the program to freeze the way it did on your machine. The end result is that you've read the manual, checked and rechecked the hardware, and it still

does not behave the way it is supposed to. You can't solve the problem yourself, so you need help.

If you ever find yourself in a situation where you cannot solve the problem yourself and need to access any of the various resources on the Internet, there are a few ground rules that you need to adhere to. These are called "Netiquette" (net-etiquette). For a detailed discussion on netiquette, I would suggest the Netiquette Home Page. This is essentially the online version of the book Netiquette by Virginia Shea.

There are, however, a few important points that I would like to make. First, do your own research. In the sections on Linux documentation and Other linux resources we discussed many different places you can look for answers to your questions. You should always try these first before you go to a newsgroup or mailing list. Most of the time you will find the answer yourself. This is much more beneficial to your understanding of Linux than simply having someone spoon feed you your answer. Also, if you do not find your answer, or the answer is hard to understand, telling others where you look and what you found, keeps others from giving you the exact same answer, thereby saving time for everyone.

What problems a person should be able to solve on their own is not as easy to define as you might think. For example, I have read posts to mailing lists describing problems and quote the error message they are getting. The user has never used the program before and so the error is pretty meaningless (other than saying "something" is misconfigured or not found).It might be reasonable to post such a question to news group. However, I remember one case where the person posted the problem for the third time and no one had answered. He was indignant that no one had a solution for him. Well, I took the exact error message he was getting and did a search on Google for it. The very first post was someone else who posted months before and the reply was very specific steps to solve the problem. Had this person searched google before the first post, they would have had a solution already.

I cannot tell you enough that Google is your friend. It might take a little practice on phrasing your search text properly to get the answer you are looking for. However, it is worth it. First, you may not always get the specific answer you want, but you find a lot of cool and useful sites. I cannot begin to count the sites I have stored in my booksmarks that I stumbled across when looking for something else. Second, there is a **great** chance you **will** find something that will help, particularly if you have a specific error message. Third, it's a great way of finding information, whether you have a problem or just want to find more information.

If you don't find an answer on Google or any other search engine, try searching an online forum for the answer. On linuxnewbie.org is a great forum with tens of thousands of posts. They have a search mechanism to dig through old posts. Maybe there is something there that will help. If if is a common problem, you will probably find an answer. If it is uncommon, then at least you have tried all the common solutions.

When you finally do post, list not only what you are trying to do, but how you *expect* it to behave. Don't simply say "it doesn't work right." While working in tech support, I have numerous calls from customers who assumed that something was not working correctly, when it actually was. They had incorrect assumptions about the way things were "supposed" to work. Since their assumptions were wrong, the expected behaviour was not what was

"correct".

Also, make sure you tell people what you have already tried and what the results were, *even if* what you tried couldn't be correct. For example, I had trouble connecting my DSL card because the instructions from my ISP talked about an DSL modem and not an expansion card. Plus, I already had an existing ISDN connection and a switch-box to which all of my phones connected. When I posted to a mailing list, I listed the various combinations and locations of the cables and various boxes, indicating why I though each one was incorrect. This got me more detailed answers than in I simply asked "How do I connect my card?". Less than 20 minutes later I was on-line.

Whether you post to a newsgroup, mailing list, forum or to the owner of sites like mine, remember that no one ***owes*** you an answer. We will do our best to help, but sometimes we cannot. Maybe we don't have any experience with a particular issue or maybe we just don't have the time at the moment. Also, people will often simply ignore messages when the answer is listed in a HOWTO, the man-page or some other obvious place.

You may end up with someone who simply replies "man whatever", where "whatever" is some command. Don't be annoyed at terse answers like that. If you already looked at that man-page, then you should have already said so in your post. If not, then there is probably something in that man-page which will help. Don't expect people to hand you the answer on a silver platter, especially if you don't do any work yourself.

If the reponse does not answer your question, then be polite when telling them, *even if* you feel that the response would "obviously" not answer your question. Sometimes people respond based on pre-conceptions that you do not have. I find this to especially be the case when someone does not provide details of the problem, like error messages, things that were changed recently, and so on.

Another very polite (and useful) thing is to thank people for their efforts, even if they did not solve your problem. If you get a reputation for someone who simply asks the questions and never responds to the solutions, you may find that people stop answering you. Also, if the reponse solves your problem, reply back to the newsgroup or mailing list and add (SOLVED) (or something similar) to the end of the subject. That way other people will know that the issue has been solved.

Also, when you reply, you should reply to the group and not to the individual and certainly not to both the group **and** individual. Since the person replied to your post, there is no need to reply to both as he or she will certainly see your message. If you reply just to the person, then others in the group do not have the benefit of knowing if the solution worked or any other information that was exchanged.

Depending on what program you use when replying to a mailing list or news group, you need to pay attention to the default behaviour when you simply click "Reply". Some reply to the group and some reply to the sender. I have been convinced that the proper behaviour is to reply just to the individual. This is because it is less dangerous to make a mistake and reply to a single person when you wanted to reply to the group, as compared to replying to the group when you ment to only reply to the individual. Either way, you need to be aware of the

behaviour of your mail or news reader.

When starting a *new* subject (also called a "thread"), do **not** simply reply to an existing message. The advanced mail and news readers that come with Linux keep track of the messages and can display them hierarchically, based on who replied to whom. These are called "threaded" readers as they properly manage the various threads. They typically allow you to collapse or expand individual threads, giving you a much better overview than if each message was listed individually. If you start a new topic by replying to an existing message, you defeat the whole purpose of threaded readers and may end up having your question go unaswered because people simply do not see it.

11.2.1 Calling Support

The most commonly known source of help is the company's tech support department. With Linux, however, there is no one to call. At least, there is no one single company or organization to call. If you bought Linux as part of a complete system or from an established distributor, they will probably be able to help you. Many have free e-mail support for those who can't afford the phone support. Many also will charge you a certain amount per minute or a flat fee per call. However, even if you are left with other sources, such as USENET newsgroups or the mailing lists, going through "tech support" has the same basic principles, so I will address this as though you were getting help directly from the company.

Tech support is like any system. You put garbage in and you're likely to get garbage out. Calling in and demanding immediate results or blaming the support representative for the problem will probably get you one of a few responses. They'll tell you that it's a hardware problem if you've called a software company, a software problem if you've called a hardware company, or they'll say there is "something" else on the machine conflicting with their product, but it's your job to figure it out. You may even get an answer that, yes, that board is bad, and you can return it to the place of purchase to get a refund or exchange. In other words, they blow you off.

If the board was bad, getting a replacement solves the problem. If there is a conflict, however, you will probably spend even more time trying to track it down. If the problem is caused by some program problem (conflicts or whatever), reinstalling may not fix the problem.

Rather than spending hours trying to track down the conflict or swapping out boards, you decide to call tech support. The question is, "Which one?" If there is only one program or one board, it's pretty obvious which tech support department to call. If the problem starts immediately after you add a board or software package, the odds are that this has something to do with whatever you just installed. If, however, the problem starts after you've been running for a while, tracking down the offender is not that easy. Thats why you're going to call tech support, right? So grab that phone and start dialing.

Stop! Put that phone down! You're not ready to call yet. There's something you need to do first. In fact, you need to do several things before you call.

Calling tech support is not as easy as picking up the phone and dialing. Many people who are having trouble with their systems tend to think that it is. In many cases, this is true. The problem is basic enough that the tech support representative can answer it within a few minutes. However, if it's not, your lack of preparation can turn a two-minute call into a two-hour call.

Preparation for calling tech support begins long before that first phone call or the first news post. In fact, preparation actually begins before you install anything on your system. I mean *anything* before you install your first program, before you make the change to .profile to change your prompt, even before you install the operating system.

In previous chapters, I talked about purchasing a notebook and detailing your system configuration. This kind of information is especially important when you call a hardware vendor to help track down a conflict or when the software *should* work. You may never use most of this information. When you do need it, however, you save yourself a great deal of time by having it in front of you. This is also important when you post a message to a newsgroup and someone asks for the details of your configuration.

By knowing what product and what release you have before you call, you save yourself time when you do call. First, you don't have to hunt through notes or manuals while the clock is ticking on your phone bill. Even if you can't find the release, don't guess or say "the latest." Though you can get the release number from the installation media, this may not be exactly what was used to install. The best source is to run uname -a. This tells you exactly what release the system is currently running.

The problem with Linux is that there is no one "official" release. There are "official" releases of the kernel, but that doesn't necessarily tell you everything about your system. If you purchase a complete Linux system (either just the software or with hardware), you should have some documentation that not only lists the kernel version but also that distributors version as well.

If you guess, the support technical might have to guess, too. This is important because fixes are almost always release-specific. If you say "the latest" and it isn't and the "bug" you have was corrected in the latest release, the analyst is not going to give you the fix because he or she thinks you already have it. This wastes the analysts time, wastes your time, and in the end, you don't get the correct solution. More than likely, if you guess and say "the latest" when posting to a newsgroup, you will get some "friendly" reminders that you should provide more specific details.

Should it be necessary to contact a support organization, at the very minimum, you should have the following information:

- Operating system(s) and versions
- Machine type: 486/DX 50, P6/133, etc.
- Make and model of all hardware (rarely is just the brand sufficient)
- Controller make, model, and type
- Symptoms of problem: noises, messages, previous problems
- An *exact* description of the error message you received and the context in which you

received it
- Drive organization: partition sizes, special drivers
- Special devices/drivers, such as disk array hardware and software
- What the machine was doing when the problem occurred
- What the sequence of events was that preceded the problem
- Whether this problem has occurred more than once and if so, how often
- Whether you recently installed any device drivers or additional hardware
- What the last thing you changed was
- When you changed it
- Whether this a production machine and whether you are down now
- If this is related to a software product you have installed, what the exact version is
- What distribution of Linux your are running, whether and from where you downloaded it, copied it, or purchased it
- What version of the kernel you are running and what options you are using
- Whether any additional packages are not part of the standard distribution
- How urgent the problem *really* is

The last question is essential to getting you the service you need. If you are not clear to tech support about the urgency of the situation, you may end up waiting for the available support analyst or you might get the "try this and call back later answer." By explaining the urgency to everyone you contact, you are likely to get your answer more quickly.

On the other hand, most tech support is based on an honor system. The support organizations with which I have dealt will believe you when you call in and say your system is down. (This was also the case when I worked in support.) Many customer service people are not in a position to judge the severity of the problem. However, the support analyst is. Saying that your company is down because you can't figure out the syntax for a shell script is unfair to other people who have problems that are really more severe than yours. Simply turn the situation around when you are the one waiting for support on a system crash and someone else is tying up the lines because he or she can't install a printer.

Once you have all the details about the problem, you are ready to call, right? Well, not yet. Before you actually start dialing, you must make every effort to track down the problem yourself. The first reason is pretty obvious. If you find it yourself, there is no need to call tech support.

This doesn't apply as much to newsgroups, but you do save time by listing the things that you have tried. If there is no specific solution to your problem, other newsgroup readers will probably make suggestions. If you list what you have tried, no one will suggest doing something that you have already tried. Telling them what you have tried applies to tech support as well.

Many vendors have bulletin boards containing answers to commonly asked questions. There may even be a WWW page for the bulletin board to make access even easier. Unless your system won't boot at all, this is usually a good place to look before you call support. Again, it's an issue of time. It is generally much easier to get into a bulletin board than to a support engineer. You may have to spend a little time becoming familiar with the particular interface

that this company uses, but once you have learned your way around, you can not only find answers to your questions, but you often find treasures such as additional programs that are not part of the base distribution. Even if you don't find the solution, knowing that you did look on the bulletin board saves the support engineer a step. In addition, access a Web page or a bulletin board can keep you up-to-date on patch releases.

I mentioned that some companies have fax-back services. Often, answers to common questions are available this way. If you try the fax-back service, as well as newsgroups or on-line services like CompuServe, you have saved time if you need to call into support. Even if you don't get the solution to your problem, you may have gotten some of the suggestions that the tech support representative would have given you. Because you already know that something doesn't work, you have saved yourself the problem of getting a "try this and call back" answer.

From the tech support perspective, this is very helpful. First, there is the matter of saving time. If it takes 20 minutes just to get through the basic "sanity" checks, then that is 20 minutes that could have been used to service someone else. Why do you care if someone else gets help instead of you? Well, if you happen to be the one waiting to talk to the support representative, you want him or her to be done with the other customer as soon as possible to be able to get to you more quickly. The bottom line is that the more quickly they're done with one customer, the quicker it is for everyone.

Make sure that any hardware you have added is supported before you call to support. If not, getting effective support is difficult at best. Tech support may have to guess at what the problem is and possibly give you erroneous information. In many cases, you may be referred to the hardware vendor and simply told they can't help you. Not that they won't try. The issue is usually that they don't have any information about that product, so the best they can do is go from knowledge about similar products. If the product you want to use deviates from the norm, generic information is of little value.

If this is a software problem, make sure that the release you have is supported on this release of Linux. One common problem is that the software may be ELF, but your kernel only supports a.out, in which case, you have a problem.

If a piece of equipment is not "officially" supported, the support representative or people on the newsgroup may never have seen this before and may be unaware of quirks that it has. A printer may claim to be HP LaserJet-compatible, but the driver may send commands to the printer that the clone does not have. Many people will insist that this is a problem with the operating system. No one never claimed that the hardware was going to work. So, if the hardware vendor claims it is 100 percent compatible, it is up to them to prove it.

On the other hand, because of the nature of the job in tech support and the nature of people using Linux, the representatives probably have encountered hardware that is not officially supported. If they try to get it to work and they succeed, then they are in a position to try it the next time. If they have successfully installed something similar, then many of the same concepts and issues apply.

This same thing applies to software. Make sure the software is supported by the OS. It may be that the particular release of the application is only supported with a kernel after a particular release. In that case, neither the people on the newsgroup, the Linux distributor, nor the application vendor will be able to help. They know that it will not work. I remember one call into tech support in which the customer was having trouble with a version of our spreadsheet product that has been discontinued for more than two years. To make things more interesting, the customer was trying to get it to work not on our OS, but someone else's.

Also try to determine whether it is really an operating system problem and not specific to just one application. If you call your Linux distributor with a problem in an application that you got somewhere else, make sure the problem also occurs outside of that application. For example, if you can print from the command line but can't print from WordPerfect, it's not an operating system problem. However, if the OS and WordPerfect both have trouble printing, then it is probably not an issue with WordPerfect. The reverse is also true.

If the problem is with the software and deals with configuration, make sure that all of the associated files are configured correctly. Don't expect the distributor or the people on a newsgroup to check your spelling. I had one customer who had problems configuring his mail system. He spent several minutes ranting about how the manuals were wrong because he followed them *exactly* and it still didn't work. Well, all the files were set up correctly except for that he had made something plural although the manual showed it as being singular.

Even after you have gathered all the information about your system and software, looked for conflicts and tried to track down the problem yourself, you are still not quite ready to call. Preparing for the call itself is another part of getting the answer you need.

One of the first questions you need to ask yourself is "Why am I calling tech support?" What do you expect? What kind of answer are you looking for? In most cases, the people answering the phones are not the people who wrote the code, although you will find them by subscribing to mailing lists. Due to the nature of Linux enthusiasts, many *have* spent hours digging through the source code, looking for answers or creating a patch. However, this is not the same as writing the SCSI hard disk driver. Therefore, they may not be in a position to tell you why the program behaves in a certain way, only how to correct it. Despite this, Linux users may have a better overall knowledge of the product than many of the developers because they deal with more diverse issues. Therefore, they may not be in a position to tell you why the program behaves the way it does, only how to correct it.

If you are contacting the support representatives via fax, e-mail, or any other "written" media, be sure that there are no typos. Especially when relating error messages, always make sure that you have written the text *exactly* as it appears. I have dealt with customers who have asked for help and the error message they report is half of one release and half of another. The change required is different depending on the release you are running. This is also important to know when calling. Telling the support representative that the message was "something like" may not do any good. If there are several possible errors, all with similar content, the exact phrasing of the message is important.

This is also a problem with two systems when one is having the problem and the other is not. It is not uncommon for a customer to describe the machines as "basically the same." This kind of description has little value when the representatives are trying to track down a problem. I get annoyed with people who use the word "basically" when trying to describe a problem. I don't want the basics of the problem, I want the details. Often customers will use it as a filler word. That is, they say "basically," but still go into a lot of detail.

Many customers insist that the two systems are identical. If they were *identical*, then they both would be behaving the same way. The fact that one works and the other doesn't indicates that the machines are *not*identical. By trying to determine where the machines differ, you narrow down the problem to the point at which tracking down the problem is much easier. You can even find the problem yourself, thus avoiding the call to support.

Once you get tech support on the phone, don't have them read the manual to you. This is a waste of time for both you and the support representative, especially if you are paying for it. Keep in mind that although there may be no charge for the support itself, you may be calling a toll number. If this is during the normal business day (which it probably is), the call could still cost $20-$30. However, this also depends on your support contract. Many customers will pay tens of thousands of dollars a year for support so that they can have the manual read to them. They don't have the time to go searching for the answer, so they pay someone else to do it for them. If you want a premium service, you have to pay a premium price.

The same applies to newsgroups. Don't waste bandwidth by asking someone to give you the option to a command. RTFM! (Read The Friendly Manual) Every version I have seen comes with manual-pages, as well as dozens of documents detailing how to run your system.

If you do read the manual and your problem still does not work out the way you expect it to or you are having problems relating the doc to the software, ensure that the doc matches the SW. One customer was having trouble changing his system name. He said the documentation was worthless because the software did not work as it was described in the manual. It turned out that the doc he was using was for a release that was two years old, and he never got the latest doc! No wonder the doc did not match the software.

If you don't know the answer to the question, tell the support representative "I don't know." Do not make up an answer. Above all, don't lie outright. I had a customer who was having problems running some commands on his system. They were behaving in a manner I had never seen before, even on older releases. To track down the problem I had him check the release his was on. None of the normal tools and files were there. After poking around a while, I discovered that it was not our OS. When confronted with this, the customers response was that their contract for the other operating system had run out.

Getting information from some customers is like pulling teeth. They won't give it up without a fight. To get the right answer, you must tell the analyst everything. Sometimes it may be too much, but it is much better to get too much than not enough.

When talking to support, have everything in front of you. Have your notebook open, the system running if possible, and be ready to run any command the support representative asks you. If you have a hardware problem, try to have everything else out of the machine that is not

absolutely necessary to your issue. It is also helpful to try to reinstall the software before you call. Reinstalling is often useful and several companies seem to use this method as their primary solution to any problem. If you have done it before calling and the problem still persists, the tech support representative won't get off with that easy answer. I am not professing this as the standard way of approaching things, though if you believe reinstalling would correct the problem and you have the opportunity, doing so either solves the problem or forces support to come up with a different solution.

Another common complaint is customers calling in and simply saying that a particular program doesn't work right. Although this may be true, it doesn't give much information to the technician. Depending on its complexity, a program may generate hundreds of different error messages, all of which have different causes and solutions. Regardless of what the cause really is, it is almost impossible for the technician to be able to determine the cause of the problem simply by hearing you say that the program doesn't work.

A much more effective and successful method would be to simply state what program you were trying to use, then describe the way it behaved and how you expect that it should behave. You don't even need to comment on it not working right. By describing the behavior, the technician will be able to determine one of three things. Either you have misunderstood the functionality of the program, you are using it incorrectly or there really is a bug in the program.

People who call into tech support very commonly have the attitude that they are the only customers in the world with a problem. Many have the attitude that all other work by the entire company (or at least, tech support) needs to stop until their problem is resolved. Most tech support organizations are on schedules. Some have phone shifts scattered throughout the day and can only work on "research" problems during specific times of the day. Other organizations have special groups of people whose responsibility it is to do such research. In any event, if the problem requires special hardware or a search through the source code, you may have to wait several hours or even days for your solution. For the individual, this may be rather annoying, but it does work out better for everyone in the long run.

The attitude that the analyst needs to stop what he or she is doing and work on this one customers problem becomes a major issue when problems are caused by unique circumstances. The software or hardware company may not have that exact combination of hardware available. Although the combination ought to work, no one that has not tested it can guarantee there will be no problems. As a result, the support representative may have to wait until he or she is not working on the phone to gather that combination of hardware. It may also happen that the representative must pass the problem to someone else who is responsible for problems of this type. As a result, the answer may not come for several hours, days, weeks, or even *months*, depending on the priority level of the contract.

In addition to the priority of the contract, there is also the urgency of the problem. If you have a situation in which data is disappearing from your hard disk, you will be given a higher priority than your contract would imply.

While I was working in support, I talked with many other support representatives. Often a customer would have a support contract with his or her vendor and the vendor would have the contract with us. The vendor would call us if they could not solve the problem. I had a chance to ask many of them some of the more common problems.

There are several common complaints among tech support representatives. Although it may seem obvious, many people who call in are not in front of their machines. Its possible that the solution is easy enough that the support representative can help even without you at the machine. However, I talked to a customer who had printer problems and wanted me to help him fix things while he was driving down the freeway talking on his car phone.

Another very common issue that support representatives bring up is customers who come off as thinking they know more than tech support. When they are given suggestions, their response is usually "That won't work." Maybe not. However, the behavior exhibited by the failure often does give an indication of where the problem lies. If you are going to take the time to call support, you must be willing to try everything that is suggested. You have to be receptive to the support representatives suggestion and willing to work with him or her. If necessary, you must be willing to start the problem from scratch and go over the "basics." The customers who get the best response from tech support are usually those who remain calm and are willing to try whatever is suggested.

People have called computer manufacturers to be told how to install batteries in laptops. When the support representative explains how to do this and that the directions are on the first page of the manual, one person replied angrily, "I just paid $2,000 for this damn thing, and I'm not going to read a book."

At first glance, this response sounds reasonable. A computer is a substantial investment and costs a fair bit of money. Why shouldn't tech support tell you how to do something? Think about a car. A car costs more. So, after spending $20,000 for a new car, you're not going to read the book to figure out how to start it? Imagine what the car dealer would say if you called in to ask how to start the car.

The computer industry is the only one that goes to this level to support its products. Sometimes you get very naïve customers. At least, they are naïve when it comes to computers. In attempting to solve a customers problem, it is often essential that tech support knows what release of the operating system the customer is using.

Some customers are missing some basic knowledge about computers. One customer was having trouble when we needed to know the release. Although he could boot, he was having so much trouble typing in basic commands, like uname. We told him to type uname then press Enter and it responded "dave: command not found."

We then asked him to get the N1 floppy and read the release number off the floppy. He couldn't find it. Not the floppy, the release number. So after 10 minutes of frustration, we decided to have him photocopy the floppy and fax it to us.

"Wow!" he said. "You can get information from the floppy that way?"

"Sure," we said. "No problem." (What's so hard about reading a label?)

A few minutes later a fax arrived from this customer. It consisted of a single sheet of paper with a large black ring in the center of it. We immediately called him back and asked him what the fax was.

"Its the floppy," he said. "Im still amazed that you can get information from the floppy like that. I must tell you, I had a heck of a time getting the floppy out of the case. After trying to get it out of that little hole, I had to take a pair of scissors to it." (The case he mentioned was actually the plastic jacket.)

Many of us laugh at this because this is "common knowledge" in the computer industry. However, computers are the only piece of equipment about which the consumer is not expected to have common knowledge. If you drive a car, you are expected to know not to fill it with diesel when it takes regular gasoline. However, trying to load a DOS program onto an UNIX system is not expected knowledge.

One customer I had was having trouble installing a network card. The documentation was of little help to him because it was using a lot of "techno-babble" that most "normal" people couldn't understand. The customer could not even answer the basic questions about how his hardware was configured. He insisted that it was our responsibility to know that because we wrote the operating system and he's not a computer person. Well, I said that it's like having a car that won't start. You call the car dealership and tell them it won't start. The service department asks you what model you have. You say that they should know that. They then ask if you have the key in the ignition. You say that you are not a "car person" and don't know this technical stuff.

In the past few years, many software vendors have gone from giving away their support to charging for it. Support charges range anywhere from $25 a call for application software to $300 for operating systems like UNIX. As an end user, $300 can be a bit hard to swallow. However, in defense of the software industry, it really is not their fault. As more and more computers are being bought by people who have never used one, the number of calls to support organizations have gone up considerably. People treat computers differently than any other piece of equipment. Rather than reading the manual themselves, they much prefer to call support.

Would you ever call a car manufacturer to ask how to open the trunk? Would you ever call a refrigerator manufacturer to ask how to increase the temperature in the freezer? I hope not. However, computer tech support phones are often flooded with calls at this level, especially if their support is free or free for a specific warranty period.

The only way for a company to recover the cost of the support is either to include it with the cost of the product or to charge extra for it. The bottom line is that there is no such thing as a free lunch, nor is there free tech support. If you aren't paying for the call itself, the company will have to recover the cost by increasing the sales price of the product. The result is still money out of your pocket. To make the situation fairest for everyone involved, companies are charging those people who use the tech support system.

I remember watching a television program a couple of years ago on air plane accidents and how safe planes really are. The technology exists today to decrease the number of accidents and near accidents almost to zero. Improvement to both airplane operations, air traffic control, and positioning could virtually eliminate accidents. However, this would result in increasing the cost of airline tickets by a factor of 10! People won't pay that much for safety. The risk is too low.

The same thing applies to software. It is possible to write code that is bug-free. The professor who taught my software engineering class insisted that with the right kind of planning and testing, all bugs could be eliminated. The question is, "At what cost?" Are you willing to pay 10 times as much for your software just to make it bug-free? One support representative put it like this: "How can you ask us to hold up the entire product for an unknown length of time, to fix a single problem that affects few users and is not fatal? Would you expect Ford to ship their next years model of Escort three months late because they found out that the placement of the passenger door lock was inconvenient for people taller than 6'9"?" As ridiculous as this seems, calls reporting bugs are often at this level.

Because of the nature of Linux and software in general, it is going to be released with bugs in it. Although no one organization is responsible for it, Linux has as good a track record as any commercial OS. One key difference is that you don't have a huge bureaucracy causing Linux 96 to come out in 98. New versions of the kernel come out every few months and it is literally only a matter of days for patches to appear when bugs are detected.

After years of tech support, I am convinced that the statement "The customer is always right" was not coined by some businessman trying to install a customer service attitude in his employees. It must have been an irate user of some product who didn't bother to read the manual, tried to use the product in some unique way, or just generally messed things up. When this user couldn't figure out how to use whatever he or she bought, he or she decided it was the fault of the vendor and called support.

You as the customer are not always right. Granted, it is the responsibility of the company to ensure that you are satisfied. This job usually falls on the shoulders of tech support because they are usually the only human contact customers have with hardware and software vendors. However, by expecting tech support to pull you out of every hole you dig yourself into or coming across representatives as a "know-it-all" or "I-am-right," you run the risk of not getting your question answered. Isn't that the reason for calling support in the first place?

11.2.2 Consultants

You may someday find yourself in a position where you cannot continue to try to solve problems over the phone. You need someone to come to your office to look at the problem first hand. This is where the computer consultant comes in. Sometimes consultants are called in to evaluate and analyze the current situations and make recommendations and sometimes even implement these recommendations.

Computer consultants are like lawyers. They often charge outrageous fees (several hundred dollars an hour) and rely on the fact that you know little or nothing about the subject. They have a service that you need and want you to pay as much as you are willing to pay.

Fortunately, all you need to do to see whether a lawyer is qualified is to look on his or her wall. If the diploma is from Joe's Law School and Barber College, you'll probably go somewhere else. However, there are few laws governing who can call himself a computer consultant. Therefore, you need to be extra careful in choosing a consultant.

I had one consultant call for a customer of his who was having trouble with a SCSI tape drive. The consultant almost got upset when I started talking about the technical issues involved such as termination, proper cabling, etc. You see, he had a masters degree in electrical engineering and therefore was fully aware of the technical issues at hand. I asked him how much RAM his system had. He responded, "Do you mean memory? Well, there is, uh, 32, uh, what do you call them, megabytes." (No, I'm not making this up.)

Another time a customer was having a similar problem getting a network card working. Again, it was the issue of the customer not having the basic computer knowledge to know about base addresses and interrupts. The difference between thin-net and twisted pair was foreign to him. He had worked for many years on mainframes and had never had to deal with this level of problem. After more than half-an-hour of trying to help him, it became apparent that this was really beyond what tech support is there for. I suggested he hire himself a consultant. In the long run, that would ensure he got the attention and service he need. There was a long pause, and then he said, "I am the consultant."

One of my favorites is a consultant in Texas who was trying to do long-distance hardware troubleshooting for a site in Alaska. Despite the fact that they had a modem connection, it is often quite difficult to check hardware settings and cabling through a modem.

My auto mechanic has a PC running a DOS application written specifically for automobile workshops. Aside from the fact that the consultant has them start Windows and then click on an icon to start this *DOS* application, it does it's job (it's the only thing the machine is used for). Recently they discovered that they were running out of hard disk space and needed a larger drive. So, the consultant came in put in a larger hard drive and things looked better. Because it was not part of their contract, he charged them for two hours labor to replace the drive, plus 10 percent more than the average market price for the hard disk. Now, so far, this seems like an acceptable practice. However, they took the smaller drive with them, although they charged full price for the larger drive. It wasn't defective, just too small.

These stories represent four basic problems with computer consultants. First, you don't have to have studied computer science or even a related field to open shop as a computer consultant. Although electrical engineering is a related field and the person may know about the computer at the transistor level, this is comparable to saying that a chemist who knows what goes on at the molecular level inside an internal combustion engine is competent to fix your brakes.

The next issue is that although the person had worked with computers for years, he or she knew little about PCs or operating systems. I have seen enough times consultants who assume that all computer systems are the same. They worked for years on Windows so they are qualified to install and support UNIX, right?

There is also the issue of the consultant not making house calls. They have to. They have to be willing to come to your site and check the situation themselves. You cannot be expected to shut down operations to bring a computer to their office, nor should you tolerate them trying to do remote support (i.e., across a modem).

Lastly, if you do need to hire a consultant, make sure you know what you are paying for. When you do decide on a consultant, make sure that you know specifically what services are being provided and what obligations the consultant has. These services include not only hardware and software, but what work the consultant is going to provide. If the consultant needs to replace a defective hard disk, the cost of the disk is included but the time to replace it may not be included.

The best solution is to ask your friends and other companies. If you have a good relationship with another company of similar size and product, maybe they can recommend a consultant to you. Another source is the Internet and on-line services like CompuServe. Ask people there what their experiences have been. Web search engines, like Yahoo or Alta Vista, can give you names of companies that specialize in Linux as well.

How to Get the Most for Your Money

Deciding that you need a consultant doesn't mean that you are throwing yourself to the wolves. With a little preparation, you can be ready and ensure that you don't make any costly mistakes. There a four basic steps to follow when deciding which consultant to go with:

- Define the project.
- Find the right person for the job.
- Agree *in writing exactly* what the job entails and what is expected from both sides.
- Makes sure the job gets done correctly and on time.

When you think you have found the right consultant, you must treat them like a telephone company: Get it in writing! This, along with finding the right person, are the two essential factors in deciding which consultant to choose.

Let's look at the right person first. There are several ways to go about choosing a consultant. First, you can pick up the telephone book and find the one with the fanciest ad. Personal referrals are also a way, but this can cause a lot of problems. If the consultant is a friend or family member of the person making the recommendation, you can get yourself into an awkward position when you either find he or she is not the right person for the job or he or she is not really competent and you have get rid of him or her. Personally, I think recommendations from other companies are best. They have had real experiences with the consultant and (should) know their capabilities.

Part of choosing the right person is making sure that he or she has the skills necessary to get the job done. Never hire a consultant who doesn't know the product or issue at hand but insists can learn it. You are paying for an expert, so that's what you should get, not someone still in training. The process is basically the same as hiring an employee. You can request a resume and references and then call those references. Things to ask the references should include the following:

- What did you think of this consultant in general?
- What did you think of the consultants technical abilities?
- Did he or she interact well with your employees? < LI>Did he or she follow through with commitments? Finish on time?
- When the project was finished, were there any points of dispute? How well did the consultant react?
- Did you understand what the consultant did?

When you have your first meeting with the consultant, there is nothing wrong with having your expert present to "test" the consultants knowledge. This is the same thing as an interview you are trying to determine whether to hire this person. Therefore, you have the right to ask about his or her technical abilities.

In one company for which I worked, we had a very bad experience with a consultant. The company ran mostly PCs with Windows for Workgroups, but there were several UNIX servers and workstations. We found a consulting firm that were "experts" with Microsoft's Exchange because we were planning to implement this on the company's intranet. We explicitly told the consulting firm that one of our goals was to get connected to the Internet. We scheduled a three-day workshop during which the consultant would go through the details of configuration and give us guidance on how to implement it.

When the consultant arrived, we were pleasantly surprised that it was one of the owners of the firm. However, the pleasure was short-lived when we discovered that he had no understanding of Internet mail and therefore could provide us no guidance on how to configure MS-Exchange for the Internet. We also later discovered that he was no expert with MS-Exchange because he spent the entire afternoon on the last day trying to get a basic configuration issue to work.

This taught us two things. First, just because someone is the owner of a consulting firm does not mean he or she knows what he or she is talking about. Unlike with doctors, few laws govern who can call him- or herself a consultant. Second, we were not clear with what our expectations were or what the consultant was to provide. Nothing was in writing other than that the consultant would give us a "workshop." It was obviously up to the consultant to decide whether he had achieved this goal.

There are many areas in which a consultant is necessary. You cannot hire experts in every area. It would just be too expensive. Even if you do have people in your organization who are experts, it is often useful to have someone come in with a fresh perspective. As an employee, you often have emotional responses involving your system or company that a consultant doesn't have. This is helpful to get to the core of the issue.

Another aspect is the specialization. A consultant has a particular skill set in which he or she knows almost everything (at least that's what you're hoping). Being really good at this one subject means that he or she may not be as useful to a company to hire the person full time. However, if the company is involved in a project that requires that skill, it is cost-efficient to hire the expert and get the job done more quickly. I think of setting up an Internet server as the primary example. After I had done it a few times, it became a lot easier. However, once I have done it a dozen or so times, it might become easy. Potentially, I could hire myself as a

consultant to develop Internet servers. (But then again, maybe not.)

When you hire a consultant, you must know what you want out of him or her. What information do you expect the consultant to impart on you or what project do you expect the consultant to complete? What does "complete" really mean? If the project is configuring a Web server and all the consultant does is hook you up to the Internet, then the job is not done. If the project will take a long time and you expect regular status reports, have the consultant define when these reports are due. If he or she says every other Friday, then handing it to you on Monday is not acceptable.

You may not be able to use the correct "buzzwords" to define what you want, but you can come up with a clear idea of what you want.

Once you have the concept of what you want, you should work with the consultant to define the project in the correct terminology. However, don't let the consultant confuse you. If you don't understand, say so. There is nothing wrong with not understanding something. If you were an expert on this subject, you wouldn't need a consultant. One thing that our MS-Exchange consultant did a lot of was talk in techno-babble. He would throw out a technical word to make him sound like an expert. The problem was that he really didn't know much about the subject and often used the words in the wrong context. If you get the feeling that the consultant is trying to baffle you with buzzwords and techno-babble, it's time to get another consultant.

When dealing with a consultant, you are bound to face concepts and vocabulary that are foreign. What about the other way around? Will the consultant know everything about your business? If the consultant specializes in your area, you would hope so. However, you are probably hiring a computer specialist, not a legal specialist or medical specialist or wholesale distribution specialist. Therefore, there is a lot that you will have to explain to your consultant.

Do not assume that the consultant knows your business at all. Specify *every* aspect of the project. One example is a wholesale soft drink distribution company. When people buy large quantities of soda, they are most familiar with buying by the case. A consultant you hire to develop a tracking system may take this for granted and write the program to deal only in cases. What if you distribute containers of cola syrup as well? These are not measured in cases. If you assume that the consultant knows this and don't tell him or her and he or she programs for cases, who is responsible for paying for the changes? You said you wanted a tracking system and you got one. The project description didn't mention the kind of units.

Don't let the consultant get away with estimates on anything. If he or she estimates anything, it can be off. Just like the estimate on car repairs. The more vague the job description, the easier it is for the consultant to postpone or claim that something was never agreed on, in terms of time as well as money.

If the job will take a while and you have said you want status reports, you can use these reports for the basis of payment. For example, the project is to take 10 weeks with five bi-weekly status reports. Each time you get a status report, the consultant gets one-fifth of the total fee. Another way would be to set "milestones." Each phase of the project is to be done by a certain date. At each milestone, the consultant gets a certain percentage of the total. The idea

of completion-based payment is important if you have deadlines to meet yourself. The consultant must be made aware of these as well. It is not unreasonable to make completion within the time specified be part of the contract. However, you need to be clear in the contract what is to be done and by when.

The consultant may not be working solely on your project during the time you contracted him or her. This is acceptable, provided he or she meets all his or her commitments. Explaining to you that he or she couldn't meet the deadline because of a problem at another site should tell you that the other customer is more important. They might be, so find a consultant who will consider you most important.

Do You Get What You Pay For?

Well, that depends. Just because a consultant asks for a high rate does not mean he or she is good. I consider Ferrari or Jaguar as examples. These are very expensive cars. They have a "performance" comparable to their price in that they go fast. If you buy a Ferrari consultant, he or she might be done with the job in a short time. However, as with the Ferrari, you might spend as much on repairs as on the cost of the car.

On the other hand, a consultant's rates will get higher as he or she gets better. Not only does he or she have more technical ability, but he or she has the ability to do a better job more quickly. As a result, you pay a higher rate for his or her time, but you pay for less time. Therefore it comes out cheaper in the long run. Even if it is not cheaper on your checkbook, having the project done faster may save you money.

Some consultants are paid $200 an hour, some are paid $1,000 a day. Those are reasonable prices. Your employees (probably) don't get paid that much, so why should you pay a consultant like that? Well, first, a consultant may not be "on the job" when he or she is at your site. Depending on the project, there maybe hours, days, or even weeks of preparation. Plus, there are all the administrative costs for the consultants office. You have to pay for the people in your IS/IT department out of your own budget, but for not the company receptionist. The consultant does.

Legal Issues

Remember that the consultant may have complete access to all of your data. Though I am not saying he or she is likely to be a spy for your competition, you need to be careful. Even if the consultant doesn't have access to your more precious trade secrets, having him or her sign a nondisclosure agreement is a wise decision. This could be as simple as stating that the consultant is not to disclose any aspect of the job to anyone, or it may go into detail about what is and is not to be kept secret. Talk to your lawyers about this one.

When the consultant finishes the project, who owns the project? Well, you do as far as the project within your company is concerned. The consultant is not going to charge a license fee to use the program you paid him or her to develop. (We hope.) However, what about the code itself? This was done on your time, so like the code a regular employee writes, it's yours, right? Well, it may be the case that the consultant does keep the right to the code he or she has written, although the compiled, running program is yours. Make this clear in your contract. If you want the right to everything written by the consultant, make sure that part is written in the

contract as well.

One important aspect of the contact is the default terms, that is, what happens if the consultant defaults on the agreement. This is especially important if you have deadlines and by not meeting them you loose money. It is not unreasonable to deduct a specific amount from the total for going past the deadline. Not only does the consultant not get paid for those days past the deadline, but money is deducted from what is owed him or her for the other days. I have seen consultants who intentionally overextend themselves just to get a contract. They can promise to have it within a certain time, but have no pressing need to unless they will be loosing money.

You have to be careful with this one. If the project is a feasibility study and it turns out that the project is not feasible, did the consultant do his job? Sure, he or she determined whether the project was feasible. Therefore, he or she did his or her job. Also, what happens if the cause of the delay is not the consultants fault? If you promised him or her certain resources that were not available, you are at fault.

You might even get a consultant who has an attitude of all or nothing. That is, if he or she doesn't deliver on time and what is promised, you don't pay him or her. However, you can guarantee that this consultant will probably have you spell out everything you want done so there is no room for discussion.

Points to Remember

When dealing with consultants, remember these general issues that will help make things easier:

- A consultant is not one of your employees. Don't insist that he or she arrive at a certain time or work until a certain hour. Maybe part of what he or she is doing can be done at his or her office. You're concerned with him or her getting the project done on time and not being physically present at your site.
- Judge the price you pay by what it would take you to do the job without the consultant. How many hours would it take? How much money might you loose? If you would end up paying more than a "high-priced" consultant, the consultant is cheap. However, comparison shopping is also valid for consultant. Get a second or even third estimate.
- Insist on some kind of proof that the consultant knows what he or she is talking about. A resume is fine, but references are better.
- Make sure the consultant communicates well. Can he or she express himself? Does he or she understand your needs and requirements?
- Be comfortable with the consultant. If there is something about him or her that you don't like, you don't need to hire him or her, just as it would be for a normal employee.
- Don't judge the consultant by personal appearance. Then again, I wouldn't hire a slob. It's okay to expect him or her to be clean, but don't expect a suit.

11.2.3 Other Sources

Although, as I said, there is no such thing as a free lunch, you can get pretty close sometimes. For about less than a $30 start-up fee and about $10-$20 a month, you can get support that is comparable to the tech support of a software vendor. The only problem is that it might be several hours (rarely longer) before you get an answers. I am talking here about things like mailing lists, forums on company web sites and so forth.

If your problem is general, such as getting DNS to work, or you need help with a shell script, this is not a bad place to start. Despite the delays that are caused by the very nature of this media, responses are fairly quick. Unless your system has crashed and you are loosing thousands of dollars an hour, this is an excellent source of information.

One valuable source are the USENET newsgroups. To gain access, however, you need a "news feed," that is, some other site that will provide you with the newsgroups, because USENET uses "store and forward." Turn around time can be days, depending on where you get your feed. Compuserve, on the other hand, stores all of its messages in a central location. The minute your message is posted, it is available for everyone. Talk to your ISP about finding a news server.

Mailing lists are another way of getting information. The difference between lists and newsgroups is that you receive *all* of the messages posted to the list. Therefore, you need to be careful about what you subscribe to. I subscribe to just a few and I have *hundreds* of messages a day to wade through. With many, you can subscribe directly from the Internet.

Throughout this site, I made references to several other works that I feel would be very helpful if you want to learn more about a particular subject. Aside from books, many magazines provide useful information. Not to be outdone by the commercial UNIX dialects, Linux has its own monthly magazine, called *Linux Journal*. It includes articles for almost all skill levels. You can reach the *Linux Journal* at *subs@ssc.com*, (206) 782-7733.

Another place to check is the Internet. Unfortunately, its hard to be more specific than that. Thousands of resources are out there for every conceivable topic. Most major computer companies have a Web site. Often all you need is to add "www." to the company name to get access to the Web page. I have done this to get to Intel (*www.intel.com*), Conner (*www.conner.com*), and even CNN (*www.cnn.com*). Here you can get product information, as well as press releases. Many of these sites have links to other sites, so it’s easy to bounce from site to site.

Also, several search engines have already done a lot of research for you. The one I use most frequently is Yahoo (*www.yahoo.com*), which also links to other search engines. Web sites are broken down by category, which makes browsing easy. You can also input search strings to help you find what you are looking for. If you input a phrase that’s fairly common, you can end up with thousands of matches.

Chapter 12 Security

Linux, like any computer system, has a set of security issues that need to be considered. Regardless of what mechanisms are in place, the basic concepts are the same. In fact, the security of a computer system is very much like the security of your house, just as running a computer system is like running a household. You only want to let those people in that should be let in and you only want people accessing resources that they should. (Do you *really* want your 3-year-old playing with your collection of Dresden Porcellan?)

The term *security* is common enough. On a personal basis we think of it as freedom from risk or danger; being safe. We might also think of this as the methods we undertake to prevent someone from breaking into our house. In computer science terms, both of these ideas are applicable, depending on what you are referring to.

If we talk about being safe from risk when working with computers, we are often talking about things like regular backups and reliable hardware. Although these are very important issues, these are not what is generally meant when referring to security. On computers systems, security is more along the lines of preventing someone from breaking in. The definition can be expanded by saying computer security is preventing someone from doing something that they are not allowed to do. This could be anything from reading other peoples mail to stopping the printers.

In this section we're are going to be talking about what mechanisms exist to keep people from poking around and doing things they shouldn't. Well talk about what tools Linux provides to control access, change what users can access and how to make sure users are not even trying to do things they shouldn't.

12.1 Real Threats

One of the things that I enjoyed most about one job I had was that I was one of the few people that most of the end users felt comfortable talking to. One day I was approached about the way we required passwords to be changed every couple of months. Computers are there to be used and not to keep people out. Many were annoyed that they even had passwords, let alone had to change them regularly. The biggest problem is not that he was right, but that he, as well as many users and even system administrators, don't understand the dangers involved.

The stereotypical image of a pair of teenage computer enthusiasts breaking into a military computer and almost starting a war, may be good for Hollywood, but the times have changed. Yes, there are still those kind of hackers running around, but they are not likely going to break into systems with the more advanced security techniques employed today as most of the security is good enough. But then again, maybe not.

Hacking has become almost a cult phenomenom with newsgroups, magazines and even their own language. The people that belong to this culture are not only equipped with the latest technology, they have an almost never-ending list of new security holes that they can use to break into a system. Since they spend much of their free time trying to break into systems, they may have found some of the security holes themselves. However, the techniques they use

go beyond just the list of known holes although these are probably things that are tried first. Instead, there is a methodology to the attack.

More and more, hackers are not just randomly trying systems across the country. Instead, there is usually some motivation for attacking a particular site. It may be just the notoriety of being the first to break into the crystal palace that is some major corporation. In some cases, this is what these people do for a living. The ability to break into a competitors computer system and look over the shoulder of his R&D people, may be worth the investment of hiring a hacker.

As we all know from many of the detective shows we see on TV, criminals are caught because of the clues they leave behind. This also applies to the computer hacker. Breaking into a computer is less likely to leave evidence that can trace directly back to perpetrator. Instead, it is usually a case of being caught in the act during a subsequent break-in. Then there is the added problem of criminal jurisdiction as the hacker could just as easily be on the other side of the world as on the other side of town.

Just knowing that you should lock your front door or buckle your seats belts is enough for many people to do it. However, I am not one of those. Understanding that someone could walk away with my TV or my head could go flying through the windshield is what motivates me to do what I should. I am then also less likely to forget to do or intentional not do it one time because it’s inconvenient. I take the same approach to computer security.

Most system administrators are aware that there needs to be "security" on their system. I put it in quotes, because it is often just a phrase that is brought up at staff meetings. When addressed, this often just means forcing users to change their password at regular intervals or making sure that users were logged out when they went home. One company I worked at forced users to change their password every six weeks, but the root password was only changed when someone left the company. It was too inconvenient. Added to that the fact that the root password for all the machines were variations on a single theme, so once you figured out one it was easy to figure out the rest.

With all the talk of the Internet, the kind of security most often in people’s minds is the attack from outside. Although this is a very real threat, it is not the only one. Personal experience has taught me that inside attacks can be just as devastating.

In this same MIS shop everyone had the root password to every machine also the administrator password on our NT machines. There were people who only administered the UNIX machines and others who only administered the NT machines. However they had the password to all machines. One employee was not satisfied with the speed that the hardware vendor was reacting to a problem he was having with one of the NT machines. Since they were the same vendor for the UNIX machines he decided to "motivate" them to make a personal call.

On several, irregular occasions he killed the Oracle database process. Since most everyone used that database, the company was brought to a standstill for the couple of hours it took to discover the problem, reboot the system and clean up. Eventually he was caught, but not after causing tens if not hundreds of thousands of dollars worth of damage.

Keeping the UNIX root password from him would have probably prevented him from doing this exact thing. However, there are other things that he could have done to damage the company if that was his intent. Nothing can prevent this kind of act. However, if passwords are limited and something goes wrong, it is not so easy for the guilty party to deny it.

In the beginning, I was a firm believer that information about what security holes should be kept secret.security by obscurity I had an obligation as the all-knowing UNIX guru to protect the innocent system administrators in the world. Therefore, I felt it was improper to discuss these issues publically.

As I began to read more about security, I discovered that I was one of the few people that shared this belief. Most of the books and articles and books that I read presented the material as "Here's the threat and here's what you can do about it." By not only knowing that there is a threat but why it is a threat, you can correct the problem as well as identify other potential problems that may not have been discussed.

On any computer system, there is always the danger that something can be compromised. Now the word "danger" can span a whole spectrum of meaning and it all depends on what you are talking about. It might be dangerous to leave a bowl of sugar on the counter where you're two-year-old can reach, just as it might be dangerous to walk through Chernobyl without a radiation suit. It's purely a matter of scale.

The dangers involved with an insecure computer system are like that. If someone else found out the password of another user on our system, the danger of damage is low. On the other hand, if someone found out a password for a computer at the CIA, the danger is greater.

The damage caused can also span the entire spectrum. Sometimes there is no real damage. Someone who breaks into a system might simply be curious and wants to look around. This is comparable to having someone wandering through your living room.

The "Worm" that Robert Morris let loose on the Internet in 1988 was such an event. Although little real damage was done, it "infected" 2100-2600 computers. Many machines were brought to a standstill as the filesystem filled up and the system could no longer write it's log files and was busy running the processes that the worm started. In the end, it has been estimated that between $1 Million and $100 Million was lost due to time spent cleaning up and the loss in productivity when the systems were down. Even with the lowest estimates, the lost was stunning.

On the other end of the spectrum is the case that was documented by Cliff Stoll in his book Cuckoo's Egg. The information that these intruders from then West Germany gathered from over 450 government and military computers was sold to the Soviet KGB. There were a few convictions and one of the prime suspects was found burned to death in a wooded area near his home.

Computer intruders also have the ability to cause physical damage. A virus that's introduced to a system acting as a file server for DOS PCs could change the scan rate of the monitor which can cause it to explode. One computer that was broken into that Cliff Stoll was monitoring was used to regulate the radiation doses given to cancer patients. If the computer behaved unexpectedly as a result of the hackers actions, it could have meant the death of a

patient.

In any information system, whether it is a computer or filing cabinet, there are some basic security issues that need to be considered. First, there is one aspect of security that no operating system can help you with: the physical security of your system. You might have all the security implemented that Linux provides, but if someone can walk off with your computer, even the highest levels of operating system security don't do any good. Just as a security policy in an office has no effect if someone can just walk away with sensitive files.

One of the easiest and most effective types of physical security is simply a locked door. This prevents the "crime of opportunity" from ever happening, such as someone from just walking away with pieces of equipment, or the whole machine for that matter. The only thing that can prevent this kind of theft is more elaborate security measures that are beyond the scope of this book. However, it is something that you must give serious thought to. Locking the door to the computer can also prevent people from breaking into the system. Anyone who has a set of installation disks or an emergency boot disk set can gain access to your system if they have access to the computer itself.

Another aspect of physical security is access to the machine itself. It may be impractical for someone to walk off with your computer. However, a knowledgeable user with root access to a another Linux system can gain access to yours if they have physical access. Even without access to another system, if that user has access to the installation floppies, they can get into your system. Once in, it doesn't matter what kind of security is has been configured on the hard disk since the only security the system knows is what it has been told by the floppy.

The next issue is privacy. This can be the company's privacy or that of individuals. You don't want unauthorized users to have access to payroll records, just as you don't want to have access to other employees personal files.

One of the most commonly ignored aspects of this is the power of small pieces of information. As individual items, these pieces may have no significance at all. However, when taken in context they can have far reaching implications. Police use this same concept to investigate crimes and intelligence agencies like the CIA use it as well. Extending this to the business world, such techniques are useful for corporate spies.

There are other cases where security is important in business. What if someone came along and changed an important piece of information? For example, an employee who thinks he is underpaid may want to change it. Whether this information is on paper or in a computer, the integrity of the data is an important part of security. Along the same lines is the consistency of the data. You want the same behavior from the system is identical situations. For example, if salary is based on position, inconsistent data could mean that the night watchman suddenly gets paid as much as the company president.

Another aspect is the concept of auditing. Like an audit of a company's books, auditing in a computer security sense is a record of the *transactions* or *events* that occurred on the system. This allows the system administrator to follow the tracks of suspected perpetrators and maybe catch them in the act. It was a combination of auditing and accounting for time on the system that led Cliff Stoll to discover his hackers.

When preparing one company for connection to the Internet, I checked the security on the system. I found dozens of holes in the system. Keep in mind that this was actually my first attempt at being a hacker. Added to that, I exploited no real bug in the software, instead I just took advantage of "features" that were not considered in a security context. By using just the tools and programs that the system provides, I was able to gain complete access to the system. Once the system is compromised, the danger of further compromise grows steady. The only safe thing to do is to reinstall from scratch.

Its not meant to scare you to say that every system has the potential for being broken into. In the end, every security related decision and every function in the program was written by a human. The security could be mathematically tested, but who is to say that the mathematical test is not flawed?

The first step in stopping the would-be intruder is to keep him from getting to your system in the first place. This is similar to having a lock on your front door. You could go to the extreme of fencing off your property, hiring full-time guards, installing video cameras and alarms, but this is too extreme for most people. First they probably can't afford it. Second, the threat is not that great compared to the costs.

But what about your business. The potential loss from someone breaking in can be devastating. Corporate spies can clean out your sensitive data or a disgruntled former or current employee can wipe out your entire system.

With regard to the Internet, the only way to ensure that no one can break in is to completely cut yourself off from the rest of the world. This also means no modems, ISDN lines or any other device that can be used to call in and out. For some companies, this may be the only way to go. However, because of the fantastic market potential on the Internet, it may not be a wise decision.

If there is a physical connection to the outside, there is the *potential* that someone could break in. However, once you have made the decision to connect to the Internet, you need to be much more aware of security than when you network was isolated.

When a system is improperly accessed, the attacker may not necessarily continue with the attack immediately after gaining access. Instead, he might create himself backdoors to gain access to the system as a later time. He can add entries to .rhost files to give him access later. For example, putting the line + + would give him access from any machine with any account. New accounts can be created that give him access. He can also use one machine to gain information about other machines and the network in general.

An unauthorized user gains access to a system and is able to determine what files and directories this account has access to. He then places .rhosts and .forward files in every home directory he has write permission on. He now has unlimited access to all of those accounts, even though he never knew their password.

In the **`.forward`** file is a pipe to a script that copies **`/bin/sh`** in **`/tmp`** and makes it SUID to that user. Whenever **`/tmp/sh`** is started the UID is the new user. Now access can be obtained to other machines with the appropriate entries in **`.rhosts`** or **`host.equiv`**.

12.2 Restricting Access

Regardless of what security issue you are talking about, any breach in security can be prevented by not allowing access to the system. Now, this can be taken to extremes by not letting *anyone* to have access. However, by limiting access to the system to only authorized users, you substantially lower the risk of breaches in security. Keep in mind that there is no such thing as a secure system. This is especially important when you consider that the most serious threat comes from people who already have an account on that system.

Access control has been a part of UNIX for a long time. It is a fundamental aspect of any multi-user system. The most basic form of access control is in the form of user accounts. The only way you should be able to gain access to a Linux system is through an account. Users usually gain access to the system when they have an account set up for them. Each user is assigned an individual password that allows the access. Access to files is determined by the permissions that are set on the files and directories.

Access to various services such as printers or even local file systems is also determined by permissions. In addition, some services, as we shall see shortly can be restricted at a much lower level.

12.3 Passwords

In some cases, passwords may be blank, meaning you only need to press enter. In other cases it can be removed altogether so you are never even prompted to input your password. Removing the password may not always be a good idea. Since you have the source code, Linux allows you the option to prevent users from either having no password or having to just press return. Since we are talking here about security and accounts without passwords are not very secure, we'll restrict ourselves to talking about accounts that have passwords.

On many systems (including many Linux versions) you cannot force users to use (or not use) specific passwords. As a system administrator it is your responsibility to not only enforce a strong password policy, but to educate your users as to why this is important. Later, we'll go over some examples of what happens when users are not aware of the issues involved with password security.

If you write your password on to a Post-It and stick it on your monitor, no operating system in the world can do anything about it. But what about cases where you inadvertently give someone your password? This happens when users choose passwords that are easily guessed by someone trying to break in. Often users will choose passwords that are easy to remember, such as their license plate number or spouse's birthday. Linux cannot do anything to keep you from using your license plate number as a password. However, some features can be easily built in to limit what you can use as a password.

12.4 File Access

Although this password protection stops most attempts to gain unauthorized access to the system, many security issues involve users that already have accounts. Unchecked, curious users could access payroll information and find out what their boss gets paid. Corporate spies could steal company secrets. Disgruntled workers could wreak havoc by destroying data or slowing down the system.

Once logged in, Linux (among other UNIX dialects) provides a means of limiting the access of "authorized" users. This is in the form of file permissions, which we already talked about. File permissions are one aspect of security that most people are familiar with in regard to UNIX security. In many cases, this is the only kind of security other that user accounts.

As we talked about earlier, each file has an owner, whether or not some user explicitly went out there and "claimed" ownership. Its a basic characteristic of each file and is imposed upon them by the operating system. The owner of the file is stored, along with other information, in the inode table in the form of a number. This number corresponds to the User ID (UID) number from **`/etc/passwd`**.

Normally, files are initially owned by the user who creates them. However, there are many circumstances that would change the ownership. One of the obvious ways is that the ownership is intentionally changed. Only the owner of the file and root can change its ownership. If you are the owner of a file, you can, in essence, "transfer ownership" of the file to someone else. Once you do, you are no longer the owner (obviously) and have no more control over that file.

Another characteristic of a file is its *group*. Like the owner, the file's group is an intrinsic part of that files characteristics. The file's group is also stored in the inode as a number. The translation from this number to the group name is made from the /etc/group file. As we talked about in the section on users, the concept of a group has only real meaning in terms of security. That is, who can access which files.

What this means is that only "authorized" users can access files in any of the three manners: read, write and execute. It makes sense that normal users cannot run the fdisk utility, otherwise they would have the ability to re-partition the hard disk, potentially destroying data. It also makes sense that normal users do not have write permission on the **`/etc/passwd`** file, otherwise they could change it so that they would have access to the root account. Since we talked about it in the section on shell basics and on users, there is no need to go into more details here.

12.5 The Root Account

There is also access to the all powerful root account. On a Linux system root can do anything. Although it is possible to restrict roots access to certain functions, a knowledgeable user with root privileges can overcome that. There are many instances where you have several people administering some aspect of the system, such as printers or the physical network. I have seen it myself where one person says "Well, he has root access, why can't I?"

Access to the root account should be limited for a couple of reasons. First, the more people with root access, the more people who have *complete* control over the system. This makes access control difficult.

Also, the more people that have root access, the more fingers get pointed. I know from experience that there are people who are going to deny having done something wrong. Often this results in a corrupt system, as there are everyone has the power to do everything, someone did something that messed up the system somehow and no one will admit. Sound familiar?

The fewer people that have root, the fewer fingers need to be pointed and the fewer people can pass the buck. Not that what they did was malicious, mistakes do happen. If there are fewer people with root access and something goes wrong, tracking down the cause is much easier.

Rather than several users all having the root password, some people think that it is safer to create several users all with the UID of root. Their belief is that since there are several lognames, it's easier to keep track of things. Well, the problem in that thinking is that the system keeps track of track of users by the UID. There is no way to keep these users separate, once they log in.

My personal suggestion is that if several users need root powers, that you make it company policy that no one logs in as root. Instead, you grant each required user the su system privilege. They then login with their own account and do an su to root. Although everything is still done as root, a record of who did the su can be written to **`/var/adm/syslog`**.

Once an intruder gains root access, then your entire system is compromised. It is therefore important to not only limit who has access as root, but to record who uses the root account. One way is to implement a policy that no one logs in as root, but must first login with their own account and then do an su to gain access to root.

Another security precaution is to define secure terminals. These are the only terminals that the root user can login from. In my opinion, it is best to only consider directly connected terminals as "secure". That is, the root user can log into the system console, but not across the network. To get access as root across the network, a user must first login under their own account and then use su. This also provides a record of who used the root account and when.

12.6 The Network

If you have a stand-alone Linux system, or one that is connected on an internal network with *no* connection to the outside world, then security is much less an issue. (It does *not* go away) However, if you connect to the Internet, such as for a HTTP or FTP server, then security is a primary consideration.

One way of avoiding compromising your system is to have your WWW server connected to the Internet, but not to your internal network. Should someone be able to break into the WWW server, the worst that can happen is that your WWW server is down for a day or so as you reload from backups. If the intruder had access to the internal network, your livelihood could be threatened.

By it's very nature, UNIX is not very security oriented. When it was first designed and implemented, UNIX was by programmers for programmers. The environment was of cooperation, not privacy. As UNIX moved into universities and businesses, that changed. Security was an issue. Because security was not built into the ordinal concept, it had to be included "after the fact." Therefore, security solutions were not as far reaching as for later systems.

The severity of this problem can be demonstrated by what I found at one company I was working for. In preparation for connecting the company to the Internet, I conducted a security check of the internal network. I wanted to see just how far I could get. One of the first steps that a burglar does before he breaks in is to case the joint. He may observe it for several days or weeks before making his move. To make his presence less conspicuous, he may watch several scattered locations and then choose the easiest target. (Or may choose all of them in turn.) A computer break in is basically the same. The only difference is the tools the burglar uses and the information that is collection. However, in both cases the more careless you are as the potential victim, the easier time the burglar has in gathering the information and breaking in.

Since we are not trying to keep someone from breaking into our house, let's talk about the tools that a hacker would use to break into your computer system. One of the most innocuous and most dangerous is finger. In the many papers and books that have been written recently about computer security and break-ins, finger is always mentioned. I have used it myself on our internal network and have collected a great amount of information. What information is provided is dependant on the operating system and the version of finger. However, at the very least it can provide information about who is logged in, where they logged in from and so on.

One common tactic used is going on the belief that an account that is not used that often will have a easy to guess password. Based on my personal experience, this seems to be true. Usually people that don't use their computer accounts are not as aware of the security issues and are more than likely to choose a password that is easy to remember and therefore easy to guess. What are good passwords and what are not is something we'll get into in a minute.

Finger often delivers information stored in the .plan file in a user's home directory. This may contain personal information that a hacker can use to try and guess the password. If the password is not easy to guess, the information obtained from finger can be combined with other information that may be useful. However, one thing that finger quite often delivers is a user's home directory. If that home directory is exported through NFS, an attacker may be able to mount that directory, copy an .rhosts file into the directory and can access to the system without even supplying a password.

At the same company, there was a very arrogant system administrator who would simply not accept the fact that "his" system was insecure. However, one of the home directories that was exported via NFS was his. Since I had root access on my machine, I could import his home directory. His **`.rhosts`** file was writable so I could give myself permission to use rlogin to his account from any machine as any user on the network. Once in, I planted a Trojan horse version of **`su`**, since I knew he would eventually use it to get access to the root account. Even if I wasn't root, having a writable **`.rhosts`** file allowed me to gain access.

One very common attack is the dictionary attack. Here the hacker uses common words, encrypts them using the same as the password taken from the password file and then the two are compared. Remember that the **/etc/passwd** file is readable by everyone and the seed is contained within the encrypted password is always the first two characters. Once I have access to the system, I can bring a copy of this to another system and using that seed I can encrypt the words from my "dictionary." In addition to just words in a dictionary, using place names and other proper nouns related to the target.

With just my first attempt at cracking passwords I was able to crack almost 200 on one system alone. In fact, this was the first time I tried to hack a system at all. Among the passwords I was able to gather was the one used by the head of purchasing, the head of sales and the company president! This list only contained about 30 words including the name of the town and state we were in, the days of the week, the months and a few words related to the company. Plus the program only had to run about half an hour. What kind of luck would a serious hacker have with 30,000 words running the program for a week?

Although this seems to be a major security hole, it is very effective if you use passwords that are not easy to guess. The reason is that the encryption goes only one way. You take a word, use the seed to encrypt it and then compare it to the encrypted password. However, there is no way to take the encrypted password and use the seed to figure out the un-encrypted password.

Keep in mind that snatching the **/etc/passwd** file does not necessary mean you have to break into the system to begin with. I was able to get it on one system using the "guest" account that had a *very* easy password. With just a single password I could then log into the system. Once in, the potential for more serious and directed attack is much greater. I could continue to use these accounts or edit the .rhost files in various home directories to continue to gain access even after the passwords get changed. Remember, here I got almost 200 hundred on my first attempt!

It was once common to find UNIX machines that had an account *guest*. This stems from the time when people were not so worried about security and computer resources were freely shared. Often the password for such accounts is very easy to guess. Thinking about it for a minute, I though about the first word one might say to a guest: welcome. Sure enough, that's what the password was. So, on my very first try as a computer hacker I was able to "break in."

When exporting filesystems or directories there are several things to watch. First, I recommend against *ever* exporting a filesystem to the whole world, especially one that is writable. There is generally no need to make this information available outside of your company and if so, there are probably just a few trusted hosts. Try to see if the same result can be reached by making the information available via ftp or the Web.

If there is a + in the **/etc/hosts.equiv** file, this is a wildcard that says any non-root user can login without a password. If an attacker gets into a machine as root that has an entry in the hosts.equiv, they could do an su to the user bin or sys,. Then they could use rlogin to gain access to the other system and then have access to many key files and directories. Permissions could then be changed to set the user id on executables to root and once the program is started, the user is root.

One way I got the **/etc/passwd** file was through ftp. Anonymous ftp was disabled on this system, but I simply used the "guest" account which had a password that was easy to guess. The most obvious solution is to disable ftp. However, if it is a necessary service, you can limit the potential for damage. You need a passwd file when using ftp, but it doesn't have to be the same one that you use when logging in normally. In fact, there are many things you can do to configure ftp to allow people access to your system without open it up for them. We'll get into configuring anonymous ftp shortly.

Once in I could copy the password file to my home machine and begin to crack it. Not just try to crack it. I knew going in that the odds were in my favor. Once I had the passwd file, I was statistically guaranteed that I would crack at least one password. People are people and will tend to choose passwords that are easy to guess.

Within about 20 minutes I was able to create a password cracking program on my own. Since I had never done this before, it took that long. Since, the program was only a couple of dozens lines (without the enhancements I later made) it was easy to do. I discovered subsequently that there are already password cracking programs available on the Internet that are much more powerful.

I then created a "dictionary" of words to try. I encrypted each using the seed/salt that was in the password file and then compared this encrypted word with what was in the password file. If they matched, I had found a password.

The dictionary that I had created contained only about 50 words, including the name of the company, the city and state where it was located, the generic term for the product that the company produced and a few other words related to the company and the area where we were at.

Since there were only 50 words to compare, the program ran relatively quickly. Within half an hour, I had found almost 200 passwords out of about 850 users! Most of these still had the original, start-up password *welcome*.

I then went back to the original system and did a search of the word "phone" in any file or directory name. Soon, I had a copy of the company's telephone book that I used to crack more password. In the end, I had 235 passwords.

An analysis of the passwords showed some interesting things. One person chose as a password the geographic area he was responsible for. His girlfriend, the personal secretary of the company president, chose his name as her password. Other people chose their first name, their spouse's first name and other easy-to-guess password. One even chose *123456*.

One thing bothered me about the system in general. Of all the passwords on the system, over 400 (almost half) had the same seed. I could have sped things up by encrypting all the words in the dictionary with this one seed and I would have still cracked over 100 passwords within about five minutes!

Since I used the same password on many different machines, I went on the assumption that other people did the same. As you might expect, several people used the same password elsewhere. The reason I only cracked about 10 passwords on other machines was that very

few people actually had accounts on other machines.

I then tried some of these passwords in our bookkeeping and management software. Here too I was able to crack "only" about 20 passwords including the head of the purchasing department and the head of sales.

For a real hacker, the speeds of machines have become an advantage. Whereas checking a single password on a Microvax several years ago would have taken hours, the same password can now be cracked within a matter of minutes. It has been estimated that in order to encrypt a dictionary with 250,000 words using all 4096 seeds and several machines networked together, you would need just a few hours.

On several machines I was able to list what filesystems were being exported. Also using finger information, I could tell what filesystems were used for home directories. I mounted one of these filesystems and discovered that since I had root access on my machine, I had root access on the mounted filesystem. I could now write my own **.rhost** files to give me complete access to any of these users accounts.

The first thing was to checked to see which machines were "personal workstations." Often there is an entry in the **/etc/hosts** or HINFO DNS-record to describe who this machine belongs to. If there are a lot of PCs and only a few "workstations", these probably belong to the system administration group. However, if everyone has a workstation, this trick doesn't work.

Since I could now look in the **/etc/passwd** file, I found out who were the system administrators as this was written in clear text in the GEOS field. I then found out what filesystem their home directory was on and mounted that via NFS. I could then edit their .rhosts file to give me access to their account.

Using the same information this told me who the system administrators were and what areas they were responsible for. I could then concentrate by attacks on their accounts. As the system administrator you should know who the other admins are. There is no need for the users to know this. In my opinion, there should be nothing in the password to identify the user. If you need this information regularly, put it in a file somewhere that is not world readable.

Having access to their account doesn't necessarily mean I have root access. However, it does mean that I have access to an account that sooner or later will want to get root access. More than likely this will be with the su command. With write permission to that user's directory, I could trick them into giving me the root password. I could create a "Trojan Horse" version of su that comes first in the user's path (maybe changing the path if necessary). The next time he uses su I have the root password.

12.7 What You Can Do About It

If you are the system administrator of a Linux system and security is even a minor issue, then you definitely need to read "The Cuckoos Egg" by Cliff Stoll and Internet Security and Firewalls by Cheswick and Bellovin. Although we have covered some of the issues that they confronted and the techniques they used to monitor their intruders, there's nothing like

reading it yourself. Plus, if you hear the true stories, they sink in better than hearing just the theory.

"The Cuckoos Egg" reads like a spy novel and is difficult to put down. Even though I knew the outcome before I started reading, it is difficult to put down. I say "is" because I am in the middle of it reading as I write this.

Watching Your System

In the preceding paragraphs, I detailed many of the holes that are used to break into a system. I also addressed the methods that hackers use to gain information about your system to exploit these holes. In this section, I am going to talk about specific methods people have used (including myself) to circumvent normal security.

One aspect of watching your system that can cause the most problems is what to do when you do see that someone is hacking your system. Remember that in many places the mere fact that someone has gained unauthorized access to your system, they have committed a crime. Like any criminal, they will want to cover their tracks. If you let them know you have caught them, they might end up removing all the files on your hard disk (rm -rf /) and then disappear.

Takes a look at the holes we talked about in the previous section. Use those as a guideline for determining what security measure you want to implement on your system.

Accounts

User accounts should be monitored and inactive ones should either be removed or disabled. "Inactive" should be defined by the company's security policy (e.g. three months). Users should be contacted by telephone and told that they need to come *in person* to get their accounts reactivated. All accounts must have passwords on them. If possible, configure the system to disallow null passwords.

User account areas (home directories, etc) should be monitored regularly to check for possible compromise. This includes removing or monitoring the contents of .rhosts and .forward files. These files need to be owned by the account for which they are in the home directory and permissions set to readable by the owner only (permissions 600).

Require that new user accounts be requested by the persons supervisor or someone else known to the system administrators. You don't want someone calling up and saying that they are new in the Accounting Department and need a new account. The request can be done via email, but confirmation of the request should be done telephonic in cases the supervisors account was compromised. All accounts, as well as changes to groups and permissions must be requested by the supervisors.

The root/administrator account should be the only account on the system that is shared. Only users that have a need should be given access to this account. Since the root password should be different for all machines, is then possible to give root access to only those machines that are necessary.

All guest accounts should be removed from the system. There is no need for a guest account. You should know in advanced that someone will be using the system and you can create an account for them. This limits access to the systems as well as provides a record of activity.

Monitor accounts that are no longer "active", as break-ins are less likely to be noticed. The "Cuckoos Egg" hacker used an account from someone who was on an extended leave. Since Cliff Stoll was aware of this, he knew that whoever was using the account was doing so "improperly". One alternative would be to simply remove the account. When the user returns a new account can be generated. If the person leaves the company then the account should be disabled or removed.

Know who is on vacation and consider disabling their account. Depending on the system, you could set up an at job that turns their account off the last day before they go and turns it back on the day they return. If that is not an option, occasional checks of the system to see if one of these people is logged in might provide clues to a break-in.

Many software products will create their own users. Be careful of these. Make sure you are aware of exactly what their purpose is. If deleting is not possible, make sure that they have limited access to the system. If there are guest accounts on your system and they are not needed, delete them.

Make sure that all accounts have passwords. If the system allows null passwords or simply the enter key, run your password cracker at least once a day to make sure.

Avoid group accounts, other than root/administrator. You can accomplish the same goal by placing everyone in a group and giving access permissions to that group.

Depending on how sensitive your data is, you might consider setting alarms on system an accounts for when they are accessed at "inappropriate" times. What these times are and who can access the system should be specified in your company’s security policy (see below).

You can also get users to monitor their own accounts. By using the last command you can show the last time a user was logged in. By having the user check this themselves, you obviously save yourself the trouble, and they know better when they were logged in. Fortunately, this information is provide for you each time you log in. Therefore you can have your users check this and report any inconsistencies.

Passwords

Words that can be found in a dictionary are not good choices for passwords. With just a few lines of code, you could write a simple program that searched through a list of words and tried them all as passwords. However, if activated, Linux will prevent you from choosing any of these as your passwords. It also prevents you from making simple changes to the password like rotating (*strawberry* becomes *awberrystr)* or reversing (*yrrebwarts*).

The passwd program source is relatively easy to modify to add all of the features we discussed. When checking the validity of the password, the program could first check to see if the input was very obvious things like the users name. Next, a dictionary containing common words could be scanned. The input password could also rotated and reversed to see if they

match anything on the "no-no" list.

Some systems have added the ability for the system to generate a password. This could be anything from generating random characters, like *rY3h%n0&* to combining random syllables like *bofandi*.

If your system doesn't allow you to select passwords for your users, you could regularly run a password cracking program. If you are successful in cracking a password that password ***must*** be changed immediately. Simply mail the user saying that it has been determined that the password selected is unacceptable and must be changed. Never include that password in a message.

Password attacks are perhaps the most common way of getting into a system and not bugs in the system itself. Studies have show that unless the system stops "bad" passwords, password guessing *will* eventually succeed. The hackers in the "Cuckoos Egg" used the same techniques I did to crack passwords and gain access. As Cliff showed, known or assumed accounts names and guesses at passwords succeed amazingly often.

Here are some guidelines when dealing with passwords:

Don'ts

- Don't use your login name in any form (as-is, reversed, capitalized, doubled, etc.).
- Don't use your first or last name in any form.
- Don't use your spouse's or child's name.
- Don't use other information easily obtained about you. This includes license plate numbers, telephone numbers, social security numbers, the brand of your automobile, the name of the street you live on, etc.
- Don't use a password of all digits, all the same letter or keyboard patterns like qwerty. This significantly decreases the search time for a cracker.
- Don't use a word contained in (English or foreign language) dictionaries, spelling lists, or other lists of words.
- Don't use a password shorter than six characters.
- Don't use the same password on multiple machines.
- Don't use a password that has appeared in any published work as being a "good" password.
- Don't ever use your password again, if it is discovered.

Do's

- Do use a password with mixed-case alphabetics.
- Do use a password with non-alphabetic characters, e.g., digits or punctuation.
- Do use a password that is easy to remember, so you don't have to write it down.
- Do use a password that you can type quickly, without having to look at the keyboard. This makes it harder for someone to steal your password by watching over your shoulder.
- Do change your password often.
- Do make remembering the password easier, so you don't have to write it down.

- Do choose a phrase and use the first letters of that phrase. You could also use a line from a song. For example, the first line of "Yellow Submarine" is "In the town, where I was born" would become: Ittwiwb.
- Do use some nonsensical word like slewblue
- Do combine words with some punctuation in the middle: rain;drain, lemon?curry

If you are a system administrator consider running something a password cracking program at regular intervals to see if users are actually using good passwords. Do not allow them to us them by replacing the password program on those machines where possible.

Keep Your Eyes Open

A prefect crime is more than just one where the perpetrator gets away clean. It is one where the crime is not even detected. If an intruder can access a system *undetected* he is safe. If you do detect an intruder, your company security policy (see below) should detail what to do. If you are monitoring his activity to see what other machines he is trying to break into, don't let him know you are there. If he is clever enough, he might have built in a backdoor, like one of those we discussed earlier.

Certain auditing packages like COPS will monitor and report on changes to key files. Even a shell script that simply compares values is sufficient to catch these kind of changes. Since hackers are aware of these kinds of tools, it is not a good idea to run them automatically from cron jobs. A hacker could look in the cron tabs and see what programs are being executed and either disable them or work around them.

Another thing you can use is SATAN (System Administration Tool for Analyzing Networks). This is an interactive, complex application that checks a wide range of security "issues." Although it didn't find any more security holes than I did by hand (in fact, I found more), it doesn't matter. SATAN is based on HTML and perl. You have all the source code and you can quickly expand it to exploit other holes that you know about. The problem is that as of this writing, certain browsers give it problems. You may have to change the way the browser reacts to the perl scripts. Its available at a lost of places, such as *ftp://ftp.win.tue.nl/pub/security*.

Know your system. Know what kind of activity is normal for every hour of the day. Imagine it's late Friday night and you know no one is still working. However one computer is busily working on some process. Is it an 'at' job that someone started? Or is it a crack program that's going through a password file? This is how one system administrator *was* able to detect a person trying to crack passwords.

What processes are normal? If suddenly a new program appears on your system and you are the only one who has access to a compiler or can install software, where did it come from? What processes run with UID of 1? Is someone's shell suddenly starts running with a UID of 1, you know you have a problem.

Excessive processes can result in a *denial of service*. That is, the system is so busy doing work for the hacking, that it doesn't have time do do other things. Although you can limit the number of processes each user has, if those processes are disk intensive, a hacker could bring the system to a standstill. If the hacker were to keep writing to the file system, you could run

out of space or inodes which might cause the system to panic. Even if the system doesn't panic, cleaning up after this will cost a great deal of time and money.

Filesystem Security

Knowing what the permissions should be is useful in detecting intruders or other improper activity. If the permissions on files (particularly programs) get changed, you should know why. This is especially important if the files are SUID. If a program is owned by root and changed to be SUID, this could allow someone improper access to the system.

Fortunately, the rpm database has much of the necessary information. Among the information stored is the permissions of the files, owner, group and a checksum. Well go into details on using rpm to check this later on.

You should also check the write permissions on all system directories and files. If an intruder has write permissions on a system directory, he can change log files or add his own version of system programs. While you're at it check the ownership of system directories as well. It does little good if no one but the owner can write to a file, but the owner is a normal user.

In principle, no one should have write permission to a users home directory other than the user. If some else has write permission, they can overwrite that users .rhosts file. Even if the file is write protected, write permission on the directory means the file can be erased and a new one put in its place. You should also check the existence and content of .rhosts files to ensure that they do not give too much access. Obviously, if .rhosts are not allowed at all, they should be removed.

I also recommend that you be aware of every SUID or SGID program on your system. Know what it is there for and why it should be SUID/SGID. If you know that you won't need it, consider removing it or changing the permissions. Also, check ownership of all system directories and files. There are some files on the system that need to be writable by everyone. Make sure you know which ones they are so you can see if there have been any changes.

Look for files without an owner. That is, the owner in the inode does not have an entry in /etc/passwd. This could be innocent, when a user has one UID on one machine and another UID on other machine. Using cpio or tar to copy files, the UID of the source is copied to the new machines. This happened to me once, but maybe there is something else behind it. Both -nouser and -nogroup are options to find so it's easy to hunt for these files.

Check *specifically* for "wierd" filenames like "..." (three dots) or "..(space)" or "..(backspace)" or anything that might be unusual. It is "possible" that these files were created by accident, but they are common ways of hiding files on a system. Someone could also create filenames with control characters in them. This could help mask them. On most systems, the **`ls`** command has an option (e.g. -q) that will print out the directory list with a ? instead of the control characters.

If you system does not have RPM compatible packages, you should create a list of important information, prior to adding any users. Make a complete list of your entire system. Include the owner, group and permissions. For binaries and other non-writable files, get sizes and creation dates. Include the sums, using the sum command or create M5D checksums using one of the

tools available on the net. These should never change. Maybe the inode number itself is important, as well. If the inode is different, that means the file was copied.

Once you have your checklist, move it someplace away from that machine. It should ***NOT*** be stored on the local machine. If a clever hacker gets into the machine and finds this list, what's to prevent him from changing it so it matches the modifications he made to your system?

Devices nodes are one group of files that are often overlooked. Check access permissions on device nodes like mem, kmem, hard disks, tape drives. If the intruder has write permission on /dev/kmem or the hard disk, he can change things directly without using the standard tools. In addition, there is rarely a reason why device nodes should exist anywhere other than in **`/dev`**. If you find one, find out why it's there. Check the major and minor to see what kind of device it is.

The Network

If you provide Internet or any network services you should monitor these as well. Remember that treats do not need to come from outside. Disgruntled employees or someone who has been bribed your competition can compromise security just as much as someone from outside. Good security does not mean pulling the plug on all network connections, but it does mean taking a few simple precautions.

12.7.1 Trusted Hosts

Trusting other computers is a two edged sword. Many systems that disallowed trusted hosts did well against the Internet worm, compared to other sites that did not. You need to specify in your company's security policy just what kind of access is allowed. Maybe it's the extreme where everyone trusts everyone else. Maybe it's the extreme that no one trusts anyone. The middle ground would be to say that the database server trusts no one, although the database server is trusted by the others. That way if one machine is compromised, the database server is safe.

You need to weigh convenience with security. When I was able to crack the account of one system administrator, he already had an .rhosts file that allowed access to his account on every machine from every other machine by both his own account and root. Therefore, once I had broken into one machine using his account, I could break into all of them.

If you are setting up a system for the first time, you need to define you access policy before you hook up the machine to the rest of the network. Once on a net work where security "can" be broken, the new system is no longer secure.

If you are taking over a system, you need to check it to make sure that it adheres to both the security policy and common sense. Check /etc/hosts.equiv to see who is given access and *every* **`.rhosts`** file on the system. Make sure that they are what you want. Never allow wildcards of any kind. Make sure that you specifically define who has access and from what machines.

One common mistake is that the **`.rhosts`** file is world-readable. No one should be able to figure out what access another account gives. Just because someone knows what other machines can reach this one does not mean that he can access that account. However, the more information an intruder has, the more directed the attack and the greater the chances of success. Fortunately, the remote-command/login functionality does not work on most newer Linux distributions if the .rhost file is readable by others.

12.7.2 FTP

Anonymous FTP should ***not*** be made available on every host on the network. Choose one machine (preferably a server or standalone host) that is protected from your internal network. This can be the same machine as mail or WWW server. This makes monitoring for security violations much easier. In the section on configuring a Internet server, we go into details about securing your ftp server. Here, I'll just cover some basic issues.

Incoming transfers to this server should be in a separate directory (i.e. incoming). This is the *only* directory where the user ftp can write. However, they cannot read this directory. This is to keep your site from becoming a repository for pornography, pirated software and other nasty stuff. Check often the contents of the directories into which ftp is allowed to write. Any suspicious files you find should be deleted.

Although the ftp directory should not be writable by the ftp user, you should still check for "hidden" directories or files. Review what is being abused to take appropriate action, based on what your security policy says. If you can determine where the stuff is coming from, notify both CERT(Computer Emergency Response Team) and/or that site. If you can't find a phone number for that site, do not send the system administrator email. If the other site is compromised, the intruder may check through the email files.

12.7.3 NFS

NFS, by it's very nature is insecure. One of the basic premises is that you are a trusted machine to begin with. A major flaw in NFS security is that it is name based and not based on IP address. Hostnames can be easily changed, which is an even bigger problem when access is granted to machines without domain names.

If it's not properly secured, NFS can be used to gain access to a system. You need to be sure that the filesystems that you are exporting do not allow extra permissions and that you allow access to only those machines that need it. Be specific about who has what access.

I don't recommend that any filesystem be accessible by the world unless it's completely harmless and read-only. Even then, you could still provide the files via anonymous ftp and limit the potential for compromise. An example would be your man-pages and other documentation. It might be a good idea to share this directory to every system in an effort to keep things consistent and to save space.

Even if you do implement such a system, you should not export it to the world. By making the filesystem(s) accessible to only specific machines, you limit the potential for compromise. You know exactly the consequences of what you did. By using wildcards and making the

systems available to everyone, you can't be sure of can happen.

Even if you set up your NFS "correctly", you should check the configuration at regular intervals. If your system has been compromised it would be a simple matter for someone to add an entry or change on to give him access. The showmount command will show you a list of machines that are currently mounting your filesystems. You should use this to check to see just who is accessing your system.

Check the /etc/exports file at regular intervals to ensure that you exporting only those directories that you think you are. Although it really is dependant on your company, the safest thin is to only export directories and filesystems to machines within your local domain. If you have machines outside of your domain, implementing a firewall that allows NFS is more difficult. Besides, I have yet to hear a convincing argument as to why it should be done at all.

The showmount command shows machines currently remotely mounting your filesystems. Only local machines should appear here. Monitor this. Only "normal", non-system directories should be mounted and they should be read-only if possible.

You can find details of setting up NFS here.

12.7.4 Modem Security

Is access to you machine possible by modem? I had worked for one company for more than year before I found out that there was a modem on the system. It was connected to a terminal server that has it's own password, so you actually needed two passwords to get into he system. However, this is important for every system administrator to know.

What are the characteristics of the modem and the port? Is hangup forced when user logs out? If the connection is broken does the system log the user out? What are the permissions on the port? Can it be used by normal users to dial out? Are the answers to these questions in keeping with your company security policy.

12.7.5 Backups

Your system backups are an integral part of your security policy. Not only are they useful when the system goes down, but can be helpful in and investigation (see below). One thing to consider is how long to keep your backups. If an intruder gains access to the system and does nothing for a month, do you have a clean backup from *before* the break-in? Do you have a copy of a clean system?

In one company I was in, we had five tapes for each machine, one for each day of the work week. We then got a tape loader that could hold enough for two weeks. However, each August there was a company shut down for three weeks. Several people from the IS department as well as some people in sales and customer service continued to work through the vacation. Therefore, regular backups were done. What would happen if someone came back from the three week vacation to find a file missing? There is no backup old enough to find the file!

In addition to the security aspects, doing regular backups is an integral part of administering your system. For details on backup strategies, see the section on backups under Basic Administration.

12.7.6 The Official Word

There are several organizations and agencies that deal with computer security issues. Perhaps the most widely know is the Computer Emergency Response Team CERT at Carnegie-Mellon University. They serve like a clearing house for known security problems for most common operating systems. They regularly issue CERT Advisories that detail the steps necessary to correct security problems, without revealing too much about how to use the problem to break in. For details, check out their web site: *www.cert.org*.

One organization that is vital for the security of your system is your own management. They have to take an active, if not pro-active stance in promoting security on your system. It is up to them to define what security means for the company and how important it is. In addition, the must give you, as system administrator, all the tools necessary to put these goals into effect.

Security and Network Policies

A security policy is a set of decisions, that collectively determines organizations posture toward security This not only includes what is and what is not acceptable behavior, it also defines what actions are taken when the policy is violated. A network policy defines what is acceptable when using the Internet. They cover different areas, but are very much intertwined.

Before you define a security policy you need to define your security stance. This is more or less decided by your company's attitude on security. If you believe that everyone should have access to everything and nothing will be limited, your security policy will be significantly different than if you want security above all, no matter how inconvenient it is for your users.

It's often difficult to define what is considered a "acceptable" behavior. Some companies give their employees the freedom to hang themselves. That is, they have complete access to the Internet, including email, WWW, ftp and so on. If the company discovers that they spent all their time downloading games and not working, they get a warning, a reprimand and finally termination. On the other end of the scale some companies say that a computer is for company business and will not be use *at all* for personal use, even if it means you can't get email from your brother.

One thing I feel should be in there not matter what end you are on is that you must clearly state that employees' activity on the Internet should present the "proper" image for the company. I had to put the word "proper" into quotes, because this will obviously be different from company to company.

I worked in two places that were very similar on the surface. Father-Son businesses, both with about 1500 people worldwide. One was very rigid and formal "Good morning, Mr. Smith and the other was very laid back "Mornin' Tom, how's it going?" What was proper in one place was not in the other. On a business trip to Australia, I was told that when you call someone Mr. or Mrs. you are angry or upset or want to be sarcastic.

The first step in defining either your security or Internet policy is to *define* what is and what is not permitted. Spell it out in clear text, so that everyone know what it means. In order to make things easier and perhaps the list smaller, you could simply defined the "don'ts." Define what is *not* permitted. This could include the hours during which Internet activity is not allowed and the types of material cannot be brought into the company i.e. pornography, pirated software.

Also part of the security policy should be what protocols and programs you are going to allow. If you are only going to allow outbound connection, then the policy should state this. If inbound connection are okay, what protocols can be used? Are incoming ftp and HTTP connections okay, but not incoming telnet? If so, this needs to be spelled out in the security policy.

A key aspect of your security policy is your stance on passwords. If you have decided that passwords are to be a specific length and cannot have specific contents such as the user's first name or spouse's name this needs to be spelled out.

The policy should also define the system administrator's responsibility. On a Linux system it's a simply matter to change the source code to the passwd program to check a list of unauthorized passwords or do some manipulation of the password so as not to use unauthorized passwords, but spelled backwards. If necessary, the security policy can state that it is the system administrator's responsibility to ensure that such password *cannot*be used. This can be easily accomplished by using the npasswd program.

Have your company management sign a password security policy and make all employees sign it as well. This policy should specifically define what is unacceptable behavior when dealing with passwords. Make sure that the employee is aware of the consequences of violating this policy such as letters of reprimand and even immediate termination. Users must be told that they will be held accountable for action taken by anyone using their account.

At first, termination might seem a little harsh for a person who gives is password to someone else in the same department, for example. However, there is no need to. If that other person really needs access to the data, either the permissions on the file should be set or the file should be copied to a common area. If access to the account is necessary, have their supervisor or someone else who is known to the system administrators call. The sysadmin will either copy the file, change permissions or change the password to something known in accordance with the company password policy. This password will them be changed again when the account is no longer needed.

Users must keep their passwords to themselves and must never be written down anywhere. This includes blotters, calendars, post-its and especially in files on the computer. The hacker in The Cuckoo's Egg scanned email files and found one where the user was telling a co-worker his password.

Users must change their password from time to time. Certain dialects of UNIX can force users to change their passwords. If the version of Linux you have cannot, you could implement a program that checks for specific dates and then notifies users. One possibility is to send mail to half the employees one month and the other half the next month.

However, users must know to never reset passwords to *specific* values based on email they have received. This would prevent a hacker from compromising the mail system and send a message to an unsuspecting user. Would your users be able to recognize mail if it doesn't come from a real administrator. All your mail should do is say that the password time has expired and that it should be changed. If the user gets a message to change it to a specific password, then it didn't come from an administrator.

12.7.7 Changing Attitudes

Although your company has a security policy, you need to concentrate more on changing people's attitudes. Perhaps a violation of the policy leads to someone's termination, but does that recover the millions of dollars of research that was lost?

If a user chooses and easily guessed password, then it will be cracked using a dictionary attack. No question. Even if the hacker only has access to small, low-powered PC, he can quickly crack the password. Many users believe that if a password in not in the traditional UNIX dictionary file (/usr/dict/words) then it can't easily be broken. However, there are dozens of dictionary files spread out all over the Internet that contain lists that are much longer. In addition, the words are not limited to just English anymore. There are dictionary files for several other languages, as well.

In his paper "Foiling the Cracker: A Survey of, and Improvement to, Password Security," Daniel Klein of Carnegie Mellon University reported that during tests he conducted 2.8% of all passwords were "guessed" within 15 minutes. He further states that on a machine with 50 accounts, at least one will be cracked within the first 2 minutes! Without user support the number will be a lot higher.

As system administrator or IS manager, you have to educate your users. Explain the need for the passwords and security, in general. Make them aware of the real cases where laxed security had detrimental effects. Be sure that they know the dangers are real.

One thing I found useful was making comparisons that the user understands. For example, compare the inconvenience of having difficult password to the inconvenience when the system crashes. It might take 5 seconds a day longer to type in the correct password, but if the database is down for two hours, then the user could have typed their password 1440 times. In other words, once a day for almost four years.

Another comparison that works well is that of car keys. No one would think of leaving their car unlocked, let alone change the car so that an ignition key is no longer needed. It is just as inconvenient to have to use keys to a car, just as it it to user password on a computer account. It's just a necessary evil.

Finally, there are threats. I don't mean holding a gun to their head and force them to use good password and follow good security practices. Your security policy should state the consequences of giving out passwords or letting other gain access to your account. Users should be aware that they *could* be held legally responsible for anything done on the system with their account. Especially if they are negligent.

For example, check TFTP, (Trivial File Transfer Protocol) which is often used to transfer files automatically. My suggestion is to disable it completely. There is nothing that can't be done with other means and the risks are too great. If not, there is the potential for accessing files on your system without any password at all.

One significant file is **`/etc/passwd`**. Since is it is world-readable, if TFTP is enable, someone could easily download this file without a password. Once they have it, they can use a dictionary attack to try and crack some of the passwords. Another way would be to copy .rhosts files into users' home directories to gain access to the system.

Another useful tool is rpcinfo. This communicates with the portmapper daemon and provide information about what kind of services are being run. One very dangerous service is NIS. Although useful in propagating passwords to other machines, a clever hacker can "persuade" NIS to give him a copy, thus making the system vulnerable to dictionary attacks (among other things.) Although you need to know the NIS domain name, it is much easier to guess than users' password as it is more than likely some variant of the company name or the Internet domain.

There is no way to make a computer completely secure, other than lock the room and turn the computer off. Systems can be made impregnable to the casual intruder, as well as make it more difficult for the experienced cracker. However, there are no guarantees with it.

12.7.8 System Security

In early versions of UNIX, account passwords and file permissions were the only types of security implemented. As computers became more widespread and those who wanted to gain unauthorized access became more devious, it became apparent that this was not enough. Since the US government was steadily increasing the number of agencies that had computers, the level of system security needed to be increased as well.

In 1985, the National Security Agency's National Computer Security Center (NCSC) created a set of computer security standards for the Defense Department, titled "Trusted Computer Systems Evaluation Criteria". This is commonly known as the "Orange Book" as it was published with an orange cover. (This is part of a series of documents by the DOD related to computer security, all with different colored covers.)

Within the Orange Book, there are four broad classes of security levels for computers:

- D Minimal security
- C Discretionary protection
- B Mandatory protection
- A Verified protection

The C class contains to sub-levels, C1 and C2, with C2 offering slightly more security than C1. Class B offers three sub-levels: B1, B2 and B3.

Traditional PC based operating systems, like DOS and Windows fall within class D. This minimal protection does not mean there is no security, just that it is not as high as the C class. You can buy add-on products to add passwords to your system or change the file attributes to prevent accidental erasure. There are even product available that will allow you to add passwords to DOS and Windows systems, but that's about it.

Class C systems include the features and functions to employ *discretionary protection* . That means that it is up to the system administrator's discretion to decide how much access people have. Class C1 systems offers enough security to let users keep their data private from other users and prevent it from being accidentally read or destroyed. As we've already talked about, standard UNIX already provides this level of security in the form of user passwords and file permissions. Class C2 demands tighter login procedures, auditing of security related events, and isolation of system resources.

B-Class systems implement *mandatoryprotection*. That is, the system administrator *cannot* turn it off if he or she likes. Class B1 systems have *labeled protection*. This means that security procedures and *sensitivity labels* are required for each file. (A sensitivity level is basically a security classification) Class B2 adds the requirement that the system must be able to account for every code in the system. This helps to prevent security holes such as *Trojan horses*. Class B3 deals with the security of data access, in terms of prevent tampering and notification of security-relevant events.

The most secure class, Class A1, requires *verified* designs. Although they are functionally the same as B3 systems, A1 systems have also been formally defined, as well as *proven* by tests.

For years, the orange book was seen as the bible for computer security. Often people would see a system that followed the guidelines specific ed for a C2 level of trust and call the machine C2 "secure." This is a misnomer. The machine is *trusted* to provide a certain level of security, but it is not "secure"

Recently, groups in several countries have gotten together to update the guidelines defined by the orange book. They have developed the "Common Criteria," which is a standard for security *criteria*. These countries are Canada, France, Great Britain, the Netherlands, Germany and the US. Acceptance by these countries has made this, more or less, the de facto standard for information technology security worldwide.

Two of the more important basis documents for the Common Criteria (CC) is the orange book and the Information Technology Security Evaluation Criteria from the Commission of the European Community (ITSEC). However, the CC is not just a synopsis of other documents, but rather it's planned that the CC will replace these other documents.

Two of the key concepts in the CC are the *protection profile* and the *security target.* The protection profile is not product specific, but after being reviewed, it becomes part of the CC. It documents a particular IT-security problem and the appropriate solution. For these the requirements for specific product types can be developed.

Security targets allow protection profiles to be fit to a specific product. In other words, the product as a particular goal, in regards to security. With this, the security target forms the basis of the evaluation. A product evaluation determines whether a particular product has properly identified and addressed a particular IT-security problem.

The CC will be expanded as needed. The version planned as of this writing will contain requirements for cryptology. Cryptology solves problems of confidentiality, data integrity, and verification. The first version already addresses the issues of data protection and secure communication, even over open networks.

The evaluation process has several stages. First, a product manufacturer identifies an IT-security problem and decides to develop a solution and wants to have it evaluated. If a protection profile exists for this problem, the manufacturer can fit the profile to the product through the security profile.

If there is no security profile, a new one can be developed and a standard established to measure similar products. However, a security target can be defined without reference to a protection profile.

First, the security target is evaluated according to the CC. Then the product itself is evaluated according to the security target. If the product passes the evaluation, it is given an Evaluation Assurance Level (EAL). The evaluation, which is conducted by an organization *independent* of the manufacturer confirms that there are no obvious security errors. In the case of a higher EAL, the evaluation confirms that there are no hidden errors. Also the evaluation confirms that there is user documentation.

One of the advantages that the CC brings is that it is flexible and provides a clear concept of security. Products that have been evaluated and certified by the CC will gain significance and acceptance. The costs resulting from the evaluation process will be compensated by the improvements to security as well as the increase in market demand for certified products. As of this writing, most of the protection profiles deal with network issues. However, because of it flexibility, the CC can be implemented is other areas.

For the current version of the CC, check out the web site of the Nation Institute of Standards and technology at: *http://csrc.nist.gov/nistpubs/cc/*.

12.7.9 Security and the Law

The laws governing computer break-ins are going to differ from state to state and country to country. Although there are now federal laws covering break-ins, they only apply to the United States. What about hackers that come in from other countries? Cliff Stoll can tell you horror stories of the problems he had.

One thing Cliff did was to take very careful notes of the intruders activities and keep print-outs of the hacker’s activity on his system. What made this useful in court is many aspects that he was very careful about how he handled the evidence.

There are several guidelines to follow if someone breaks into your system. The first thing is to contact CERT and your local law enforcement agency. Both will give you guidelines on what to do.

One of the first things that the law enforcement agency will do is to determine whether a crime has been committed. Although federal law says that the mere fact someone has gained unauthorized access to your system, they have committed a crime, there may be other issues involved such as theft of trade secrets, lost in work, etc.

Because of the federal laws involved, the FBI *might* have jurisdiction, or at least want to be involved. However, I recommend contacting your local authorities first and let them determine if the FBI should be involved. Additionally, the local authorities can provide you with information on how to proceed.

One thing that the law enforcement authorities will help you with is evidence collection. Maybe you know your system inside and out, and have monitored the intruders activities, but that does not mean what you have would be considered valid evidence in court. Your local authorities can tell you how to handle things correctly.

If information has been stolen, then you are going to want to find out what it was. This is important in estimating the financial losses for unauthorized disclosure. As an extreme example, let's take a case where an intruder steals plans for a new machine. You had planned to patent it, but since your system crashed, you are delayed. Although it would be foolish for a competitor to try and patent it themselves, they could publicize your research to destroy your competitive advantage. Therefore, it would be much more difficult to obtain a patent yourself. The amount you lost in royalties are *real* damages.

If you decide to pursue the issue and press both civil and criminal charges, you have to be willing to make a commitment. The police (or whatever agency is involved) cannot do it alone. They need your help in terms of both time and resources. They need someone there to show them the logs, identify the data that has been stolen as well as identify any evidence that is found in the hands of the intruder. Even after the intruder is caught, you will still have to spend time to support the investigation such as identifying data or appearing in court.

Unless you live in large metropolitan areas, there is a good chance that your local authorities may not understand the technical aspects of the crime. Basic concepts like data and networks are something they probably heard about, but understanding them is something else. There are just too many kinds of crimes for them to be experts in them all. Even if they have one computer crime a year, they just don't have the experience. Therefore, you may have to explain just what root access is and what the extend of the access/damage could be for someone with root privileges. In others area where crimes are reported regularly, there are special units that deal with these types of crimes.

Obviously, if you can't prove "who dunnit" then there is no way to collect any compensation. That is why it is vital that the rules of evidence be followed. Although the police can give you specific guidelines, here are few points to consider while you are waiting for the police to arrive.

However, do not let this discourage you. In most places, there is a difference between criminal and civil charges. In a criminal case, the prosecution has to prove their case *beyond a reasonable doubt.* In a civil case, the plaintive needs to prove *preponderance of evidence.* This means that some one can be declared not guilty in a criminal trial, but still be held liable in civil case. Look at the O.J. Simpson case as an example.

First, if the only evidence you have is based on on-line information such as files in the user's directory or email messages, you are on thin ice. Just as an intruder can steal files, he can also plant evidence. Although this kind of "evidence" might be sufficient to get a warrant to search the suspects house. This might not be enough to prove the person's guilt.

It might be sufficient for you to use this information as grounds for termination of an employee. But you must also be careful. Is there a reasonable expectation of privacy when sending email or storing files? If it is company policy that *anything* on the computers is company property, then you may have a case. I have worked for companies that have said email will not be read by *anyone*. There is a reasonable expectation of privacy and the company could be sued if they looked through someone's email. Here again, talk to the law enforcement agencies.

Speed is also important when gathering evidence. Maybe an intruder has used one machine as a storage house for information that he has collected from other machines. Copies of all the files and try to maintain the directory structure. This *might* be useful as evidence since the likelihood that two people have the same directory structure is low (sort of like dental X-rays). If the intruder deletes all the files, your evidence is gone. There are repeated cases where password files from other machines have been found along with password cracking programs.

As I mentioned before, don't let the intruder know you are watching. The best (least bad?) thing he could do is simply disappear, maybe breaking in through some other hole that you don't know about. The worst that could happen is that the intruder reformats your hard disk in an effort to cover his tracks.

Another aspect of evidence is "chain of possession." This means that it can be proven in court where the evidence was the whole time. Who obtained it, who secured it, who handed it to the police are all aspects of chain of possession. Once you have a piece of evidence you should mark it with your initials and then seal it in a container so no one else can get access to it.

In the Cuckoo's Egg case, the logs of the hackers activity proved to be a vital piece of evidence. Cliff was able to prove that certain action on the system were made by hackers other than the one he was tracking. There were patterns to his behavior that Cliff recognized and could separate from those people who were just having a look around.

Although what I have just talked about provides the foundation for a security investigation, don't take it as gospel. Laws are different from state to state and from country to country. Talk to your law enforcement agencies *now*, before the first attack. Find out what services they can offer in case of a break-in. Most importantly, find out what the law is governing break-ins, rules of evidence and especially privacy as you don't want to lose the case and get sued yourself.

Chapter 13 Installing and Upgrading

When I first started working with Linux I had to first copy everything to floppies and install from there. There were none of the bootable CDs, with easy-to-install distributions. It took ages to load from the floppies and often I would discover something was missing and would have to either start over again, or look for the missing pieces.

By the start of the new Millenium, Linux had matured to point where is was as easy to obtain and install as any other operating system. Those who didn't want to do everything themselves, could buy a number of distributions at their local department store. Many stores were already selling complete systems, with Linux pre-installed for them.

13.1 Getting Your Own Copy

There are many different ways that you can get your own copy of Linux. It has reached the point where it is readily available in any computer store and many department stores, as well as many online shops, like Amazon.com or places that just provide operating systems, like OS Heaven.net.

Many distributions are available directly from the vendor, typically through an FTP server. For example, you get obtain copies of SuSE Linux from ftp://ftp.suse.de. There is typically a pub directory for public access and underneath that there may be sub-directories for the various releases, or as in the case of SuSE, you first have the architecture like i386 or Sparc and then the specific releases.

In these directories you are likely to find a number of sub-directories, broken down by functionality and then the individual packages. In some cases, you may be find ISO CD images. These can be burned 1:1 onto a CD, from which you can boot. For example, a "live" CD for SuSE 9 can be found here. In this case, it is a CD image of a runable system which you do not install onto your hard disk

If you download an ISO CD image, then typically all you need to boot from the CD. If you download individual packages, you will need to create a boot disk. In all likelihood, you will have a boot directory i.e. ftp://ftp.suse.com/pub/suse/i386/9.0/boot/ which contains images of boot disks, module disks, as well as instructions on where to go from here. Look in the dostools directory for a program, probably called rawrite.exe, which can be used from a DOS or Windows system to write the floppy images to a disk, which you can then use it to install. Note also that in most directories, you will find a README file, which gives you more information. There is typically also a text file which describes the contents of each of the module disks. I have never had a need to create more than one module disk, it it is possible

When you download a copy of Linux from and FTP, as well as provided with many commercial distributions, you will find one more module disks. Modules are small programs typically hardware drivers, which are loaded into the kernel as needed. When you are booting you system, you typically do not need all of the hardware drivers, so a number of them are compiled as modules. You also have the limited space on a floppy, so many, non-essential drivers are created as modules. During the installation you will need to insert the appropriate

module disk to load any missing driver.

Once you have downloaded the files, the next step to read any available documentation, particularly the README file. The next step is to make floppies using the floppy images, as we discussed above. Also make sure you have made a copy of any necessary module disks. While you creating your floppies, go through the section on preparing for the installation. This gives you some great tips and information about doing the installation.

One of the disks that you just create is a boot disk hopefully you labeled them all correctly. Insert this disk and power on the machine. If the machine is already running for example, you used the same machine to create the floppies, then I would suggest that you shutdown the system and **turn off** the power. Linux is very robust, but during the installation, I like to play it safe and start from a fresh a system as possible which for me means a cold boot. At this point you will be guided through the installation by being asked a series of questions.

Note that many new commercial distributions have an "automatic" installation mode. This makes a number of decisions on it own about how the system disks should be partition, what software to install. If you are going to use the entire computer from Linux, this is a not a bad option for a beginner.

If you run into problems during the installation, it is unlikely to be a general problem with Linux on your hardware. Rather it is more likely to be a problem with the floppies. One likely cause is bad media, which means you will have to re-create the floppy. Another place problems can occur is during the actual download. This can be caused simply by a poor connection. However, it can also be caused by one of the most common mistakes when transfering files via FTP. That is, transfering binary files as text. When a file is transfered as text, the FTP client may make some changes to the file, to make it a "text" file for the operating system to where the file is copied. Since the system does not know any better, it will also "convert" binary files, if they are transfered as text or ASCII. Therefore, you need to ensure that you always download these files in binary mode.

13.1.1 Preparing for the Installation

I have provided you with an "installation checklist" that you can print out before you start your installation. Try to refer to it as much as possible before you start the installation. Though a lot of the information won't be asked for the install, gathering this information can help you identify problems before you start. I suggest you include the complete checklist in your notebook.

You will notice that there are some very basic questions like keyboard language and time zone. It may seem almost too basic to include this type of information in your notebook. However, I speak from experience when I say that the more you write down, the less likely you are to make mistakes — even if the information is "basic".

Before you start, you need to check out your system. The very first thing you need to do is check whether your hardware is supported. I'm sure a few of you out there are groaning, thinking this is an attempt to blow you off. I talked with many customers while I was in tech support who go ballistic when I even bring up the question of whether the hardware is

supported.

With Linux, the word "support" has a completely different tone than that for commercial operating systems. Companies that produce a commercial OS usually have a list of supported hardware and platforms. If your system doesn't conform, they have the right to "blow you off." On the other hand, "support" under Linux means that there is a driver for it. Whether there is a driver in the current kernel is almost secondary. Because there is a driver, someone took the time to write it and is anxious to hear about any problems.

Commercial distributions of Linux walk the razor's edge between Linux's philosophy and that of other OS vendors. So far, I have had nothing but good experiences with Linux vendors. If they don't have a driver themselves, they often know where to get one. However, the reaction you get from them will depend entirely on your attitude.

"It works under DOS!" is a common response when people learn that some system component is not supported. However, as I have said many times, all it really means is that the hardware is probably not broken. I say "probably" because I have seen defective hardware work under DOS, but not on UNIX. Under DOS, a lot of hardware is accessed through the system BIOS. The BIOS is often built especially for that machine. It understands the hardware in a way that Linux doesn't. Because the Linux device drivers access the hardware directly, the hardware has to behave in a standard way. Otherwise, the device drivers don't know what to expect.

Users have also commented that the driver works under another version of Linux or even another dialect of UNIX. If it works under another dialect of UNIX, the driver for that piece of hardware was probably provided for you. If it works on another version of Linux, maybe that version has that latest kernel. Because the jump has just been made from 1.2 to 2.0, there are a lot of new drivers. Make sure that you have the correct release.

Does this mean that your no-name hardware won't work? Not at all. I have one machine that is running a fair bit of "unsupported" hardware. Much of it is clones of supported hardware which causes a lot of grief. However, it works. When I tried to install something that wasn't supported and it didn't work, I wasn't frustrated because the unsupported hardware wasn't guaranteed to work. Well, I was a little frustrated, but I knew to expect it.

There is also the issue of conflicts. Linux is good about enabling you to install a wide number of cards at their default. The common place for conflict is with multiple cards of the same type, such as more than one SCSI host adapter. However, with the list in front of you, you will be able to confirm this before you try to install and something goes wrong.

Once you have installed the operating system and it works diligently for six months, then the first problems may crop up. Now, what was the model of the hard disk? Rather than digging through a box of papers looking for the invoice, you will have it right in front of you in the checklist. Okay, knowing that you should fill out the installation checklist is easy, but knowing what to put in each entry is the hard part.

You can install Linux in several different ways, depending on the distribution you bought. If the Linux distribution on the CD-ROM does not have exactly what you need, knowing about the different versions available might help you decide which one is best for you. Note that the version of the kernel that installs on your system will *not* be the latest, 2.0. However, you will

find a copy of the 2.0 on the CD-ROM, as well as instructions on how to upgrade it.

An important thing to consider for the installation is the installation media. If you want to install Linux on an older laptop that has neither a CD-ROM drive nor a network connection, then you probably need to think about doing installing it from floppies. Many Linux versions will allow you to do a network install via PPP, so at the very least, you need a modem.

Normally, underneath the root directory is a subdirectory dosutils. Among other things, this subdirectory probably contains the program rawrite.exe, a DOS program used to write disk images onto your floppy. Also underneath the root directory, you will probably find a directory on the CD-ROM called images, which contains the images that are used to make the boot and root floppies and floppy sets used to install the rest of the packages.

In addition, at least one subdirectory contains the disk images for the various boot floppies. There is one file for any one of dozens of different hardware configurations. There is an index that you can use to select what boot disk is most appropriate. However, this is normally not necessary if there is a program to create the boot floppies for you. There is also a "bare" image that contains very few drivers.

You need to consider whether the version you have can install additional products with floppies once the initial installation was complete. I ran into this problem myself. I thought I would install the base product first and then install other components that I needed later on. However, once the installation was complete, I discovered that the only tool on the system that could read the package information on the floppies was the initial installation program. This meant that I had to either be satisfied with what I had or install again. If the distribution supports rpm, then this is not so much of a problem.

Even if you do have a CD-ROM drive, there are a couple of installation methods. With some products, you can install directly from the CD-ROM. That is, a DOS program will load the Linux kernel, which then starts the installation. In this case, you have to be able to boot under DOS and not just a DOS window. Others provide a DOS or Windows program that creates a boot floppy and, if necessary, a root file system floppy. Most distributions provide you with a boot disk so you don’t have to make one yourself. unless you get a free copy in a magazine or in the mail. Even without a diskette, I cannot think of a common Linux that does not come with a bootable CD.

During the course of the installation, you may have the choice of several different installation types, from fully automatic to fully configurable, depending on the version. For the more advanced system administrators, the fully configurable enables you to control many different aspects of the install. Fully automatic basically does everything for you, that is, it evaluates your system and essentially makes all the decisions itself. I recommend that if you are a novice administrator *and* there is nothing on the hard disk that you want to save, the fully automatic is your best choice.

Most versions enable you to select the components that you want to install. In many cases, you can select among several different "architectures" that have a set of predefined packages. If you want, you can choose a custom architecture in which you choose each package yourself. The Installation HOW-TO provides a list of the Slackware packages and what they

contain.

Take notes during the entire process and include these in a notebook. Write down everything you input and what the prompt/question was. These notes will be helpful if things go wrong and you want to try a different approach. You will know what you input the last time; this time, you can try something different.

To install Linux or any other OS for that matter, a hard disk must be divided into partitions. Each operating system may use from one to four partitions. DOS logical partitions are actually subdivisions of extended partitions. The partition table, which tells the number and location of the partitions, has been standard for years and is essentially impossible to escape, so the limit is four *primary* partitions. This means that a physical disk can have only four primary partitions and any operating sysem installed can used one or more of them.

Linux, unlike other UNIX dialects, can be installed on logical partitions, as well as the primary partitions. Therefore, it is possible to have three primary partitions and one extended partition, which then contains several logical partitions. For experiment's sake, I created ten logical partitions once, although I have never had more than 3 or 4 on an active system.

Under Linux, file systems take up the entire partition. File systems can be on different partitions or even different hard disks, or you can put all files on a single file system. Having different file systems can be safer: if one is trashed, the others are often safe; if everything is on one root, then the whole system is gone.

The more activity there is on a file system, the greater the chance it will become corrupt if the system should crash. If your database application and the data are on the root file system, all of your activity is in one place. If the database and the data are on a separate file system, there is less activity on the root file system. If the system goes down, the root file system may be okay and you can use it to fix the other problems. However, I don't mean to scare you. Disks are much more reliable today than a few years ago.

On the other hand, "small" hard disks today are on the order of 40 Gb. Therefore, you have plenty of space for multiple filesystems. Plus, there are several advantages to having multiple filesystems, such as keeping others safe when one crashes as already mentioned.

Traditionally, the way to break it up is to have **`/usr`** as a separate file system. Often, even directories under **`/usr`**, such as home are on separate file systems. This can make your backups even quicker. The root file system contains things that remain fairly constant and you need only back it up on occasion once a week, for example. Daily backups need only be done for the file systems with data on them. Plus, if you have partitions on multiple disks, performance is increased, because the disks can actually be accessed simultaneously. Also if your system crashes and the filesystem needs to be checked, the check goes alot faster, even if the partitions are on the same physical drive.

It is also common to have **`/opt`** on a seperate filesystem. In fact, since this contains "optional" software, I find that it is more common to have /opt as a separate filesystem than **`/usr`**. On very active systems, I often find **`/var`** to be on a separate filesystem. This is where log and spool files are kept. If something should go wrong and the log files become flooded with information, only that one filesystem is effected, but the system can keep going.

Also if the data is seperate from the operating system files, it makes upgrading a lot easier and safer. The filesystem containing the data can be unmount when the operating system is upgraded. It is then alot less likely that something will be overwritten.

It is also common to have temporary storage or "spool" directories on seperate partitions. The reason is that if your root filesystem fills-up or one on which an important application is running, things might come to a stand-still if there is no space left on the partition. Therefore, directories that *could* fill up quickly are seperated from the rest of the system.

Also keep in mind that the larger the filesystem, the longer it takes for it to be checked and cleaned if the system were to crash. If the filesystems are different *physical* drives, the filesystem checker, fsck, can check them in parallel, which obviously makes it faster.

On any one system using the NFS automounting facility, I had all the users on a single machine. The **/usr/home** directory was exported via NFS to other machines that mounted it on their /usr/home directory. The advantage was that the file system was only mounted when it was needed. Also, there is nothing to prevent statically mounting the home directory. No matter what machine a user logged into, he or she would have the same home directory. On many systems you might even find the home directory mounted under /.

In a nutshell here are some the basics of which directories are often mount:

/ - A stable filesystem. Few, if any changes but very important!
/tmp - Mounting /tmp protects your system if it should fill up quickly.
/usr - Contains user-related applications and programms. However, this is typically not that large.
/usr/share - This contains alot of "shared" information, like documenation and it can grow to several gigabytes. **/var** - Contains mostly logs, spools and variable information. Good choice to have mounted.
/var/spool - The spool directories. A good choice if you don't mount all of /var.
/home - User home directories. Also a good choice to mount, particularly if you do not have user quotas set up.
/usr/local - Per system or installation local files. Typically not very large.
/boot- Where boot information is kept. The reason it is often on a seperate filesystem comes from the time when LILO had difficulty with large disks.
/opt - Optional software. This is a good choice as this is where most of the user-related software goes on newer Linux distributions and it takes up a fair bit of space.

Considering the partitions size, you must also consider the swap space size as well. I've seen references that say swap should be one-and-a-half to two times the amount of RAM. Personally, I think this is too much, unless you have a good reason. The swap space should be considered a safety net, not an integral part of your memory system. If you are swapping, performance will suffer. If you think you would need the extra swap space, consider buying more RAM. That way, you are less likely to swap in the first place. We go into details in the section on the swap space.

The size of your partitions needs to be based on how much software or data you *will* have. You need to consider the growth of your system. If you put your data on a second disks, it

would be easier to backup the data, add a larger disk, and then restore the data.

If you have a larger disk, you need to be aware of the 1,024 cylinder boundary. PC BIOSs are limited to 1,024 cylinders. Until Linux is loaded and has control of the system, the BIOS is accessed. You can run into problems if any part of your kernel is above this boundary. See the section on hard disks in the hardware chapter for details.

How many partitions or filesystems you should have, is best decided before the system is installed. However, because of the way Linux works it is not a requirement. You could install the system and then add hard disks later, moving some of your files to the new hard disk. This is primary because Linux does not have the concept of drive letters like Windows. When you install your Windows system, all of the system file go on the system "drive", which is related to a **single** drive letter. Although it is *theorectically* possible to have your system files spread out on multiple drives, it is neither recommended nor supported by Microsoft. With Linux, you can put things anywhere you need to or simply want to.

At a very minimum, you must have two partitions: one for your root filesystem and one for swap space. However, it is common to have seperate filesystem for **`/boot`**, **`/usr`**, **`/opt`** and **`/var`** Details on what these directory are used for can be found in the Guided Tour section.

Even if you do not create separate filesystems for these directories when the system is installed, you can add them later. This entails booting into single user mode, mounting the new filesystems to different directories and then copying files into the new directory. You then make the necessary changes to **`/etc/fstab`** to mount the filesystem automatically when the system boots.

Regardless of how many filesystems you want to have, it is a good idea to plan everything on paper before you start. You could change things once the system is running, such as moving directories to separate filesystems. However, you can avoid some work by planning it in advance.

13.1.2 Installation Checklist

I suggest you make a copy of the checklist for each system that you install. You can fill in each entry and include it in a notebook. You will notice that there are some very basic questions concerning keyboard language and time zone. It may seem almost too basic to include this type of information in a checklist. However, I can speak from experience that the more you have written down, the less prone you are to making mistakes. Even if the information is "basic"

Note also that the mimimum space mentioned most documentation is just enough to get installed. You need to consider future growth of the system, as well. Also note that many of the details listed here may not apply to your specific distribution.

Action	Assigned Value [1]	Notes
Boot-time loadable drivers	† Yes † No	module names:

Action	Assigned Value [1]	Notes
Installation media	† From CD † From DVD † via FTP	What is the hardware configuration on the install device?
Keyboard language		
License Information		
License Number		If Applicable
Installation Type	† Fresh † Upgrade	Upgrade only valid for existing systems
System name		Must be unique within your network
Domain name		Must be unique if you ever plan to connect to the Internet
Timezone		
Geographic area		
Time zone name		
Daylight savings time	† Yes † No	
Hard Disk Config.		
Type [2]		For example: Wide-SCSI, PCI
SCSI	† Yes † No	
Host Adapter		
Bus Number		
SCSI ID		
LUN		
EIDE /IDE Controller		
Make		
Model		
Base Address		
Interrupt Vector (IRQ)		
DMA Channel		
Master/Slave		
LBA Enabled		EDIE only
SCSI Host Adapter		
Make		
Model		
Base Address		
Interrupt Vector (IRQ)		
DMA Channel		

Action	Assigned Value [1]	Notes
Termination		Where is the termination? Which device? Draw out the SCSI bus and lable everything, including termination
Hard Disk Setup		
Type	† Preserve † Use whole disk † Customized	
Hard Disk Layout		Customized or Interactive only
Linux Partition size (MB)		
DOS Partition size (MB)		
Other Partition size (MB)		
boot filesystem size (MB)		
swap space size (MB)		
root filesystem size (MB)		
Other filesystems size (MB)		
Bad Track		
Type	† None † Thorough/**destructive** † Thorough/non-destructive † Quick/**destructive** † Quick/non-destructive	See the text for a description of what each of these means.
Software	† Operating System † Graphic Environment † Connectivity/Networking † Development System † Connectivity † Online Doc (man-pages) † Source Files	You need to know how much space it will take to install all the software.
Network Information		
Network Card		
Vendor		
Model		
Type		For example: Ethernet, twisted pair, Combo

Action	Assigned Value [1]	Notes
IRQ		
Base Address		If applicable
ROM address		If applicable
DMA		If applicable
Base RAM Address		If applicable
RAM size		If applicable
Local Ring Broadcast		Token-ring only
Slot Number		If applicable
IP Address		Must be unique within your network and within Internet if you ever plan to connect to the Internet
Netmask		Based on the network class you choose. The system should calculate this value for you.
Broadcast address		
Video Card		
Make		Not all are listed. You may need to simply choose something like VGA or find the chipset that your card uses.
Model		
RAM		
Base Address		Normally not needed
Interrupt Vector (IRQ)		Normally not needed
Max resolution		
Max colors		
Frequencies supported		
Selected mode		resolution and colors
Monitor		
Make		Not all are listed. You may need to simply choose something like "other 15"
Model		
RAM		
Max. resolution		
Max. colors		
Frequencies supported		
Graphical Login (i.e KDE)	† On † Off	If on your system will start-up by giving you the graphical login. Otherwise, you must run startx, to start up X.
Mouse		
Make		
Model		

Action	Assigned Value [1]	Notes
Type		Serial, bus, PS/2
Base Address		Bus mice only
Interrupt Vector (IRQ)		Bus mice only
Serial port		Serial, mice only
Bus Mouse IRQ	† IRQ2	
	† IRQ3	
	† IRQ5	
Keyboard Mouse	† High resolution	
	† low resolution	
Email System	† procmail	
	† sendmail	
	† other	
Hard disk:		
Make		
Model		
Master/Slave		Only for non-SCSI devices
SCSI ID		Only for SCSI devices
Termination		Only for SCSI devices
Hard disk controller		
Make		
Model		
Base Address		
Interrupt Vector (IRQ)		
DMA Channel		

1 Not always applicable. Depends on the kind of device.

2 You can't put down too much information when describing the hardware.

13.1.3 Hardware Requirements

Although Linux will install and run on something as small as a 386/16 with no hard disk (you boot from floppies), you cant really expect to run your business on it. To do useful work, you have to be installed on a hard disk and have at least 4MB of RAM for text-based applications and 8MB if you are running X-Windows.

However, this, too, is pushing things a bit, and you probably can only have one user working effectively. Consider an extra megabyte per user for text applications and 2MB for X-Windows, or slightly less if the users will all be using the same applications (because the text segments will be shared).

The amount of hard disk space you need is a completely different matter. Its not as easy to come up with a rule of thumb because each site will want to have a different set of applications. The *basic* UNIX utilities and programs will fit in less than 20MB. However,

there is not much you can do with that. On the other end of the scale, I installed the Caldera Network Desktop with the Internet Office Suite onto a 500MB partition containing a complete RedHat distribution. Today, distributions like SuSE Linux come with up to seven CDs, which will obviously take up several *gigabytes* when fully installed. I then had to reinstall to make the partition larger because I ran out of space. In the middle is my laptop, on which I have a 100MB partition and almost a complete Linux installation (no X-Windows).

Most of the commercial distributions list a few example installations and how much hard disk space you will need. Every commercial product I have seen lists how much space you will need. The products that are bundled with a Linux distribution also tell you how much the OS will/can take. It is therefore fairly easy to get an *idea* of how much space you will need.

For the most part, if you have a standard PC, Linux will run on it. By "standard" I mean that the components are common brands and types, there are no clones and the hardware has been on the market for more than a week.

Linux is most commonly available for Intel machines. However, a few commercial versions are available for the DEC Alpha processor. There are also versions for the Motorola 680x0 as well as the PowerPC, SPARC, and MIPS (how many am I missing?). The best thing to do is check the Hardware HOWTO to see whether the hardware you have is supported.

13.1.4 Repartitioning

If you have an existing system that takes up the whole disk, you will have to backup and reinstall the existing system to make room for Linux. You need to be careful because you may make partition too small for all the files that were there before!

A couple of tools are available on the Internet that will reformat a DOS partition with an existing system (assuming there is enough free space). I have never used such a tool, although I have read messages on the Internet from people who have done it successfully.

During the installation, you will have a chance to either install on an existing partition or repartition the drive. As with other PC-based OSs, use the fdisk tool to partition the drive. Some distributions have full-screen, menu-driven versions, but they all perform the same functions, such as creating, deleting, and making partitions active.

However, the Linux fdisk can do much more than that. For example, it can change the partition type, which is helpful when you need to have a system with several different OSs. DOS can only create a single primary and a single extended partition. I have used Linux to create all of the necessary partitions and then used DOS to recognize them correctly.

Another useful thing is that the Linux fdisk will do what you tell it. I have wanted to delete an entire extended partition but couldn’t because logical partitions were there. Linux will do that for you.

As with other operating systems, Linux fdisk will see that other partitions are there, even though it may not know what type they really are. Linux is very good at recognizing the type, although there are a few cases in which I have seen it off the mark (but not by much). The reason is that all fdisk versions just read the partition table, which is in a common format.

13.1.5 Swap Space

Swap Basics

An older rule of thumb was to have a swap space at least the size of physical RAM, if not twice as large. However, you also need to consider growth. If you expect to increase your RAM in the future, you should consider that when you set how much space you are going to use for swap. RAM just slides into your system; increasing swap may require reinstalling the operating system, particularly if you have an older version of Linux.

So, how much do you assign to swap? Good question. In general, I still support the suggestion is twice as much as RAM. This is the "good reason" I mentioned for having more swap than physical RAM. Creating a large swap space is easier to do it now and waste the space than to reinstall later. Another good reason is when you have more than one user running graphical applications. In this case, then even setting swap to two times RAM is reasonable. If all the space is taken up on the primary hard disk, you can add a hard disk and use the swap command to add additional swap space.

The key factor is how many applications you will run and what kind of applications. You need to consider the behavior of the application. If you are doing work with some kind of graphics (i.e. graphic design, ray tracing, and so forth), then you might need to conside more swap. If you have a lot of users on your system, you might want more swap, as well.

Keep also in mind that accessing swap is slower than accessing RAM. You might want to consider adding more RAM. I have a co-worker who has 3 GB of RAM in his system as he does a lot of graphical work and it is an extreme burden on his system to be constantly swapping in and out. He never gets close to using all of his RAM, so he does not need a large swap space.

Versions of the Linux kernel prior to 2.4.10 "liked" to have at least twice as much swap as RAM. However, with the newer kernels, this is no longer true.

One important thing to consider is what happens when the system runs out of memory. Being out of memory (OOM) is a dangerous thing and can cause the whole system to come to a stand-still or even crash. Within the Linux memory management functions is something called and OOM killer which will stop (or kill) processes when the system runs out of memory. The problem is that currently (Sep 2004), processes are killed "blindly". That is, without regard to their importance. In general, the system calculates how "bad" the process is longer a process runs and the more memory it uses, the greater the chance is that it will be killed. (For details look at the file **`/usr/src/linux/mm/oom_kill.c`**).

The biggest problem is that it arbitrary. That is a program could be running correctly, not causing any problems, but it just runs a lot and uses a lot of RAM. Imagine if this was an database. The argument whether the database should be killed or the whole system stops is philosophical, so we won't go into it here. With 40 GB being a *small* hard disk today, allocating 1 GB (i.e. 2.5%) for peice of mind is not a bad idea.

You also need to keep in mind that swapping takes up system resources. The time to access the hard disk is hundreds of times slower than the time to access RAM. Therefore, if speed is an important consideration, you should think about having enough RAM so you don't swap. The maximum size of your swap space depends on your hardware architecture and more recent kernels on the i386 can have swap partitions that are as large as 2Gb and you can have as many as 8 different swap partitions if you have kernel older than 2.4.10. Later versions support up to 32 swap spaces.

Note that I said swap *spaces* and not just swap device or swap partition. Linux allows you to create a swap *file*. Like any other file, a swap file exists on your filesystem and takes up space. The advantage is that you can add a swap file at any time, provide you have the space on the hard disk. You don't need to to repartition your hard disk or even reboot.

There are two different swap versions (or formats). Kernels prior 2.4 supports only version 0 swap spaces. Versions later than Linux 2.1.117 support version 0 and version swap. However, Linux 2.5 only supports version 1. Therefore you need to be careful when upgrading. The mkswap command can format in either format. See the **mkswap** for more details.

Another change with the 2.4.10 kernel is that the swap spaces can be up to 64 Gb in size. Note, however, that with some Linux distributions, the mkswap command can currently (Sep 2003) only create swap devices that are 2GB or smaller.

Managing Swap

In many cases, once the system is installed, you never have to think about swap again. However, when you start using your system more actively, add new software, and so on, you will probably find that you should at least take a look at your current swap usage.

Linux provides a number of tools to monitor swap. The easiest is the **free** command, which gives you a quick overview of the memory usage. You can also use the **top** which can provide an self-updating view of your system, including memory usage by process, users on the system, and so forth. Also the **/proc/swaps** and **/proc/meminfo** files contain information about memory usage on your system.

Linux also provides tools to manage your swap space. You can add and remove spaces as you need to, as well as turn them on and off, even while they are being used.

To create a file to be used a swap, you need to first create the file. This is most easily done with the **dd** command. To create a 65 Mb file, you might have this command (from the the mkswap man-page):

```
dd if=/dev/zero of=/data/swapfile.1 bs=1024 count=65536
```

which displays:

```
65536+0 records in
65536+0 records out
>
```

Next you have to prepare the file for usage as swap space using the **mkswap**. The simplest form would be:

mkswap device size

Where "device" is either the name of a device node for a hard disk partition or the name of a file you want to use. The "size" option is only required when you create a swap file. However, it is actually superfluous and still maintained for backwards compatibility. The command you issue migt look like this:

mkswap /data/swapfile.1 Which displays:

```
Setting up swapspace version 1, size = 67104 kB
>
```

What that this does is format the swap space by adding the swap header information. Note this said that it created a version 1 swap space. We could have used the -v0 option to create a version 0 swap space if we needed to.

One key thing to keep in mind is that the kernel needs to have the swap file created to it's full size before it is used. That is, it cannot be a sparse file (one that only uses part of the allocated space).

At this point we are ready to activate the swap space. If you are adding the swap space permanently, then you will need to include it in your **/etc/fstab** file. My default (initial) swap space looks like this:

```
/dev/hda5    swap    swap    pri=42    0 0
```

This basically says that the device /dev/hda5 is to be mounted onto the special mount point swap, is of type swap, has a priority of 42 and that the filesystem should not be dumped if the system crashes, nor should the filesystem be checked on system boot.

To automatically use the swap file we just created, we might add an entry that looks like this:

/data/swapfile.1 none swap pri=5,defaults 0 0

When your system boots, all swap devices will be added which are listed in the **/etc/fstab**, unless they have the "noauto" option (just like any normal filesystem). If I wanted to immediate add the swap space without having to reboot, I can run swapon -a, which will activate all swap spaces in the /etc/fstab file(again, unless they have the "noauto" option). If the swap space is already in use, the system will silently ignore it.

As you might guess, the priority of the swap space determines the order in which the swap space is used. The higher the priority the sooner it will be used. In this example, the primary swap space in its own parition has a priority of 42 and will be used before the swap file with a priority of 5.

We can also add swap space dynamically using the **swapon** command. After creating the swap space, you might activate it with this command:

swapon /data_c2/swapfile.1

To show what is currently beeing used as swap space we issue the command **swapon -s**

This might show us:

```
Filename                        Type            Size    Used    Priority
/dev/hda5                       partition       409616  200560  42
/data_c2/swapfile.1             file            65528   0       -1
>
```

Just as you can enable swap from the command line, you can also turn it off. This is done with the swapoff command and it might look like ths:

swapoff /data_c2/swapfile.1

For more details see the swapoff man-page.

If performance is really an issue thare are number of different things you can do. If you have multiple hard disk on different controllers, you can put swap spaces on each device and give them the same priority. This works like tradition disk "striping", whereby the kernel uses both swap spaces in parallel.

Another way of increasing performance is to seperate the swap service from data device. If you have multiple hard disk controller (or maybe a single SCSI host adapter with multiple channels, you can put the data on one channel and the hard disk with the swap service on a different channel. Putting the data on the SCSI devices and the swap device on an IDE controller may also be possible. In essence you are trying to keep swap on the less used devices, so that the heavily used devices don't get slowed down by swap.

With the Linux 2.5 and later kernel, swap files are basically as efficient as swap devices. This is usefil as swap files are easier to administer than swap devices. You can add, remove and resize then as you need to. If you have two hard disks, then you can create swap files on each devices (with the same priority) and improve swap performance. However, always keep in mind that swap space is just a safety. If you really need to swap often then the best thing is to buy more RAM.

13.1.6 Installation Problems

Some possible problems could occur during the installation, one of which is that you have no free space. Many Linux distributions will allow you to install it on an existing DOS partition, so you need to be careful with this one. Fortunately, the system will tell what kind of file system it is.

If you are having trouble booting from floppies, there are a few reasons for this. The first, which I saw quite often while I was in tech support, happens on new machines when the CMOS setting for the floppy does not match what really is in the drive. Even today, CMOS are delivered in which the default A: floppy is 5.25" and 360K. If you have a 3.5" 1.44MB, it won't work right. The kernel *might* load, because it is just reading one sector after the other. However, the first time you have to seek to a particular sector, you're hosed.

Another common problem is simply that it is defective media. If you can install directly from the CD-ROM, this problem is unlikely to happen. If your CD-ROM drive needs a caddy, I have seen sticky slots that don't open all the way. It might also be that the CD-ROM is just dirty. Wipe it off with a clean *dry,* non-static cloth. If you have to create a boot floppy set, it's best to have new floppies. The cost of a new box is worth the piece of mind.

If the system hangs during boot process, pay close attention to any messages that come up, particularly the last few. These messages might indicate what the problems are. Although it didn't cause a hang, I had an Ethernet card and multi-port board that were both software configurable. Both configured to IRQ 10. Because the multi-port board was initialized first, my Ethernet card was inaccessible. Also check the Hardware HOWTO to make *sure* that your hardware is supported.

I have heard of out-of-memory errors. I have never experienced them myself because I have always installed my system with 32MB. The reason for this error is that you have very little RAM and a RAM disk is created during the install for the root file system. If the amount of RAM you have minus the ramdisk doesn't leave enough for Linux, you're in trouble. If you don't have enough RAM, get more. RAM is not prohibitively expensive, and you cant run a productive system without it anyway.

If you run into hardware problems, strip the system to just the devices you need for the install. Remove the parallel/serial ports, sound cards, network cards, mouse cards, and anything else you don't need for the installation. If the problem still continues, maybe one of the other cards is loose (like the SCSI host adapter). Remove and reseat all the cards. Check for conflicts of base address, IRQ, DMA, and, obviously, check that your hardware is supported.

SCSI devices are good things to have, but they can also be a major headache. I must admit that at times I almost have a too-relaxed attitude when installing. (Hey, I know what I am doing, right?) I often take for granted that the hardware is configured in a particular way, especially when other hardware of the same type is configured that way. This leads to problems!

I have seen systems in which a SCSI device is detected at all IDs, though no devices are there and no manufacturers ID is recognized. This happens when there is a device and HA at the same SCSI ID. The host adapter must be at ID 7. Every other device must be at a different, *unique* ID.

Maybe Linux detects errors on the disk, but the disk is known to be error free, which is often caused by either bad cables or incorrect termination. If your SCSI devices report timeouts, there may be a conflict with base address, DMA, or IRQ of the host adapter. I have also experienced timeouts when more than two devices on the bus are terminated. Remember that only the two devices at the physical end of the SCSI bus should be (must be) terminated. the SCSI devices report timeouts, this may also be due to bad connections or termination problems. However, it might also be due to conflicts with the I/O address or IRQ or DMA channels.

If you have obtained a version from the Internet, you can get read errors or "file not found" messages. I specifically mentioned the Internet because I have never seen it with commercial

versions. This might be an indication of bad media (so instead, use new floppies) or that something went wrong with the transfer.

If you used ftp to copy the files from DOS/Windows machine, the transfer mode is normally ASCII by default. These files *must* be transferred in binary mode or they could result in messages like "tar: read error" or "gzip: not in gzip format."

After installing, you may have some problems booting. You may see something like "non-system disk in drive," "disk not bootable," "NO OS," etc., probably because your master boot block is corrupt. This can occur when installing onto machines with multiple operating systems. Some OSs have to be installed on the active partition. I have experienced this *once*, when I installed a second OS on a system only to find that the second simply overwrote the first.

The fdisk program on all PC-based OSs can read the partition table. It may not see what kind of partition it is, but it will see that something is there. On the other hand, the Linux fdisk can read the partition table and recognize what kind of partitions they are.

Another cause of this kind of message is that LILO might not have been installed correctly. Therefore, you should boot from the boot/root floppy that you made for the installation. Once you get to the boot:, you can specify the root file system to be on the hard disk. Once there, check the /etc/lilo.conf file and install it again.

One of my favorite problems I saw both with Linux and other UNIX dialects. When you boot, another OS starts instead of Linux, or Linux is installed first and after installing the other OS, Linux is no longer accessible. In the first case, LILO may not have been installed. If it was installed, maybe it was not installed in the master boot record; but Linux is not active, so the other OS boots. Maybe you simply configured LILO so that OS will boot by default. Ctrl+Shift+Tab and often just Tab gives you the LILO menu. (See man-page for more details.)

Depending on the distribution, you may be able to choose what type of file system on which you want to install. If you will remember from the section on file systems, Linux supports several different types of file systems, most of which you can install on.

If you are fortunate to have a system that understands the RedHat Package Manager Format (rpm), adding additional software is fairly easy. See the rpm man-page for more details.

13.1.7 Preparing for the Worst

One thing that many administrators don’t plan for is the possibility that things will go wrong. Although good planning and preparation can prevent many disasters, the most devastating disasters are those that are you are not prepared for. When you’re doing an upgrade, a point will come when backing out is no longer possible. If the new operating system doesn’t boot and you cannot go back to the old one, then you have problems.

One problem with which I have personal experience is device drivers for "newer" hardware. In most cases, the driver will list what operating systems under which it is supported and what specific releases. Unfortunately, in a few cases, the vendor says what versions of Linux under

which it is supported because the vendor doesn't provide any support itself.

Another problem is that the driver is release-specific. If you have a driver that was written for the 2.0 kernel, there are no guarantees that it will work with the 2.2 or 2.4 kernel.

13.2 Doing the Installation

Because the installation routine of each Linux distribution is one of the most unique aspects of that distribution it is very hard to cover the details of each. I will be primarily addressing topics and issues that are common to every distribution. However, since I primarily use SuSE, I will be address issues that are specific to SuSE. If you use another distribution, I can use any help you want to provide.

Please note that this section is very much in the development stage. Rather than waiting until I have it all complete, I will be providing articles as I write them (as is in other sections).

13.2.1 Installing the File System

When doing a fresh install there is generall very little that you need to do in regard to installing a filesystem. Most of the work is done for you. All you really need to do is choose the filesystem you want to install. However, for many people (newbies and experienced users alike), which file system to choose it not always clear-cut.

In the section on supported filesystems, we talk about some of the various filesystem types and what things to consider when choosing one over the other.

If you expect that you are going to have a large number of files on your systems (like hundreds of thousands or millions), then you will need to increase the number of inodes. Basically, inodes are pointers to the data on the disk, as well as contain information about the file such as the file owner, permissions and so on. In most cases, you are limited by the number of inodes which are created when the filesystem is created. Once you run out, you need to either move the files to another filesystem, or backup the data, recreate the filesystem then restore the tape. Both are sometimes not possible. Keep in mind, however, that for most home users, the defult number of inodes is usually enough.In the section on the disk layout we talk about inodes in detail.

Generically, filesystems are created using using the **`mkfs`** command. One of the options it takes is the filesystem type. The basically tells mkfs to start the real program which then creates the filesystem. For example, the mke2fs is called when create an ext2fs. When calling mkfs, you typically have few options. So if you need to specify the number of inodes, you will need to start the real program by hand. In the case of the ext2fs, you can use the -i option to specify the number of inodes. Check out the appropriate man-page for details.

One interesting thing about the ReiserFS is that inode are allocated dynamically. That means you will only run out of indode when you run out of space on the disk.

Examples of mkfs commands are:

Filesystem	Command
EXT2 FS	mkfs.ext2
EXT4 FS	mkfs.ext3
SCO BFS	mkfs.bfs
Minix FS	mkfs.minix
DOS (FAT) FS	mkfs.msdos
Virtual FAT FS	mkfs.vfat
XFS	mkfs.xfs
XIA FS	mkfs.xiafs

A list of supported filesystem types can be found here.

13.3 Upgrading an Existing System

The fact that your computer contains an existing Linux installation presents its own set of problems. When I first started using Linux, none of the distributions did an upgrade in the sense that other OSs did them. In every installation (at that point), the root file system was **overwritten** (during which all data is lost). If your data is on a different file system, it is "probably" safe. However, all the configurations that you have done in the files on your root file system are gone.

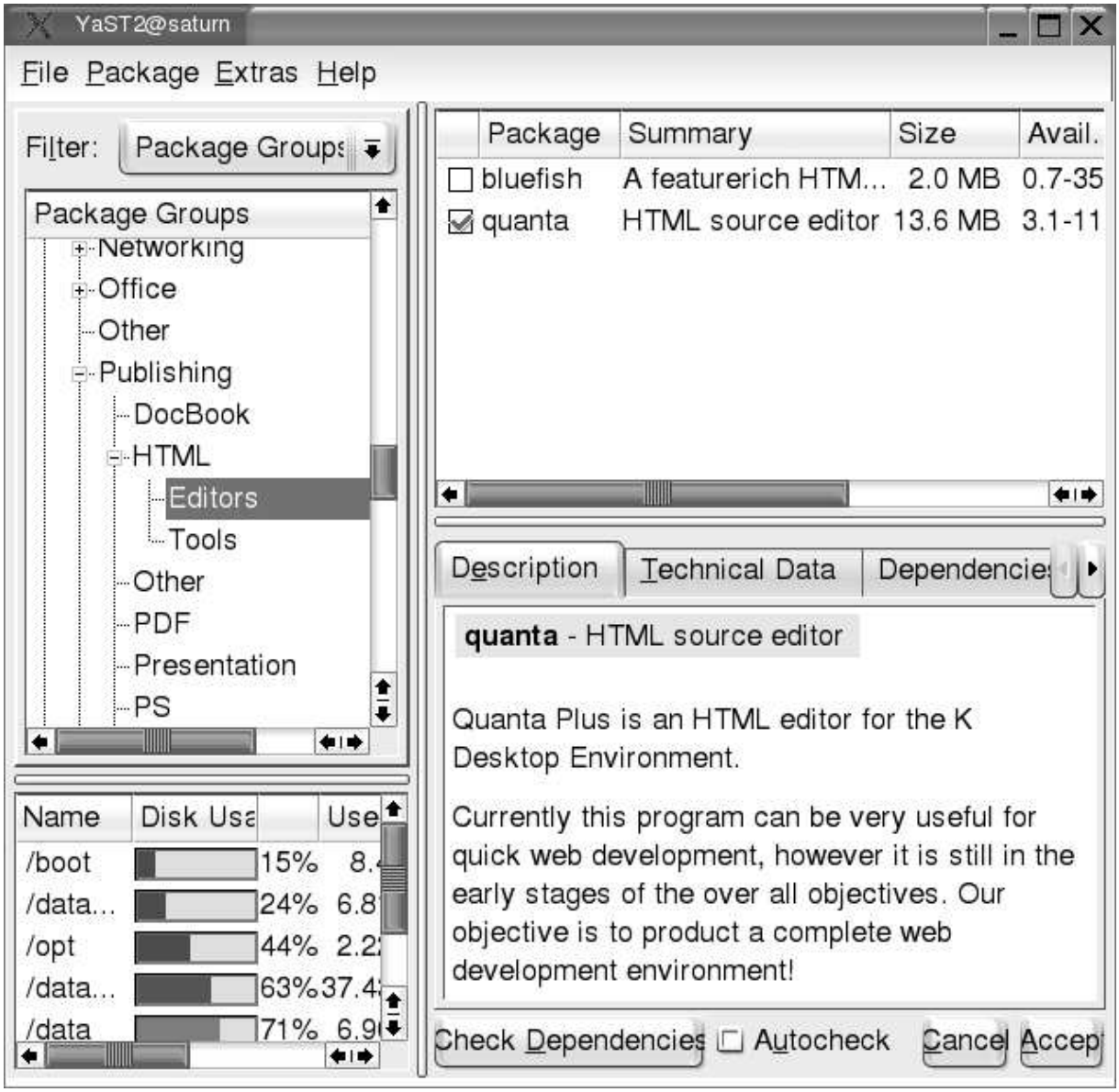
However, that was more of a design than a problem. If you wanted to upgrade the kernel, that's all you had to do. Most programs did not (and still do not) depend on specific releases of the kernel, so upgrading presented few problems for them. Keep in mind that the newer kernel versions support ELF binaries. If you have an older program that is in a.out format, it will still run on the newer kernels. However, newer ELF binaries won't run on older kernels.

Also, upgrades to the kernel occur at fairly frequent intervals. These upgrades are not full copies of the newest kernel but rather patches to an existing one. Even if you do replace the entire kernel, you rarely have to do more than replace the /usr/src directory.

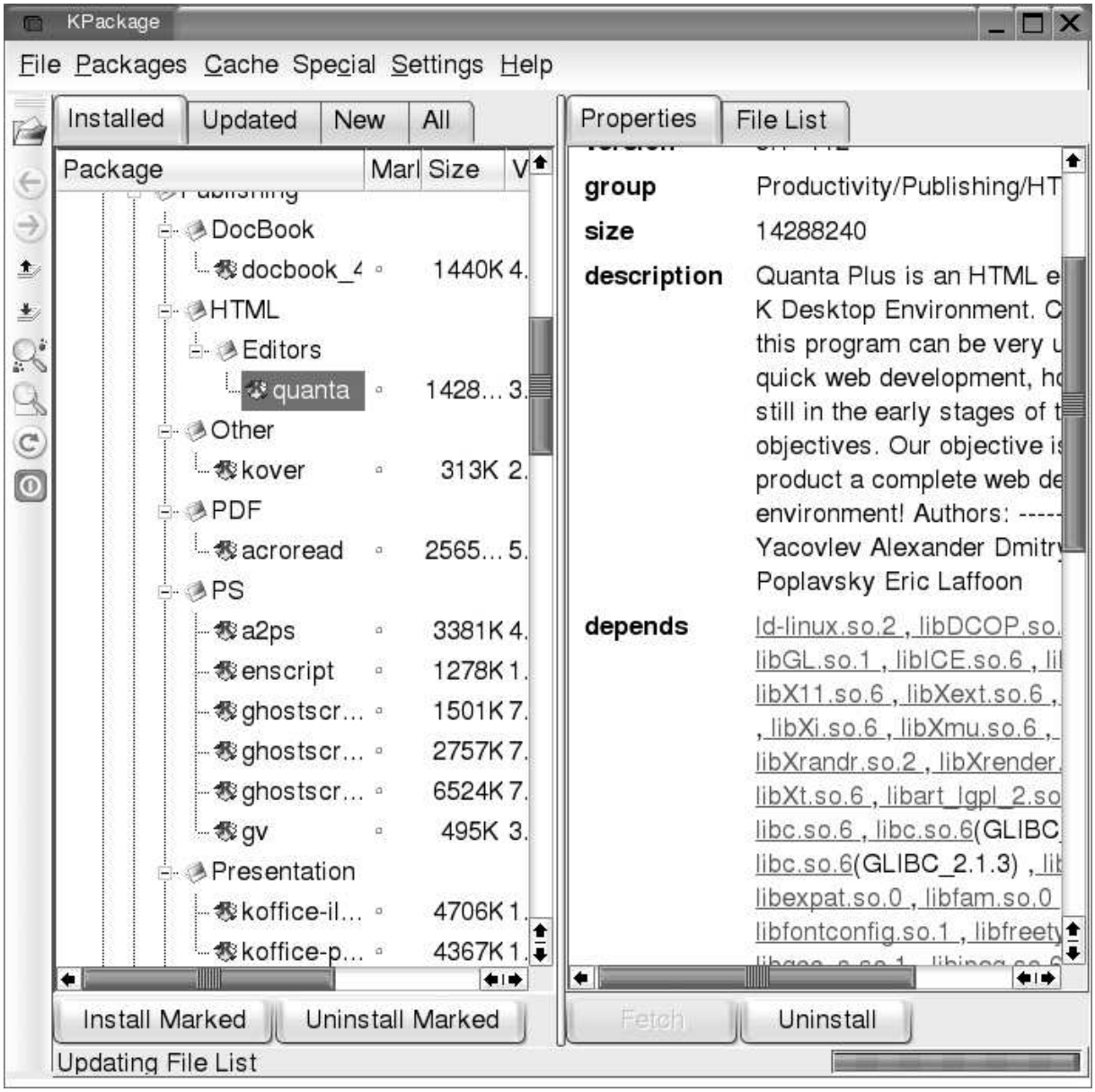
When programs or sets of programs (packages) are updated, you only need to upgrade what you want. You can find the latest versions of the packages in several places on the Internet. The most common format is RPM, so you can use the rpm program to install and upgrade your programs. Many distributions have glint, which is a graphical program with which you can remove and install each package.

One advantage of glint is that you can install sets of packages. For example, you have a set of editors, such as **`emacs`** and **`vi`**. Rather than first installing emacs and then vi, you can install them both at once.

SuSE uses YAST2, which is a very extensive administration tool and which you to not only install the software, but it can check and resolve missing dependencies (packages which are needed to run the one you are trying to install.) Here you can see what it tooks like:



If you don't like YAST2, you can try Kpackage, which is specifically for the KDE environment. Here you can see what it looks like:



In both cases (YAST2 and Kpackage) you can display the packages group based on their functionality, like editors or networking. In each case, you can see what dependencies this application has, the version and so on.

If you don't have any of these tools or simply don't want to use a graphically tool, running **`rpm`** from the command line is only slightly more complicated. Although there are a large number of options and arguments, it is fairly straightforward to install something.

If you have an older hard disk (read: smaller), you might run into problems with some of the new distributions. My SuSE 8.2 contains 5 CDs (or two DVDs) and if you decided to install *everything* (one of the options), you may find yourself running out of space. I created a 3 Gb partition for **`/opt`** on my 20 Gb drive (when 20 Gb was a *large* drive). After about a year of updates and installation of invidual software I started running out of space on /opt.

If you are upgrading your system and find out you are running out of space (or just through normal usage), there is hope. Obviously you will need some place to put it, so we will assume that you have an additional drive. Create partitions on the new drive and mount them to a directory that is not being used. A common place is /mnt, but I often create a new direcotry that is somehow related to what I am trying to move. For example, I might create a /opt2 directory and mount the files system there.

Once mount you can **copy** (not move) the files to the new filesystem. The **cp** should have an option -R or -recursive with will copy everything recursively (what else?). In the unlikely event your version of cp does not have a recursive option you can use **cpio**. Change into the old directory and run:

```
find . | cpio –pvdum /new_directory
```

Once the files are copied, you need to change the **/etc/fstab** to reflect the new filesystem. For example, assuming you want to put the **/opt** directory on a new disk (let's say the first partition of the second hard disk), your in **/etc/fstab** might look like this:

```
/dev/hdb1   /opt     xfs   defaults    1 2
```

Next, rename the old directory (i.e. /opt.old) and then rename the new directory to the original name (i.e. mv /opt2 /opt). Now reboot.

. Note that I did **not** say to remove the old directory. You want to make sure this worked before destroying anything. When you reboot, the new filesystem will be mounted onto the new /opt directory (in this example). If you run into problems, you can simply change the directory names back to the way they were and then reboot.

You *could* also simply reboot with out renamed the original /opt directory. This will mount the new filesystem over the top of the original directory. The files will still be intact, but they won't be accessible.

13.4 Adding Hardware

No one really expects your system to remain stagnant. As your company grows, your computer should be able to grow with it. You can add more memory or a new printer with little or no changes to the operating system itself. However, if you need to add something like a hard disk or CD-ROM drive, you may need to tell the operating system about these changes, especially if the piece you are adding is the first of its kind.

You tell Linux that you have new hardware by linking in the appropriate drivers in the kernel. This is done when you run the make conf when you are asked a series of questions about your hardware. In newer distributions, you will find in your kernel source directory a subdirectory, Documentation, that contains some very detailed information about what to look out for.

As when you add any device to your system, I need to remind you to have everything planned out before you start. Know what hardware exists on your machine and what settings each card has. Look at the documentation to see what settings are possible in case you need to change them. If you are adding a new device of a particular type, the odds are that you are going to

need to make some changes.

13.4.1 Preparation

As I have said before and will say a hundred times again, before you start to do anything, prepare yourself. Get everything together that you will need before you start. Gather the manuals and diagrams, get your notebook out, and have the settings of all the other cards in front of you. Also, before you start, read the manual(s) that came with your release, the installation HOWTO, and the hardware HOWTO.

Perhaps the most important thing to do before you add anything is to know how the hardware is *really* configured. I need to emphasize the word "really." Many customers have become arrogant with me, almost to the point of being angry because they insist that the hardware is set at the "defaults" and I insist that they check anyway.

Unfortunately for me, in most cases, the devices are set at the default. However, confirming this takes less time then trying to install something for a couple of hours and only then finding out the devices are not at the default. Therefore, I would insist that the customer check before we continued. I had one customer who made noises as though he was checking and "confirmed" that the device was at the default. After almost an hour of trying to get it to work, I asked him to change the settings to something other then the defaults, thinking maybe there was a conflict with some other device. Well, as you might have guessed, he didn't need to do anything to change the devices to non-default because they already were non-default! Lesson learned: *Don't lie to tech support* (or to people on the Internet).

One most important thing about installing hardware is knowing what you are doing. This may seem like an overly obvious thing to say, but there is a lot more to installing hardware than people think.

One most common problem I see is that cards are simply snapped into the bus slot with the expectation that whatever settings are made at the factory must be correct. This is true in many, if not most, cases. Unless you have multiple boards of the same type (not necessarily the same manufacturer), you can usually get away with leaving the board at the default. But what *is* the default?

Unless you are buying your hardware secondhand, some kind of documentation or manual ought to have come with it. For some hardware, like hard disks, this documentation may only be a single sheet. Others, like my host adapter, come with booklets of 50 pages or more. These manuals not only give you the default settings but tell you how to check and change the settings.

On ISA cards, settings are changed by either switches or jumpers. Switches, also called DIP switches, come in several varieties. Piano switches look just like piano keys and can be pressed down or popped up. Slide switches can be moved back and forth to adjust the settings. In most cases, the switches are labeled in one of three ways: on-off, 0-1, and closed-open. Do not assume that on, open, or 1 means that a particular functionality is active. Sometimes "on" means to turn on the *disabling* function. For example, to disable the floppy controller on my SCSI host adapter, I put the jumpers in the "on" position. Always check the documentation to

be sure.

Jumpers are clips that slide over pairs of metal posts to make an electrical connection. In most cases, the posts are fairly thin (about the size of a sewing needle) and usually come in rows of pin pairs. Because you need both posts to make a connection, it is okay to store unused jumpers by connecting them to just one of the posts. You change the setting by moving the jumper from one pair of pins to another.

The most common thing that they jumpers configure are base address, IRQ, and DMA. However, they sometimes are used to enable or disable certain functionality. For example, jumpers are often used to determine whether an IDE hard disk is the master or the slave.

The other three bus types, MCA, EISA, and PCI bus cards, often do not have DIP switches or jumpers and are software controlled. They normally come with some kind of configuration information provided on a disk or in CMOS. Provided with the machine itself is a configuration program that reads the information from the diskettes and helps to ensure that there are no conflicts. Although hardware settings are configured through these programs, it is still possible in some cases to create conflicts yourself, though the configuration utility will (should) warn you.

If this configuration information is on a disk, then this disk is as important as the Linux installation media. If this disk gets trashed, there is no way you will even be able to add another piece of hardware, your kids will hate you, and your co-workers will think you are a geek. On the other hard, you could get a copy from the manufacturer, but that's a hassle. The real problem is that some machines, like the IBM PS/2 series, recognize when you add a new card to the system and won't let you boot until you feed it the configuration disk.

In some cases, the configuration program is part of the hardware (firmware) or resides in a normally inaccessible part of the hard disk. Check the machine documentation for details.

Knowing what kind of card you have is important not only to configure the card, but unless you know what kind of bus type you have, you may not be able to even put the card in the machine. This is because all the different cards are different sizes or shapes. As you would guess, the slots on the motherboard they go into also are different sizes and shapes.

Hopefully, you knew what kind of card to buy before you bought it. Often, however, the administrator will know what kind of slots are on the machine from reading the documentation but never opens the case up so has no idea what the slots look like. If this is a pure ISA or MCA machine, then this is not a problem because none of the cards for any other bus will fit into these slots.

Problems arise when you have mixed bus types. For example, it is very common today to have PCI or VLB included with ISA or EISA. I have also seen MCA machines with PCI slots, as well as both PCI *and* VLB in addition to the primary bus (usually ISA or EISA). If you've gotten confused reading this, imagine what will happen when you try to install one of these cards.

If you are using your Linux machine as a server, the computer you bought probably has at least six slots in it. There can be more slots if you have a tower case or PCI in addition to the primary bus. Due to the distinct size and shape of the cards, it is difficult to get them into slots in which they don't belong. Usually there is some mechanism (i.e., notches in the card) that prevents you from sticking them in the wrong slots.

On some motherboards, you will find a PCI slot right next to an ISA or EISA slot. By "right next to," I mean that the separation between the PCI slot and the other is much smaller than that between slots of the same type. This is to prevent you from using both slots. Note that the PCI electronics are on the opposite side of the board from ISA and EISA. Therefore, its impossible to fill one slot and then use the other.

When installing the expansion cards, you need to make sure that the computer is disconnected from all power supplies. The safest thing is to shut down the system and pull the plug from the wall socket so that you ensure that no power is getting to the bus, even if it is turned off. Though the likelihood of you injuring yourself seriously is low, you have a good chance of frying some component of your motherboard or expansion card.

Another suggestion is to ground yourself before touching any of the electronic components. You can do this by either touching a grounded metal object (other than the computer itself) or wearing a grounding strap, which usually comes with a small instruction sheet that suggests where the best place to connect is.

When I first started in tech support, we had a machine that would reboot itself if someone who wasn't grounded so much as touched it. I have also cleared my CMOS a couple of times. Therefore, I can speak from experience when I say how important grounding yourself is.

When you open the case, you will see parallel rows of bus card slots. Near the outside/back end of the slot, there is a backing plate that is probably attached to the computer frame by a single screw. Because some connector, etc., sticks out of the slot in most cases, this plate will probably have to be removed. In addition, the card will have an attachment similar to the backing plate that is used to hold the card in place. I have seen cards that do not have any external connector, so you could insert them without first removing the backing plate. However, they are not secure because nothing holds them in place. See the hardware chapter for more details.

This plate has a lip at the top with the hole for the screw. To align the screw hole properly, insert the card correctly (provided it is in the right kind of slot). As you insert the card into the slot, make sure that is going in perpendicular to motherboard. That is, make sure that both ends are going into the slot evenly and that the card is not tilted. Be careful not to push too hard, but also keep in mind that the card must "snap" into place. Once the card is seated properly in the slot, you can make the necessary cable connection to the card.

After you connect the cables, your can reboot the machine. Many people recommend first closing the computers case before turning on the power switch. Experience has taught me to first reboot the machine and test the card before putting the case back on. If, on the other hand, you *know* you have done everything correctly (just as you "know" the hardware is at the default, right?), go ahead and close things up.

Avoiding address, IRQ, and DMA conflicts is often difficult. If your system consists solely of non-ISA cards, it is easy to use the configuration utilities to do the work for you. However, the moment you add an ISA card, you must look at the card specifically to see how it is configured and to avoid conflicts.

If you have ISA and PCI cards, you can use the PCI setup program (which is usually built into the firmware) to tell you which interrupts the ISA bus cards are using. You can also reserve specific IRQs for ISA devices. If you don't, there may be interrupt conflicts between the ISA bus card and a PCI bus card, or even between two ISA cards.

Unlike DOS, every expansion card that you add to a Linux system requires a driver. Many, such as those for IDE hard disks, are already configured in your system. Even if the driver for a new piece of hardware exists on your system, it may not be linked into your kernel. Therefore, the piece of hardware it controls is inaccessible.

13.4.2 CPU

There isn't too much I can say about adding CPUs. There are no jumpers to set on the CPU. In some newer machines, a lever pops out of the old CPU and you pop in a new CPU. This is (as far as I have seen) possible only on 486 machines to enable you to add a Pentium. These levers are called Zero-Insertion-Force (ZIF) sockets because you use the lever to lock the CPU into place and it requires zero force to insert it.

One thing that you must consider is the speed of the CPU. You may want to increase the speed of the CPU by simply buying a faster one. From Linux's perspective, this is okay: you plug it in, and Linux can work with it. However, it might not be okay from the hardwares perspective. Motherboards are often sold with the same speed as the CPU because they cannot handle faster speeds. Therefore, in many cases, you cannot simply replace the CPU with a faster one.

In other cases, the motherboard is of higher quality and can handle even the fastest Pentium. However, if you only had a 50MHz 486, then you might have to change jumpers to accommodate the slower CPU. Often these changes effect such things as the memory waits states. Here you need to check the motherboard documentation.

13.4.3 RAM

The day will probably come when you need to expand the RAM on your system. As more users are added and they do more work, the little RAM you have won't be enough. Once you have decided that you need more RAM, you still have most of your work ahead of you.

One key issue you need to consider is the kind of RAM. In almost all newer machines, RAM comes in the form of SIMM modules. As I mention in Chapter 12, these modules are about the size of a stick of chewing gun, with the chips mounted directly on the cards. These modules have almost entirely replaced the old RAM, which was composed on individual chips.

There are two primary type of SIMMs. The somewhat larger type is called a PS/2 SIMM because it was first used on PS/2 machines. This has 72 connectors on it and can be immediately distinguished by the small notch in the middle. The other kind is referred to as non-PS/2, normal, regular, etc. This has 30 pins and no notch. (There is also a 32-pin SIMM, but it is uncommon. I have never seen one, but someone who works in the chip manufacturing business told me about it.)

Two important aspects of RAM are the speed and whether it has parity. The speed of RAM is measured in nanoseconds (ns). Most RAM today is either 70 or 60ns, with a few machines still being sold with 80ns. The speed of RAM is a measure of how quickly you can read a particular memory location.

Although it is possible in many cases to mix RAM speeds, I advise against this. First, memory can only be accessed as quickly as the slowest chip. Therefore, you win nothing by adding faster RAM, and loose if you add slower RAM. I have also seen machines in which mixing speeds actually causes problems. Because the difference between 80ns and 70ns is more than 10 percent, the delay waiting for the slower (80ns) RAM makes the system think that there is a problem. This can result in kernel panics.

Another issue is the motherboard design. For example, in one of my machines, I have two banks of memory with four slots each. Because of the way the memory access logic is designed, I must fill a bank completely, otherwise, nothing in that bank with be recognized. On other machines, you can add single SIMMs. Check the documentation that came with the motherboard.

Another important issue is the fact that Linux uses only extended, not expanded, memory. Expanded memory dates back to the early days when the XT bus could only handle up to 1MB of RAM. To give programs access to more memory than the computer could handle, some memory board manufacturers came up with the concept of "bank switching." With bank switching, a 64K area between 640K and 1Mb is reserved and then portions above 1Mb are "switched" into this reserved block as needed.

When the AT bus was developed, the system could access more memory. To make it compatible with older peripherals, however, the area between 640K to 1MB was left "unused." Memory beyond 1MB is known as extended memory.

Some machines have a hardware limitation on the maximum amount of memory that can be installed. Because I use 30-pin SIMMs on an older machine, the largest available (as of this writing) is 4Mb. Because of this, I max out at 32Mb when I fill all eight of my slots. If you have 72-pin memory, there are larger modules, such as 64, 128, 256 and even 512Mb. Again, refer to your motherboard manual for details.

If you have a Pentium or Pentium Pro, you will need to add PS/2 memory in pairs because memory is read 64 bits at a time and PS/2 SIMMs provide only 32 data bits. If you have a Pentium machine that has the older SIMMs, they will have to be in groups of four.

If you experience repeated panics with parity errors, consider replacing your memory. Because of the way memory is accessed, you may have to remove or replace entire banks (like mine). I have also seen cases in which mixing memory types and speeds can cause

panics. Panics may also be the result of improperly inserted SIMMs. That is, if the SIMM is loose, it may not make a good contact with the slot. If the machine gets jarred, the contact may be lost.

In some cases, you can simply add the memory and the machine will recognize it (like mine). However, I have some machines for which you have to set jumpers to enable each bank as you add memory. In other cases, when you have filled up all the slots on the motherboard, you can add a memory expansion card. If this is on a machine like a PS/2, it requires you to tell the system of the memory expansion card using the configuration disk.

13.4.4 SCSI Devices

If the SCSI is the first device you are adding to the host adapter, you need to configure that host adapter into the system. If you are adding a second host adapter or anything after that, you will have to disable the BIOS on it. Remember from our discussion on SCSI in the chapter on hardware that the system will look for a bootable hard disk. If the BIOS on the second host adapter is enabled, the system may try to boot from this one. If it is the same model host adapter, it will probably have to change the base address, IRQ, and DMA as well.

Remember that every device on the SCSI bus, including the host adapter itself, is identified by a SCSI ID. SCSI IDs range from 07 on standard SCSI and 015 on a Wide SCSI bus. Regardless of what type it is, the host adapter is almost exclusively set to ID 7. It is this ID that is used to identify devices on the bus, not their location in relation to the host adapter.

Because there is no real need to configure a SCSI host adapter unless something is attached, the only "official" way to add a host adapter to the system is to go through the configure script. If the system recognizes that you do not have a configured host adapter, you will be prompted to add one.

If you are adding a device to an existing host adapter, the driver for that host adapter is already linked into the kernel. At least we hope so. If you are adding new kind of device onto the new host adapter, the driver may not be configured. Watch carefully as the system boots to see whether the device is recognized. If it is not, you will need to run the configure script.

Here is one example in which I have first hand experience about not paying attention to what is supported and what is not. The distribution you have may provide a disk image that has support for a particular host adapter or any driver for that matter. However, this does not mean the kernel has the support.

13.4.5 Hard Disks

If you have trouble figuring out what kind of hard disk you have, either refer to the documentation that came with the drive, call the vendor, or re-read the chapter on hardware.

The key thing to keep in mind is to make sure how the disk is configured. Every drive should come with some kind of documentation, either a detailed, multi-page manual, as is the case with many SCSI drives, or a single sheet, as is common with IDE drives. This documentation usually contains information about the default settings of the drive. Don’t trust those default

settings. Check them out yourself to ensure they are correct before you stick the drive in the machine. It is easier to check beforehand than it is after you've tried to install it and failed.

If it's an IDE drive, then one key issue is whether it is the master or slave. If you are adding a second drive to a system that only has one drive, you are adding the slave. If you already have two drives on the first IDE controller and this is the first one on the second controller, then it is the master.

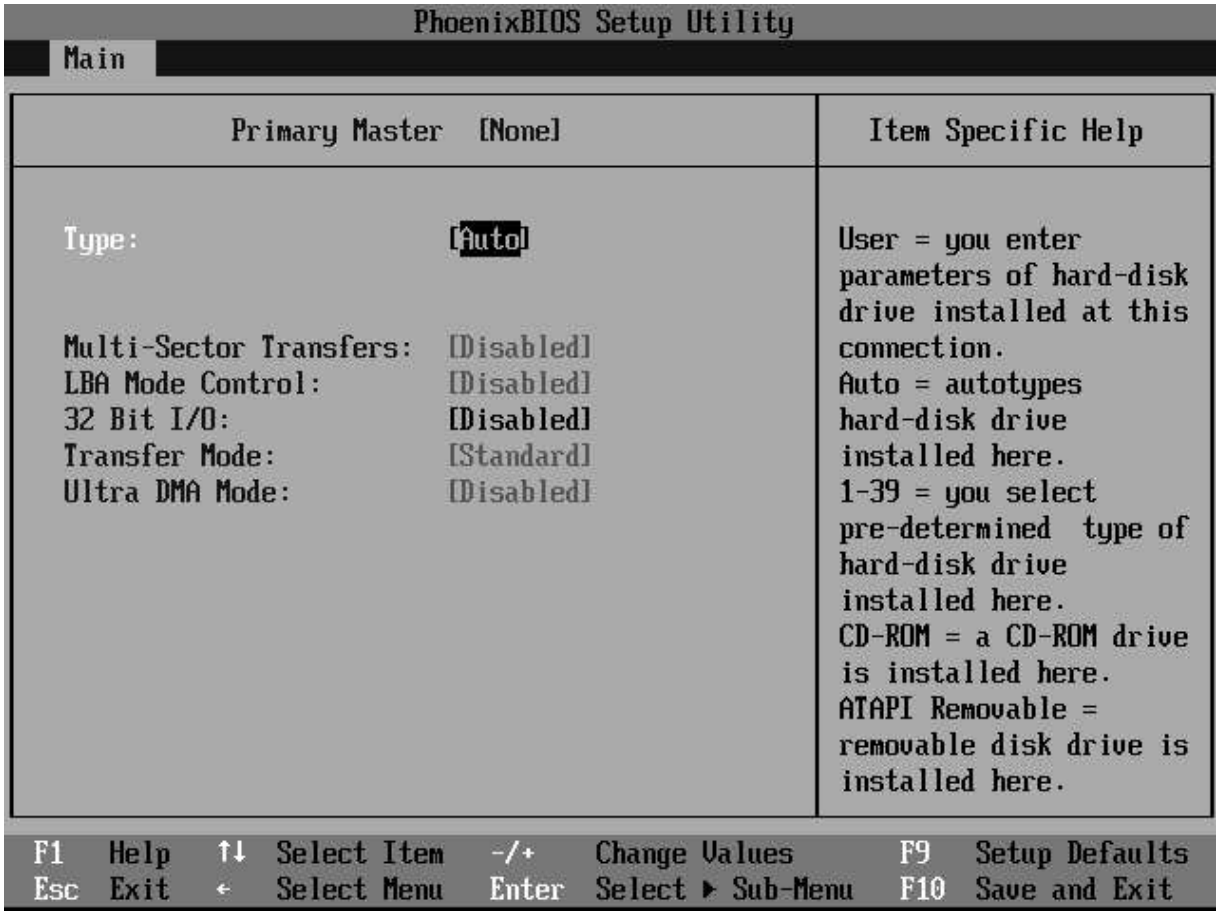
Another key issue is making sure the cabling is right. In the section on hard disks earlier, I mentioned that the position on the cable is irrelevant for IDE drives; the jumpers determine which drive is which.

A problem often crops up when you connect the cable to the drive itself. Usually there is a small "key" on the connector on the cable that fits into a notch on the drive-side connector. If the key is missing or the drive-side connector is wide enough, it may be possible to fit the connectors together backward. Fortunately, you don't have to resort to "trial and error" to figure out which is which. On one side of the cable is a colored stripe (usually red, which is line 1 of the 40-line IDE cable. In fact, on almost all ribbon cables (such as SCSI), line 1 is marked in red.

On the drive side, things are a little more complicated. The IDE cable has 40 parallel lines, but the connectors (both on the cable and on the drive) are in two parallel rows. Usually the connector on the drive is either male (pins) or there is a small "card" sticking out that is similar to an expansion card. These alternate with the odd numbered lines on one side and the even numbered lines on the other.

On the drive near the connector, often on the circuit board itself, will be some small numbers that tell you which pin is which. Sometimes there will be a 1 and 2 on one end with 39 and 40 on the other. Other times there will be just a 1 and 39. (I have seen cases in which there is just a 2 and 40.)

When you boot your system the first time after adding the hard disk, you will probably need to go into the BIOS setup to configure the type of drive. If possible, you should always set the system to automatically detect the hard drive (i.e. autodetection). In some cases, it is simple called "Auto". The figure below shows you what this might look.



If you have an older drive or BIOS, it may be possible that the the drive cannot be recognized. In which case, you will need to define the drive yourself. However, this is unlikely as your system will have to be pretty old not to be recongized. With auto configuration, the CMOS will query drive which will be able to tell it the configuration (i.e size).

SCSI drives may have jumpers to configure them, but they may also be configured in other ways such as with dials, DIP switches, or piano switches. Like IDE drives, SCSI usually has one set of jumpers, which will be for the SCSI ID of the drive. In some cases, there will be up to eight pairs of pins to indicate the SCSI ID. Others have three pins that operate in binary. (For more details on SCSI configuration, see the section on SCSI in the hardware chapter.)

Standard SCSI cable looks very similar to the IDE, except that the cable has 50 lines instead of just 40. However, the same issues with the key and slot and the number applies. On the other hand, I cant remember seeing a SCSI device in which the key and slot didn't match up correctly.

If you are not sure how the drive is configured, check the documentation that came with your drive. If you can't find that, most hard disk manufacturers have fax services and will fax you installation information on any of their drives.

Once you have added the support for the hard disk, you have to create a file system on the disk partition (assuming that you are not creating a swap partition). When you install, you tell the system what kind of file system you want and it will call the appropriate program. For

example, ext2 file systems are created using **mke2fs**.

I want to point out a couple of things. One thing that you can change when running mke2fs is the number of inodes on the files system. Normally, mke2fs will calculate a number based on the size of your partition. If you know that you will have a lot of small files (like for a mail or news server), you may run out of inodes because the average size of the file is less than what **mke2fs** expects.

You can also use **mke2fs** to create a file system on a floppy. You could use this for crash recovery by copying a lot of necessary tools onto the floppy. Keep in mind that you must first format the floppy before you can create a file system on it.

13.4.6 Other SCSI Devices

If you are adding a SCSI CD-ROM, then you must attach it to a supported host adapter to access it. Like a hard disk, if you are adding a CD-ROM and there is not already a SCSI device on your system, you may need to configure the kernel for the host adapter. Also, if you are adding it to a second host adapter, you may need to configure that as well. Before you rebuild the kernel, run make config to specify the option that you want. Here you have the opportunity to select the hardware.

If you are a good little system administrator, you have already bought yourself a tape drive and have configured it into your system. Therefore, I don't need to talk about adding one. On the other hand, a time may come when the tape drive you have is not sufficient for your needs (maybe not large or fast enough) and you have to add a new one.

Before you begin to install the tape drive, be sure that you have read your tape drive hardware manual. This manual will often contain information about the physical installation of the device, as well as configuration. Whenever I think about installing tape drives, I think of the first SCSI tape drive I had. There were three jumpers to set the SCSI ID. It didn't make sense that you could only set it to ID 0, 1, or 2. Had I read the manual first, I would have known that this was the *binary* representation of the number.

13.4.7 EIDE Drives

One limitation of the of the IDE drive is that it was limited to about 520MB because the BIOS can only access cylinders less than 1,024 because it uses a 10-bit value. Because DOS accesses the hard disk through the BIOS, it, too, is limited to locations on the hard disk under this 1,024 cylinder limit.

The older IDE controllers accessed the hard disk through the combination of cylinders, heads, and sectors per track. The problem for the BIOS is when the hard disk has more than 1,023 cylinders. So, to make the BIOS happy, EIDE drives "lie" to the BIOS by saying that there are fewer than 1024 cylinders and more heads. Often, the cylinders are halved and the heads are doubled so that the number of blocks works out to be the same.

If you have a hard disk that is used just for Linux, you can turn off this translation completely before you install and its no longer an issue. However, if you have DOS on the disk, this is a

problem. Linux still gets the geometry from the BIOS as it is booting. If the translation is on, it gets incorrect values for the physical layout on the hard disk.

However, until the kernel is loaded, LILO still uses the BIOS, therefore the Linux image must lie under this limit. The only way to ensure that this happens is to make the root partition lie under the 1,024 cylinder boundary.

Another alternative is to leave the translation on but pass the correct geometry to Linux at the boot/LILO prompt. You can even include them in lilo.conf so that you don't have to type them in every time.

13.4.8 CD-ROMs

CD-ROMs are configured into the kernel like the other devices: through the make config. If you have a SCSI CD-ROM, you obviously need to configure your host adapter as well. You can do this at the same time if your host adapter is not installed already.

For the SCSI CD-ROMs, it's just a matter of turning on the support. Non-SCSI CD-ROMs are different. There are options to turn on the support for several other specific types of CD-ROMs. Rather than listing them all here, I will point you to the CD-ROM HOWTO, which contains more up-to-date information.

13.5 A Treasure Chest of Choices

Getting a copy of Linux is obviously an important part of the install. Technically, Linux is the kernel, so it isn't all that big. You can download the latest kernel from many sites. However, if you want all the programs and utilities that go along with Linux, you will probably spend days downloading everything.

The alternative is to buy a copy. A dozen different versions are available in many more different configurations. Some are sold as a single CD-ROM, others come with CD-ROMs full of shareware or demos, still others are bundled with commercial products.

One argument that I frequently encounter against UNIX in general compared to Windows NT is that so many different versions compete with each other. With Windows NT, there is only one product. For me, that is an argument in favor of UNIX. No matter what your needs, no matter what your tastes, there is a UNIX version to suit you.

You need WWW Server and Web development tools? No problem. Development system? No problem. So, you *really* want to pay for all this? No problem. If you want, you could get it all for free even if Linux wasn't what you wanted.

In the next few sections, we will take a quick look at a few of these distributions. This list is far from complete and represents only those versions I have installed. For more information on these distributions and the others that I didn't install, the Linux Distribution HOWTO contains, among other things, contact and pricing information. There is also information on companies that provide other Linux-related products.

To be honest the best place to look at most all(?) of the distribrutions at once is Disto Watch.

13.5.1 SuSE

Distributor: SuSE GmbH

SuSE comes at the top rather than in alphabetical order simply because it is my favorite distribution. Plus it is the distribution that I used as a reference for most of this site. SuSE, pronounced soo'-suh, comes from the German acronym, "***S***oftware ***u***nd ***S***ystem***E***ntwicklung (Software and System Development).

13.5.2 Deutsche Linux Distribution DLD

DLD is a German distribution of Linux from Delix Computer in Stuttgart, Germany. It comes with two handbooks, both in German, that provide more than enough information to get you up and running. The menus during the installation process, many man-pages, and some of the system messages are in German. For the most part this is good thing, but the documentation and menus appear to have been translated by someone who does not have much experience in UNIX. Many of the terms are simple word-for-word translations of English and are not commonly used German words. In addition, there is no consistency as to what is in German and what is in English.

With the package I received, there were two additional CD-ROMs that contain a larger number of applications, programs, and archives from various sites that I could install later. There were also two floppies that I could boot from so I didn't have to create them myself. A version with the Accelerated X-Windows server is also available.

One major advantage that I found in this distribution was that it not only recognized my existing Linux partition, but when I went to install on that partition, I was prompted to save the system configuration onto a floppy. I was also asked whether the previous configuration should be used to configure the DLD installation. Unfortunately, not all aspects of the configuration were saved.

Also, when I went to remote format this partition, I was reminded that a Linux system was already installed there and I was prompted to confirm the fact that I wanted to reformat the partition. Other distributions would simply go ahead without asking.

During the course of the installation, it automatically recognized that I had a DOS partition and asked me if I wanted to mount it at system startup. DLD went so far as to identify that it was really a Win 95 VFAT file system.

Selecting a kernel to load on the hard disk was rather confusing. Other distributions give you a menu from which you select the components you have. The installation script then picks the appropriate kernel. With DLD, you have a fixed list from which to choose and it is not very clear.

At this point, two annoying problems cropped up. First, when LILO was configured, it installed Linux under the name linux1 and DOS under the name dos2. Afterward, you have to reboot to continue with the installation. Because I expected to be able to type linux to start up,

I was unpleasantly surprised to find that only linux1 worked.

One very exciting aspect of the installation was my ability to install the shadow password facility, which we talked about in the chapter on system administration. Some distributions don't even provide it, but here you can configure it automatically during the installation!

During DLD installation, you have two choices: standard and expert. The expert installation is similar to that of Craftworks in that you can choose from a couple of predefined installations or pick and choose features to suit your needs.

13.5.3 Mandrake

13.5.4 RedHat

13.5.5 Slackware

For a long time, Slackware seemed to be synonymous with commercial distributions of Linux. This was the first version of Linux I had. It, along with most subsequent versions I received from Walnut Creek CD-ROM (*www.cdrom.com*), provides regular updates. Aside from the various versions of Slackware, Walnut Creek also has a plethora of other CD-ROMs. I used several in the course of this projectGNU, Perl, X11R6, and their Internet Info CD-ROM, all of which fit well with my Linux system.

Slackware provides these various Linux programs in several sets. Most of the books that I have seen on Linux describe Slackware as being *the* distribution and these sets being the only way that programs are distributed. One advantage this has is when you want to install from floppies, particularly if you decide to install these sets after the initial installation. Some products I have seen enable you to create installation floppies, but you need to install them all at once. With the Slackware sets, it is easy to create floppy sets (although a couple of sets require a large handful of floppies).

The Slackware 96 package that I got from Walnut Creek CD-ROM is a four-CD-ROM set that contains a couple of different Linux archives. In addition, there is a 36-page installation guide, but this contains just the minimal amount of information for you to install system. If you have nonstandard hardware or trouble installing, you have to look elsewhere.

13.5.6 Turbo