

# Chapter 2

## Fundamentals of Linux

This chapter looks into how the user interfaces with the basic Linux system through the File Operating System, Administration, and Command Line Interface. To understand how the file structure is set up, we need a fundamental understanding of the file names, file path structure and how the user interfaces the Linux Kernel. Finally we need to understand the multiple definitions of the word **root**.

### Concepts Learned in this Chapter

- File Operating System
- Linux Kernel
- User Interfaces
- Pathnames, Files and Directories
- The meaning of Root
- User Home Directory
- Password Security
- Shell Interpreters
- Client-Server Relationship

## Table of Contents

Fundamentals of Linux.....	1
2.1 File Operating System .....	3
2.1.1 Device Drivers .....	3
2.1.2 Standard Input and Output.....	4
2.2 The Linux Kernel.....	4
2.2.1 Hardware.....	4
2.2.2 Kernel.....	5
2.2.3 Shell Interface .....	5
2.2.4 Applications.....	5
2.3 System Administration .....	6
2.4 User Interface .....	7
2.4.1 CLI Mode.....	7
2.4.2 GUI Mode .....	8
2.5 Command Line Interface .....	8
2.5.1 DOS vs. Unix / Linux .....	9
2.6 Graphical User Interface .....	10
2.7 Pathnames and Directories .....	10
2.8 File Storage.....	12
2.9 Filenames .....	12
2.10 Definition of Root .....	13
2.10.1 The root .....	13
2.10.2 Administrator root .....	13
2.10.3 Home Directory /root .....	14
2.10.4 Group root .....	14
2.11 Home Directory .....	14
2.12 Password Security .....	14
2.13.1 Shell Comparison.....	17
2.15 Commands Used in this Chapter.....	17
2.16 Chapter Review Questions.....	18

## **2.1 File Operating System**

The Unix / Linux Operating System (OS) treats information flow as the transfer of information between different files. Everything appears as a file to the OS – this includes data storage devices – disk and hardware interfaces.

The hard drive is a block file device that contain a series of files. The file block device driver interprets the data going to the hard drive and takes care of placing it on the physical device – thus the driver is responsible for tracking and maintaining the hard drive operation rather than the OS. The driver is another of the multitude of processes running in the background, making the total operation appear seamless to the user.

Within the hard drive is a series of files. The first file is that which we would call a directory. A directory is a unit that has as its contents other directories and files. The easiest way to distinguish between a file and a directory is the format that specifies the information that it contains. We can view a file as being a free-format of information, whereas a directory is specifically formatted with information about other files, structured to provide the name, size, attributes, dates and location. So what is the difference between a file and a directory – at least as Unix / Linux sees it – one byte. We only need one bit really, but this will allow us to add more features as needed. For example, one bit set to zero would mean it is a directory, whereas if it is set to a one, then it is a file.

The serial port of the computer is also a file. Data may be sent to it and received from it. Also included with the serial port are the parallel, SCSI, USB, keyboard, monitor, mouse, joystick, and network interface. These are all files with respect to the Operating System. Data is sent to and retrieved from the respective file.

### **2.1.1 Device Drivers**

So how does the information get transferred to a hardware device? You have experienced the situation of having to install device drivers when you install a new device in the MS Windows OS. This device driver converts from the file interface to the hardware, thus the OS can have a standard interface reference. Look at this concept as what you have might have previously learned about the OSI model.

During the installation, or when Linux detects a new device, it records it into the appropriate file. In the case of storage devices such as hard drives, CDROM, Zip drives, and Floppy Drives, the entries are stored in the **/etc/mtab** and **/etc/fstab** files. We will investigate this file in advanced administration topics.

As it is today, the vendors do not support device drivers for Unix / Linux like they do for Microsoft Windows. MS established standards for interfacing to the OS, and left it to the vendors to be compatible. For Linux, the user must create the drives for the vendor hardware. Hence we have developed the open source community to share this information. Hopefully in the near future we will see improved standardization and they will provide drivers on the same floppy / CD along with Windows, and a procedure will be available to install them. Now we only need to have all the vendors support the standard.

When you installed Linux, the device drivers were installed in the /dev directory. If the device is unknown to Linux, it will not have a driver and will not be able to operate.

So back to the device driver, it knows where to look for data in RAM, how to notify the processor when there is incoming information (IRQ, I/O address), and how to translate between the two to make the physical interface operate.

### 2.1.2 Standard Input and Output

Before we proceed with further commands, we need to understand how data is transferred between a file and an application. Previously, we noted that Linux treats everything as a file. This includes all hardware, including the keyboard and monitor.

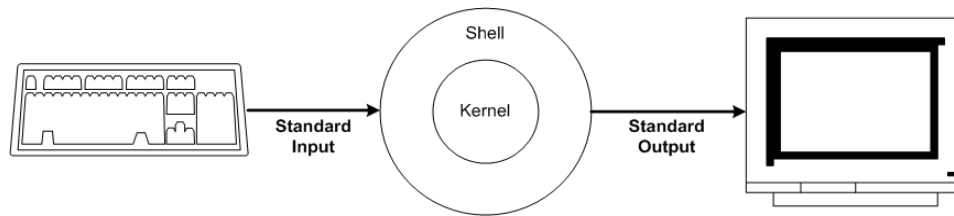


Figure 2.1: Standard Input and Output

When you type commands on the keyboard, this is considered a **standard input**. After the command is processed, the output is displayed on the monitor, the **standard output**.

We will observe later that we can direct the input to be from a different file other than the keyboard, and direct the output to a file device other than the monitor. To achieve this, we will use the **redirector**.

Another question arises, what if there is an error when the command is executed. In this case, the output is directed to the **standard error**. Typically, we observe that the standard error is directed to the monitor, where it is displayed. But this opens another excellent feature. What if an application incurs an error, but we direct the output to a logging file. As an example, say someone attempts to log onto your system for a file transfer, and they enter an invalid username / password; the system can then generate a error message and append it to the appropriate logging file. We now have a track record of who is attempting to gain access to our system.

## 2.2 The Linux Kernel

The design of the Linux system is centered on the base Kernel, developed by Linus Torvalds circa 1990. To start, we need to view a diagram, and then address what each part does and how the user interfaces the total system.

### 2.2.1 Hardware

Observe at the very center of our diagram (Figure 2.2), we have the hardware of our computer. Note that it is totally isolated from the user – so we are not able to issue commands directly to it, but must issue read and write data via various software interfaces.

### 2.2.2 Kernel

The Kernel is the actual operating system software. It takes commands from users, via a shell, and performs various operations according to those instructions. Everything to the Kernel is a “file”. The file then performs the specific required action. Instructions to the Kernel are not the commands that we type in, but rather machine code. Note that the Kernel Interface isolates the user from the hardware – one can not issue commands directly to the hardware, they must go through the Kernel.

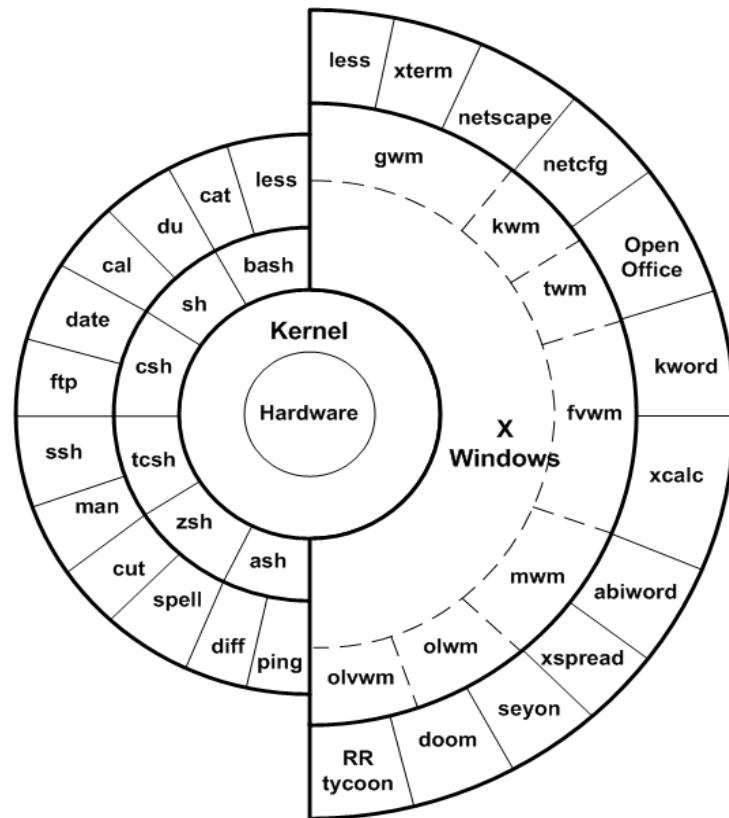


Figure 2.2: Linux Kernel

### 2.2.3 Shell Interface

Interfacing the Kernel is a special program that interprets what the user types in and converts it to appropriate machine code that the Kernel can interpret. What is great is that there are multiple shells available for each user to use, depending upon their needs and / or desires. You may chose the shell that you desire, and switch between shells as necessary. The most common (and default) shell is the Bourne Again Shell, known as **bash**. More information on the various shells is given below and in later chapters.

A special shell is the GUI interface known as X Windows, more appropriately called just “X”. As a shell, it interprets not only what the user types in, but also mouse movements and clicks. But it is not finished – it must also generate the screen and recognize when the mouse cursor is over a specific area and then relate the mouse click into an action. Thus the X Window shell is quite complex and does utilize a large amount of processor power. What is great, in Unix or Linux, we do not have to utilize it – try that in Microsoft.

### 2.2.4 Applications

Finally, we have the many user applications. There are a few that reside inside the shell, that provide you the most minimal ability to move around the file system, namely the ability to change directory and list a directory’s contents – all other commands must be created by an external program. The commands that reside within the shell are referred to as internal commands. Over the years, programmers have observed a need to perform various functions, and then

created new programs to perform that function. Now, you are able to call on the multitude of various programs to do virtually any task you need – or create a new one if you deem appropriate.

A basic philosophy of the Unix concept is to create an application that does one thing – and does it very well. Multiple commands may then be coupled together to perform a more complex task. This concept of simplicity and coupling of commands is not available in Microsoft Windows.

### **2.3 System Administration**

An individual that maintains a network is considered a **network administrator**. You are not necessarily the system manager, but you do have duties, which entail the maintenance of the system and its users.

As such, you have immense responsibilities. The level of such may be dependent upon factors such as:

- Expertise
- Seniority
- Organization size

Tasks, which you may be required to perform, include:

- Creating user accounts
- Hardware maintenance
- Software installation and upgrades
- End user training
- Network backup
- Network optimization
- Network Documentation
- Establishing and maintaining procedures and policies
- Establishing and maintaining network security and privacy
- Emergency recovery
- Planning network growth
- Insuring management is aware of system status, growth and operating requirements
- Maintaining an up to date knowledge of the industry and technical trends
- Working knowledge of multiple operating systems (Windows, NetWare, Unix, Linux ...)
- Maintaining security updates
- And other as may be deemed appropriate and necessary

An important part of being a system administrator is that of maintaining a high ethical value. One deals with and has control of a companies computer system – and all of the data and users that reside and utilize. Your responsibility can lead to both the downfall of a user and of a company.

Since a system administrator, when logged on as the network administrator (**root**) has network privileges that give unrestricted access to information and operation. One has the ability to:

- Read the e-mail of various users and their system files
- Read and alter company or personal files
- Issue messages as another user.
- Erase system files
- Delaying maintenance to the system
- Minimizing assistance at the help desk
- Allowing security measures to become lacks

Doing any of the above must be considered of the highest unethical actions. Breaching the trust of you by either the users or the company may be grounds for immediate dismissal. Also, divulging information obtained while on the job to anyone without authority is a **FEDERAL CRIME!**

A system and administrator should be open to the system's operation and setup, within limits of system security and individual's privacy. One should not design a system that others are incapable of figuring out – just to insure “they can't fire me” (don't worry, they will find a way!).

Remember the following:

- Excellent jobs are always available to the administrator that:
  - is well trained
  - is technically competent
  - works to assist others
  - maintains a reputation for high ethics
  - endeavors to expand your continual growth in education
- Never rely on “job security” – there is nothing of the sort

For more information as a system administrator, one should visit the **System Administrators Guild (SAGE)** at <http://www.usenix.org/sage/>. This site contains information regarding:

- Job and salary profiles
- Local user groups
- Technical information
- Events and conferences

## **2.4 User Interface**

After your system has been installed and is operational, you now have to use it. This requires that you input information to the operating system. There are two means by which we can communicate with the OS.

### **2.4.1 CLI Mode**

The Command Line Interface Mode allows the user to directly type in commands to the operating system. This input is normally referred to as the “Stdin” – or **Standard Input**, more commonly known as the keyboard. The normal output or response from the command is directed to the user's monitor, normally referred to as the “Stdout” – or **Standard Output**. In the CLI Mode, the computer performs an action directly in response to a command that the user has initiated. In general, the CLI mode does not provide fancy graphics, but

some programs may provide simple graphic designs to improve the user functionality. Not all programs or application commands require an output to the monitor, they just do their thing and return the prompt. For example, if you direct the system to copy a file to another location, it performs the action and returns the user prompt.

### **2.4.2 GUI Mode**

Two common options are typically available to the user when utilizing a Graphical User Interface (GUI). They are KDE and GNOME. Others are also available, but will not be utilized during this instruction format. In general, both KDE and GNOME appear to be similar and both work in a similar manner, but each have their own strengths. For these set of exercises, either will work just fine. If working in the GUI mode, you will observe a small monitor at the bottom of the screen; this starts a terminal mode (XTerm), where the CLI commands may be typed in. Later versions of Red Hat and other distributions may not include the icon, but it may be accessed via either the GNOME or KDE button.

KDE has a big K in the bottom left corner, whereas GNOME has a big foot. As a matter of routine, the K for the KDE screen is referred to, but they are to be considered equivalent for the exercises. If you are using GNOME instead of KDE, the instructions will be generally the same. In later versions of Red Hat and Fedora, you may observe a “Red Hat”.

## **2.5 Command Line Interface**

How does a computer work? You may have previously learned that it functions on binary signals, but you need to understand that those bits actually represent commands. Today’s computers utilize commands that are 16 to 32 bits long, and may be augmented with additional information such as an address.

These commands were developed into a language called Assembler. A command comprises one or more bytes, called a word. Because one is able to work at the level of the processor, the most efficient coding of programs may be achieved – but the cost is a very intimate knowledge of a given processor. Commands for one processor are not necessarily transferable to a different processor.

Now we have the problem of converting what you type on the keyboard to something that the computer can understand. For instance, if you wish to display the contents of a file, different operating system commands might be:

DOS:	type filename
Unix / Linux:	less filename
MS Windows:	notepad filename (in an editing mode)

In order to put this into a machine language that the computer can understand, we must utilize an interpreter. When we utilize the keyboard, we use a Command Line Interpreter called a shell. In like manner when using a GUI (windows) interface, we need an interpreter that can recognize the position and button clicks of a mouse and convert that information into a machine command. Coding to create a GUI interpreter is considerably more complex than that of a



Command Line Interpreter (CLI). The program that is used to interpret our typing and convert it into a machine language command is called a shell.

When using either Unix or Linux, there are several shells available. Today the most common is called BASH, or Bourne Again Shell. This is an improvement over the previous shell, called the Bourne Shell (or just Shell). For various reasons, different shells have been developed that have specialized attributes, making the performance of a user easier. Common shells that exist include:

<b>sh</b>	Bourne shell
<b>bash</b>	Bourne again shell
<b>tcsh</b>	t Shell
<b>csch</b>	c Shell
<b>pdksh</b>	Public Domain Korn Shell
<b>zsh</b>	z Shell
<b>ash</b>	a Shell

Each of these reside in the **/bin/** directory. **bash** is one of the most popular, as it is able to support a vast number of general requirements, and is an enhancement of the Bourne shell. Sometimes it is better to have a specialized shell that will make data input easier, such a shell is the **c shell**, which was developed specifically for those working in the C programming language. **Bash** is the default shell in Red Hat (and most other Linux systems) that is set upon each new user under Linux, but the administrator may modify this if desired. The Bourne shell (sh) was originally developed by AT&T for Unix in the late 1960's. The user is able to switch between the different shells by simply keying in the shell mnemonic (e.g. csh).

### 2.5.1 DOS vs. Unix / Linux <sup>1</sup>

A few of you may have grown up or learned the original Microsoft Operating System, DOS. DOS was created long after Unix was a well established operating system in the operating world. There are many similarities between the two systems in their basic command structure. Table 2.1, MS-Dos verse Unix / Linux is a list of the most fundamental commands:

---

<sup>1</sup> Official Red Hat Linux User's Guide, Wiley Publishing

Table 2.1: MS-DOS vs. Unix / Linux Commands

<b>MS-DOS</b>	<b>Unix /Linux</b>	<b>Command Function</b>
copy	cp	Copies a file to a new name or location
move	mv	Moves a file
dir	ls	List a directory's contents
cls	clear	Clear screen
exit	exit	Exit from present application
date	date	Display or modify the date and / or time
del	rm	Delete or remove a files
echo	echo	Displays the specified message to the screen
edit	pico / vi	Basic text editor
fc	diff	Compare contents of a file
find	grep	Search for a text string in a file
format a:	mformat	Format a floppy disk in DOS FAT format
command ?	man command / info command	Display command syntax and options
mkdir	mkdir	Create a new directory
more	more / less / cat	Displays the contents of a file
ren	mv	Renames a file or directory
chdir	pwd	Displays the Path to the Working Directory
cd pathname	cd pathname	Changes to the designated directory
cd . .	cd . .	Changes to the parent directory of the present directory
time	date	Displays the time and date of the system clock (not hardware clock)
mem	free	Displays amount of available and utilized RAM

## **2.6 Graphical User Interface**

After all of the discussion about learning and using a command shell, many users are just turned off by having to learn “all that stuff”. In today’s society, it is much easier to see an object, comprehend what it is suppose to do, and then click on it. Linux supports the Graphical User Interface (GUI) for common applications. The benefit of the Linux GUI is that there are several versions that the user may choose from. Choices range in look and feel to application section. Because this is a significant topic of its own, a whole chapter is dedicated to the X Windows system.

Be aware that any Windows system (Microsoft or Unix) does not have the power to manipulate data like you have in the Command Line Interface. Data is best manipulated at the command line using command utilities that provide the user with the ability to modify data and the ability to perform tasks that are impossible to perform using Microsoft without extensive programming.

## **2.7 Pathnames and Directories**

A Directory is a symbolic area of the storage media that contains files and additional directories. Although the files and subdirectories may be randomly distributed across the media, one may view the media, normally a partition on a hard drive as a file cabinet.

The first level of directories is like the cabinet drawers. These contain additional documents and directories, which in turn may contain more documents and directories. For our discussion, from your experience from the MS Windows world, a folder and directory are the same thing. The depth of the directories is normally only limited to the user requirements.

We specify the location of a file by giving its full path, or the names of all directories that are required to navigate to the file – kind of like a road map. The directory names are separated by the forward slash (/).

When we designate the full path, we then specify the “**Fully Qualified Filename**”. It is also called an **absolute pathname**. An absolute pathname is one that starts at the very top of the directory structure and specifies each directory. The first character is written with the forward slash /.

**Relative Paths** designate a file's location relative to one's present location, without specifying one's current directory. The relative path must always be a subdirectory set with respect to the current location, and **does not include the forward slash**.

A directory location directly below a specified directory is said to be a **Child Directory**. The reference directory is the **Parent Directory** of the child.

From wherever one is, the Absolute Path will take one to the designed location; if one is in the parent directory, then only the child directory name, without the slash (/) and the rest of the directory names. Examples are:

<b>Absolute Path</b>	<b>/etc/sysconfig/network-scripts</b>
<b>Relative Path</b> (starting in /etc)	<b>sysconfig/network-scripts</b>

There are two shortcuts that may be utilized to specify a file's location. They are:

- The current working directory – it contains the contents of the present directory.
- .. The parent directory – it contains the path to the directory immediately above the present directory.

It is often necessary to specify a file's current location relative to our present location – even if it is located in our present location. This is shown as:

**\$ command ./filename**

This states that the desired command is to perform its operation on the file “filename”, which resides in the current directory. Normally, this specific process is not required, but if a command or script is issued from the directory that one is currently located in, then one does require the “.” preceding the command.

**\$ ./command**

The most common example is the running of a script that has been written in bash, perl, python, or one of the other scripting languages. These applications typically do not reside in the standard environmental \$PATH of the user.

## **2.8 File Storage**

So far the discussion has been about how to navigate around the file system, but the question should also be, how does Unix / Linux keep track of where a file is located?

When using Microsoft, files are tracked using a **File Access Table**, more commonly known as the **FAT** table. This specifies where on a hard drive the desired file is located. Microsoft also writes data to the next available spot on the drive, and hence if the space is not large enough, the file will be spread across multiple sectors. This results in the file becoming fragmented, and the need to defragment the hard drive on a periodic basis. Unix and Linux do not write data to the hard drive device in the same way, rather than finding the first available sector, the system looks for the first available block of space that is large enough to hold all of the file. Thus, the file is stored on the drive in a contiguous manner. If the data was previously stored on the drive in a different location, that space is freed up for use by another process. It is not a requirement to defragment a hard drive unless the drive becomes more than 90% full, at which time it is necessary to add a new drive.

The next question is how does Unix / Linux keep track of the file's location? This process is known as **inode**, or **index node table**. A table is maintained on the hard drive, and written to memory on bootup to enhance speed, which specifies the files location. This table includes information such as:

- Inode Value
- File Type
- Links
- Access Permissions
- User (Owner) ID
- Group ID
- Date Created
- Date Last Modified
- Date Changed
- Size
- Blocks
- Storage Device ID
- Block ID (Track, Sector, Cluster)
- ...

Thus, when the system requires access to a file, it looks up the inode, then goes to the physical location, where it reads the data. This method of storing data is very efficient in speed and overall operation, although it might not be optimum utilization of the disk space.

## **2.9 Filenames**

DOS and VMS use filenames that have only one extension – such as .com, .exe, .mp3, .doc, .bmp and many others. Unix and Linux do not care, the dot character (.) is just part of the filename with no special significance. Thus you can have a file name such as “mine.yours.ours”. For our ease in viewing files, we commonly assign a “standard” extension to a file name, so that we can easily interpret what type of file it is, but to Unix and Linux, it does not matter. The type

of file that the code is is represented by the first several bytes of the code. We will learn more in another chapter about this.

Filenames that begin with a dot are classed as **hidden**. Thus they are not normally viewed when doing a directory listing. This does not mean you can not see the files, just that you must use a special command to see them.

There are a few requirements for the creation of a filename:

1. First character should not be a hyphen ( – ).
2. The characters ?, \*, &, (, ), [, ], <, >, “space”, and “tab” should not be included in the file name, as they have special command line meaning.
3. If you need one of the special characters in a filename, enclose the filename in double quotes (“ ”).
4. Do not use non-printing ASCII characters. (These are no-see-ems ;-)
5. A system is normally configured to support the ASCII code for character display, but may be configured to support different characters of foreign languages or even Unicode (16 bits / character).
6. Filename must be unique. If you create a new file with a name that is the same as that of an existing file (where it will be stored), the original existing file will be overwritten.

## **2.10 Definition of Root**

There are many things that are confusing about Unix and Linux – and at the top of the list is the term **root**. In fact, it has four different meanings, so one has to listen carefully when someone uses the **root** word, to make sure that you understand the proper context.

The four different meanings of **root** are:

### **2.10.1 The root**

The directory system is a tree structure. The top of the structure is referred to as **the root**. The **root** is also sometimes referred to as to as the **Top Level Directory (TLD)**. To help distinguish this from the other meanings, it might be a good practice to use the word “the” in front of it – thus “**the root**”.

### **2.10.2 Administrator root**

This is the administrator of the system. This is the administrator’s username. As administrator, you have total power over the system operation and configuration. You can go anywhere you want, create anything you want, and erase to do anything you want. That includes the whole system. But once a file is erased, it is GONE! There is no recovery, so be very careful. Other users may also be assigned the status of root, even though they have a different username – this can be both good and bad. Under your normal username you can set yourself to have total root privileges – which security wise is a very bad idea. Be very cautious about who you hand out the root privilege to.

### 2.10.3 Home Directory **/root**

This is the administrator's home directory. It is located directly below the root directory, and allows the administrator additional privileges above any other user. Access to this directory is restricted to the root administrator, no one else may even read its contents. Special privileged files are maintained in this directory.

### 2.10.4 Group **root**

In addition to the user called **root**, there is a group called **root**. They are not the same entity. A group is made up of other users and groups. Various other users may be added as members to the group root, but this should be done with caution.

For every user on the system, there is a group by the same name. Additionally, we can have additional groups that we can create separately. The process of creating groups and assigning users to it will be discussed in a later lab.

For now, there is a specific group called **root**, which has certain extended privileges. To the extent that you the administrator desire, you may assign rights for a member of the group **root**. What is very important is that the user **root** and the group **root** have many more privileges than the normal user – you must exercise caution when handing out these rights. This might be to allow them to access certain directories that the other users are not allowed, or to execute a specific set of commands. To assign another user the rights to the group root, you need to add them to the group (again, the process is covered later).

## 2.11 Home Directory

Every user on a Unix / Linux system has their own private directory, referred to as the **home directory**. This is where their personal configuration and data files are maintained. All users, except the administrator, have their home directory located in the **/home** directory. When a user first logs onto a system, they are automatically opened to their home directory, thus when user jdoe logs on, he will be in the directory **/home/jdoe**.

The only users allowed into an individual's home directory are the specified user and the administrator.

The administrator also has a home directory, but it is located the **/root** directory. Only the administrator is permitted to access it.

When the user first logs onto a system, they are immediately placed into their home directory. For a normal user, the prompt that appears will typically be a **"\$"**, whereas the root administrator will have a **"#"**.

## 2.12 Password Security

Password security is of prime importance to any system. This is your first line of defense to an unauthorized user attempting to gain access to your system. Originally, Unix used the DES to encrypt passwords, and maintained them in a file called **"/etc/passwd"**. Passwords were limited to 8 characters. Today, most Linux systems default to a newer encryption method called MD5, which allows a password of up to 256 characters – try remembering one that long. Additionally,

to further enhance security, the actual encrypted password was removed from the `passwd` file and transferred to a new file called `/etc/shadow`. This added not only a more secure placement of the password, but added the feature of password aging. An installation does not require the creation of the shadow file for older distributions – this is not a normal process. A system should typically be set up with the shadow file. The newer Fedora Core distribution does not allow the option to not create the shadow file, it is automatically created.

Although in a classroom environment you may be instructed what password to use, this must be emphasized as not the normal routine. In a normal operating system, your personal password must be kept private – do not give it out to anyone.

### **2.13 Shell Interpreters**

When we are at the Command Line Interface (CLI), we use a Shell that interprets our keystrokes, converting them into appropriate bits that the kernel can execute. Several different shells exist that make this process easier for us.

The CLI is indicated by a prompt. In general as a normal user, when you are using the bash shell in Linux, your prompt will appear as:

***[username@hostname present-directory] \$***

This specifies that your username on the host is presently at a specified directory. If it ends in a `#`, then you are the root administrator, otherwise you will see a `$`, which indicates you are a normal user. From this you are able to enter various commands. The root's prompt will be (although this will vary by distribution):

***[root@hostname present-directory] #***

The original shell was developed by Bourne for the Unix system back in the early 1970's. Since then others have developed alternatives. This may seem unnecessary, but in another perspective, this is one of the powers of the Unix / Linux concept – you can develop system controls that benefit you and are not dictated to by a single vendor.

Other common shells include the sh “shell”, korn shell, tsh shell, csh shell, tsh shell, zsh shell, and the bash shell. Today, the bash shell is the default for new users.

There are pros and cons to each which may be argued by those who use them. A few comments on some include:

- |             |  |
|-------------|--|
| sh and korn | Some of the original shells that provide basic functions.  |
| csh         | A shell that supports the development of C Language program code.  |
| bash        | A shell that has been updated from the original sh and korn shell. By default, this is the shell that Linux and most other systems operate in. |

Using the **usermod** command, one can change the default shell, this will be investigated in depth later.

Although the other shells have been developed, it is left to the user to investigate them and evaluate if they desire to utilize a shell other than bash. Bash will provide you better than 99 % of all of your requirements.

Common features of Bash include:

- Multi commands per line, separated by a semicolon ( ; )
- Command Execution
- [ ] Matching of possible characters in a filename
- | Support of the pipe command
- & Execute command in the background
- \* Wildcard, match on any set of characters in filename
- ? Wildcard, match on any single character in filename
- > Redirect standard output to a file, overwriting file or creating a new file
- >! Forces the overwriting of the file
- < Redirect standard input from a file or device to a program
- >> Redirect standard output and append to specified file
- 2> Redirect standard error to a file
- 2>> Redirect and append stand error codes to a file
- 2>&1 Redirect the standard error code to the standard output
- >& Redirect the standard error to a file or device
- |& Pipe the standard error as an input to another command
- ! Called the “Bang”
- #! Called the “**SheBang**”, used in scripts to indicate which script program (Bash, Perl, Python, ...) to use
- **TAB** Use to complete the typing of a filename
- \ Called the “**whack**”, used to continue code, comment on the next line, or to escape out of the present text mode and issue a special character within an echo command.
- # Pound Sign, or Hash, used to specify a comment
- ; Semicolon, alternative to commenting out a line of code

Bash also supports the creation and execution of scripts. Using the above symbols and various commands, one may write programs to perform various executions in a batch format. Creating script files will be covered in a later lab.

Some of the shell Command Line Interpreters are:

1. Bash (Bourne Again Shell)
2. Korn
3. Csh
4. Sh (Bourne)
5. Tsh
6. Zsh
7. Ash



### 2.13.1 Shell Comparison

The following table provides a comparison between several of the available shells.

Table 2.2: Shell Comparison

	Bourne Again Shell (bash)	Borne Shell (sh)	C Shell (csh)	Korn Shell (ksh)
Command history	Yes	No	Yes	Yes
Command Alias	Yes	No	Yes	Yes
Scripting	Yes	Yes	Yes	Yes
File Name Completion	Yes	No	Yes	Yes
Command Line Editing	Yes	No	Yes	Yes
Job Control	Yes	No	Yes	Yes
Programming Features	Most	Few	Minimal	Many
User Friendly	Very	High	Basic	Low

### 2.14 Client – Server Concept

Many local networks (LANs) are typically built around the concept of everyone running their own system, sharing directories with appropriate information as desired. This is called a peer-to-peer network.

An alternative is to set up a server to maintain the client information. User's wishing to access the data must log onto the server in order to access or modify the information. This is referred to as a Client – Server Network.

The advantage of a Client – Server network is the centralization of data – hence everyone is working off of the same base. Typically when a user has opened a data file, it is locked so that another user may not open the file, thus preserving its integrity.

### 2.15 Commands Used in this Chapter

No commands were utilized in this chapter.

**2.16 Chapter Review Questions**

1. Unix and Linux interface all devices as what?
  - a. Directories
  - b. Files
  - c. Devices
  - d. Terminals
2. What is the central software core called?
  - a. Operating System
  - b. Shell
  - c. Kernel
  - d. Major
3. What software interprets user keystrokes into code for the Kernel?
  - a. Terminal
  - b. Bash
  - c. Korn
  - d. Shell
4. The network administrator is responsible for which specific network tasks?
  - a. Bootup
  - b. Optimization
  - c. Documentation
  - d. All of the above
5. If the network administrator divulges information of another user or of the corporation it is considered what?
  - a. Federal Crime
  - b. Fraud
  - c. Open Source
  - d. Espionage
6. What mode accepts input from the stdin?
  - a. Command Line
  - b. Graphical Interface
  - c. Mouse
  - d. Keyboard
7. What does bash stand for?
  - a. Bourne Shell
  - b. Beginning Shell
  - c. Beginning Shell
  - d. Bourne Again Shell
8. A fully qualified filename specifies what?
  - a. File's Location
  - b. File's Path
  - c. Full pathname and file to a file's location
  - d. File's Directory
9. What is the filename that contains the contents of a directory?
  - a. pwd
  - b. "."
  - c. ".."
  - d. dir

10. How would a user commonly specify a filename that is in their present directory?
  - a. ls
  - b. “.”
  - c. filename
  - d. ./filename
11. What characters should not be used in a filename?
  - a. ?, \*, &, (, ), [, ], <, >, “space”, and “tab”
  - b. “space”, “tab”
  - c. -, +, =, ^, %
  - d. !, @, #, &
12. What is meant by “the root”?
  - a. Top of the directory structure
  - b. Administrator’s Home directory
  - c. The Administrator’s login name
  - d. A Group of users with administrative rights
13. What is meant by the term “root administrator”?
  - a. Top of the directory structure
  - b. Administrator’s Home directory
  - c. The Administrator’s login name
  - d. A Group of users with administrative rights
14. Specify the fully qualified path to the administrator’s home directory.
  - a. root
  - b. /root
  - c. /home/root
  - d. /home
15. What entity may be assigned some rights of the administrator?
  - a. root
  - b. /root
  - c. /group
  - d. group root
16. What is the home directory of the user jdoe?
  - a. /root
  - b. root
  - c. /home
  - d. /home/jdoe
17. The user jdoe logs onto a system, what is their prompt under bash?
  - a. [jdoe@hostname /home/jdoe]\$
  - b. [jdoe jdoe]\$
  - c. [jdoe@hostname jdoe]#
  - d. [jdoe@hostname jdoe]\$
18. Using MD5 password encryption, how long may the password be?
  - a. 8 characters
  - b. 16 characters
  - c. 64 characters
  - d. 256 characters

19. What feature of bash directs execution of a program in the background?
  - a. @
  - b. \*
  - c. &
  - d. =
20. What is the “>” character?
  - a. Appends output to Null
  - b. Appends output to a new file
  - c. Sends output to a new file
  - d. Sends output to the Null file
21. What is the “>>” character?
  - a. Appends output to Null
  - b. Appends output to a new file
  - c. Sends output to a file
  - d. Sends output to the Null file
22. What is the “!” character called?
  - a. bang
  - b. gong
  - c. sha-bang
  - d. bing
23. What is the “#!” character called?
  - a. bang
  - b. gong
  - c. sha-bang
  - d. bing
24. What does the TAB key do to a filename?
  - a. Advances the cursor 8 characters
  - b. Advances the cursor to the next tab location
  - c. Completes the pathname of the file
  - d. Completes the filename characters
25. What character is used to continue a comment or code on the following line?
  - a. #
  - b. ;
  - c. &
  - d. \

## Chapter Index

A		F	
a Shell	9	FAT	12
Absolute		File	
Pathname	11	Relative Location	11
Absolute Pathname	11	/etc/fstab	3
Administrator		/etc/mtab	3
Privileges	7	/etc/passwd	14
Responsibilities	6	/etc/shadow	15
Tasks	6	File Access Table (FAT)	12
Unethical Actions	7	File Operating System	3
Administrator root	13	Filenames	12
Applications	5	Files	
Assembler	8	Hidden	13
		Naming Criteria	13
		Folder	11
Bang	16	Forward Slash	11
bash	9	Fully Qualified Filename	11
Bash	5		
Bash Features	16	G	
Bourne Again Shell	5, 9	Graphical User Interface	10
Bourne Shell	9	Group root	14
		GUI	5, 10
		GUI Mode	8
		H	
		Hardware	4
		Home Directory	14
		Home Directory /root	14
		I	
		Internal Commands	5
		Interpreter	8
		K	
		KDE	8
		Kernel	5
		Kernel Hardware Interface	5
		M	
		MD5	14
		N	
		Network	
		Client-Server	17
		P	
		parent directory	11
		Password Security	14
		Path	
		Absolute	11
		Pathnames and Directories	10
		Peer-to-Peer Network	17
B			
Bang	16		
bash	9		
Bash	5		
Bash Features	16		
Bourne Again Shell	5, 9		
Bourne Shell	9		
C			
c Shell	9		
Character			
; 16			
# 16			
Child Directory	11		
CLI Mode	7		
Client – Server Concept	17		
Command			
usermod	15		
Command Line Interface	8, 10		
Command Line Interpreter	9		
current working directory	11		
D			
Definition of Root	13		
defragment	12		
Device Drivers	3		
Directory	10		
.	11		
..	11		
/bin	9		
/home/username	14		
/root	14		
DOS vs. Linux Commands	10		
DOS vs. Unix / Linux	9		

Prompt		Stdin	7p.
# 15		Stdout	7
\$ 15		System Administration	6
Public Domain Korn Shell	9	T	
R		t Shell	9
Redirector	4	The root	13
Relative		The Linux Kernel	4
Pathname	11	Top Level Directory	13
Relative Path	11	U	
S		User Interface	7
SAGE	7	W	
Shebang	16	whack	16
Shell		X	
Relative Path	11	XTerm	8
sh	9, 15	Z	
Shell Comparison	17	z Shell	9
Shell Interface	5	;	
Shell Interpreters	15	; 16	
Standard		/	
Error	4	/home	
Input	4, 7	Directory	14
Output	4, 7	#	
Standard Input / Output	4	# 16	