1. Introduction to Linux operating system

1.1 Introduction to Operating System

Operating system is software that is required to take work from computer and it is an interface layer between hardware and user application. A user application interacts with operating system and makes aware the operating system for its achievements. These achievements are understood by the operating system and activate mapped tasks whereby hardware works.

Let us with an overview of operating system and how Linux became the operating system it is today. We will discuss past and future development and take a closer look at the advantages and disadvantages of this system. We will talk about distributions, about Open Source in general and try to explain a little something about GNU.

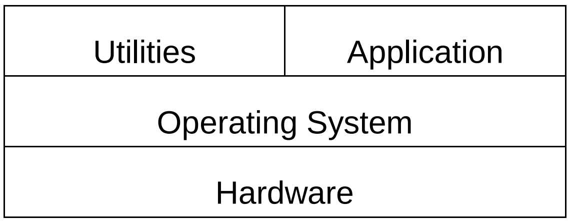


Figure 1.1A Typical Operating System

An Operating System is responsible for the following functions

- Device management using device driver
- Process management using process and threads
- Memory management
- File Management

In addition, all operating systems come with a set of standard utilities. The utilities allow common tasks tom be performed with as

- Being able to start and stop processes
- Organize files into sets as directories
- View files and sets of files
- Edit files
- Communicate between processes

1.1.1 Kernel

The kernel of an operating system is the part responsible for all operations. When computer boots up, it goes through some initialization activities, such as checking memory. It then loads the kernel and switches control to it. The kernel then starts up all the processes needed to communicate with the users and the rest of the environment (e.g. the LAN). The kernel is always loaded into memory, and the kernel functions run continuously, handling processes memory, files and devices. The traditional structure of a kernel is layered system, as Unix where all layers are part kernel, and each layer can talk to only a few other layers, application programs and utilities lie above the kernel.
The Unix Kernel looks like (Figure 1.2)

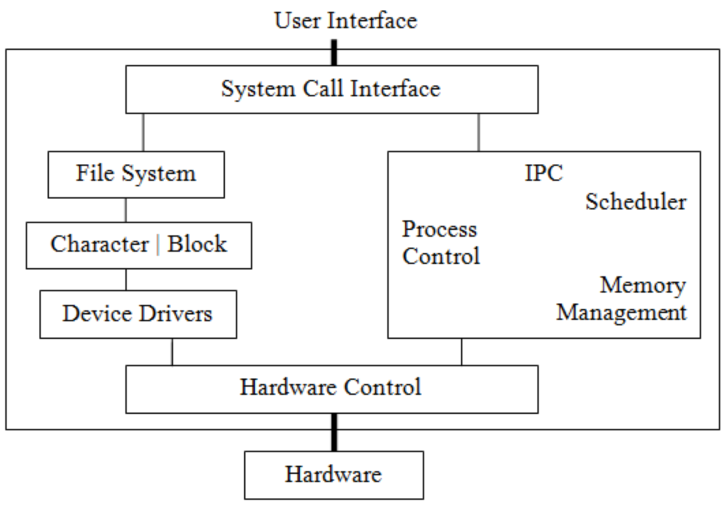


Figure 1.2UNIX Kernel

Most of the currently prevent Operating System use instead a micro kernel, of minimum size. Many traditional kernel operations are made into level services. Communication bearing service is often carried out by an explicit message passing mechanism. Mach is one of the major micro-kernel operating system whose concepts are used by many others (se Figure 1.3)

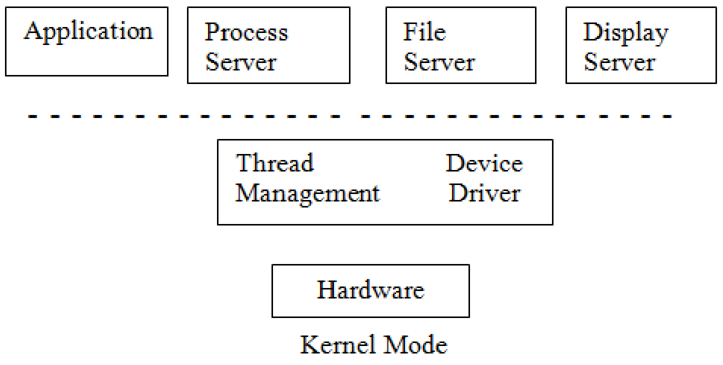


Figure 1.3 Micro Kernel Architecture

Some systems, such as Windows NT (Figure 1.4) use a mixed approach [Gray Nutt]

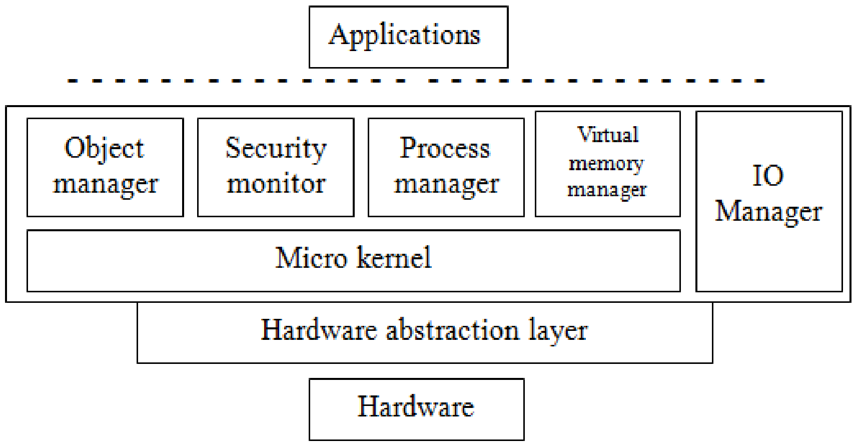


Figure 1.4 Windows NT Kernel

1.1.2 Distinguished Applications

An Operating System has been describes as an “application with no top”. Other application Interact with it, through a large variety points. In order to use an OS, you need to be supplied with at least some application that already uses these entry points.

All Operating Systems come bundled with a set of “utilities” which do this. For example

- Windows95 has a shell that allows programs to be started from Start button. There is a standard set of applications supplied
- MSDOS starts up with COMMAND.COM to supply a command line prompt, and a set of utilities
- Unix has a set of command line shells and a huge variety of command line utilities
- X-Windows supplies a login shell (xdm). Other supply file managers, session managers, etc. which can be used to provide a variety of interfaces to the underlying Unix/POSIX system

1.1.3 Command Interpreter

User interacts with operating system through the intermediary of interpreter. This utility responds to user Inputs in the following ways:

- Start or stop applications
- allow the user to switch control between applications
- Allow control over communication between an application and other applications or the user.

The command interpreter may be character-based, as in the UNIX shell or MSDOS COMMAND.COM, or can be a GUI shell like the Windows 3.1 Program Manager. The interpreter may be simple, or the power of a full programming language. It may be imperative (as in the Unix Shell), use messages passing (as in Applet Script) or use Visual programming metaphors such as drag-and-drop for object embedding (as in Microsoft’s OLE). It is important to distinguish between the command interpreter and the underlying Operating System. The command interpreter may only use a subset of the capabilities offered by the Operating System; it may offered complex skill or be intended for novices.

1.1.4 Differences between DOS and Unix

- UNIX is multi user and multi-tasking operating system where as DOS is single user, single task system.
- All the commands in Unix should be given in lover case while the DOS commands are case insensitive.
- Unlike Unix, DOS is more virus prone.
- Processor will be in protected mode in UNIX whereas DOS uses unprotected mod.
- DOS uses only 640KB of RAM during boot time unlike Unix which uses all the available RAM.
- UNIX needs an administrator which is not the case with DOS.
- UNIX employs time sharing operating system. Whereas DOS supports a pseudo time sharing knows as Terminate and Stay Resident (TSR) programs.
- UNIX supports both character user interface and graphical interface(X Window) unlike DOS which supports only character user interface.

- User requires legal username and password to use Unix machines. DOS systems can be used by anyone without any username and password.
- UNIX used single directory three (/) irrespective of how many drives or partitions are there. Where as in DOS, a separate directory tree for each partition.
- UNIX supports NFS to Share files.
- Till recently, DOS did not have proper WWW browser.

1.1.5 Differences between DOS and Unix

- UNIX is multi user and multi-tasking operating system where as DOS is single user, single task system.
- All the commands in Unix should be given in lover case while the DOS commands are case insensitive
- Unlike Unix, DOS is more virus prone
- Processor will be in protected mode in Unix whereas DOS uses unprotected mod.
- DOS uses only 640KB of RAM during boot time unlike Unix which uses all the available RAM
- Unix needs an administrators which is not the case with DOS
- UNIX employs time sharing operating system. Whereas DOS supports a pseudo time sharing knows as Terminate and Stay Resident (TSR) programs.
- UNIX supports both character user interface and graphical interface(X Window) unlike DOS which supports only character user interface.

- User requires legal username and password to use UNIX machines. DOS systems can be used by anyone without any username and password
- UNIX used single directory three (/) irrespective of how many drives or partitions are there. Where as in DOS, a separate directory tree for each partition.
- Unix supports NFS to Share files
- Till recently, DOS did not have proper WWW browser.

1.2 Linux Distributions

- RedHat/CentOS
- Debian
- Ubuntu
- Fedora
- OpenSUSE
- Mandriva
- Slackware
- FreeBSD
- Mint
- PCLinuxOS

1.3 Introduction to Linux File System

File System: A system of classifying into files (usually arranged alphabetically).

In a computer, a file system (sometimes written filesystem) is the way in which files are named and where they are placed logically for storage and retrieval. The DOS, Windows, OS/2, Macintosh, and UNIX-based operating systems all have file systems in which files are placed somewhere in a hierarchical (tree) structure.

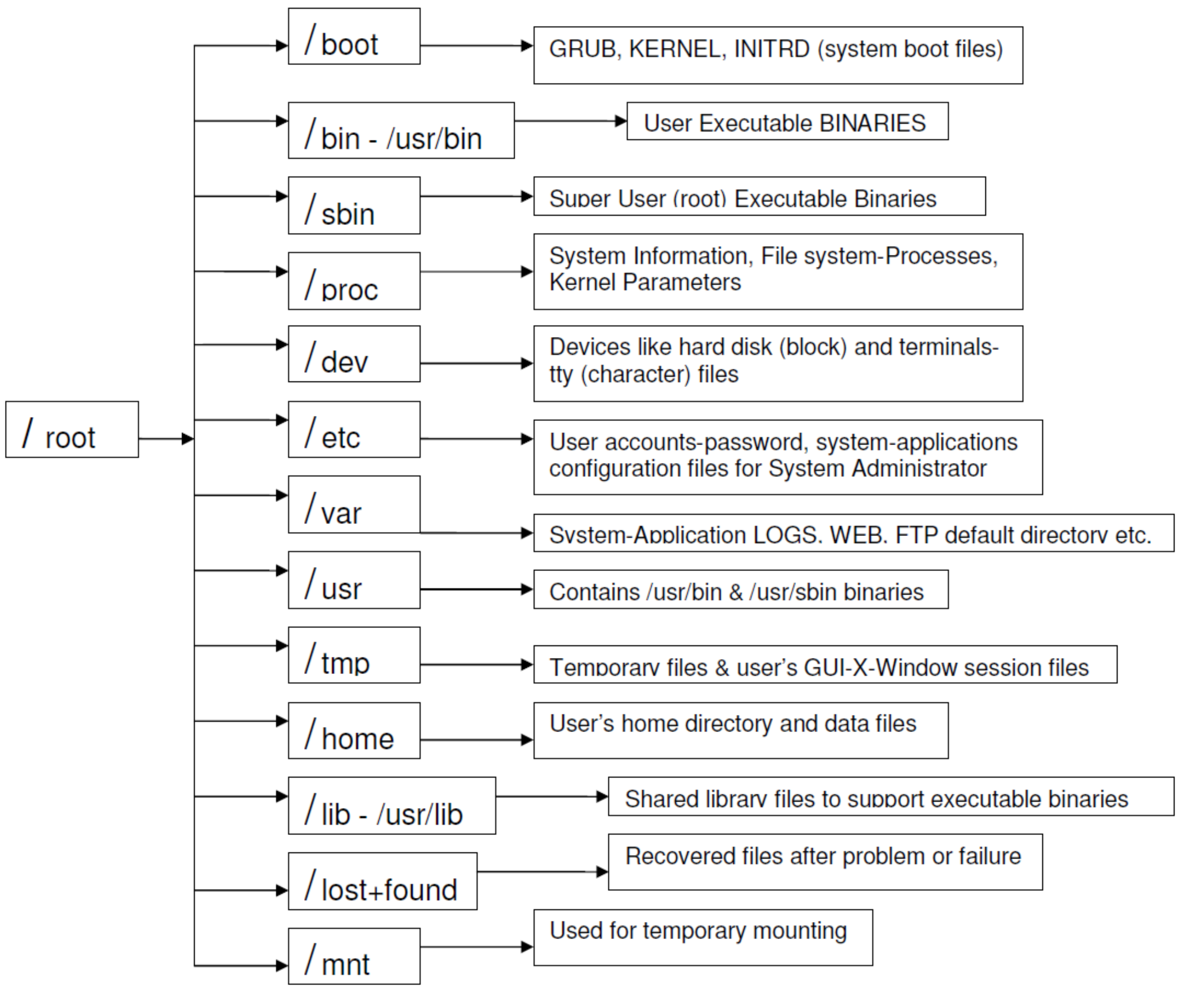

/ root
/ boot
GRUB, KERNEL, INITRD (system boot files)
/ bin - /usr/bin
User Executable BINARIES
/ sbin
Super User (root) Executable Binaries
/ proc
System Information, File system-Processes, Kernel Parameters
/ dev
Devices like hard disk (block) and terminals-tty (character) files
/ etc
User accounts-password, system-applications configuration files for System Administrator
/ var
System-Application LOGS, WEB, FTP default directory etc.
/ usr
Contains /usr/bin & /usr/sbin binaries
/ tmp
Temporary files & user's GUI-X-Window session files
/ home
User's home directory and data files
/ lib - /usr/lib
Shared library files to support executable binaries
/ lost+found
Recovered files after problem or failure
/ mnt
Used for temporary mounting

Figure 1.5 Linux File System

1.3.1 boot

This directory contains everything required for the boot process except for configuration files not needed at boot time (the most notable of those being those that belong to the GRUB boot-loader) and the map installer. Thus, the /boot directory stores data that is used before the kernel begins executing user-mode programs.

1.3.2 sbin

Linux discriminates between 'normal' executables and those used for system maintenance and/or administrative tasks. The latter reside either here or - the less important ones - in /usr/sbin. Locally installed system administration programs should be placed into /usr/local/sbin.

1.3.3 bin

Unlike /sbin, the bin directory contains several useful commands that are of use to both the system administrator as well as non-privileged users. It usually contains the shells like bash, csh, etc.... and commonly used commands like cp, mv, rm, cat, ls. For this reason and in contrast to /usr/bin, the binaries in this directory are considered to be essential. The reason for this is that it contains essential system programs that must be available even if only the partition containing / is mounted.

1.3.4 proc

/proc is not a real file system, it is a virtual file system. For example, if you do ls -l /proc/stat, you'll notice that it has a size of 0 bytes, but if you do “cat /proc/stat”, you'll see some content inside the file.

1.3.5 dev

The /dev directory contains the special device files for all the devices. The device files are created during installation. Look through this directory and you should hopefully see hda1, hda2 etc.... which represent the various partitions on the first master drive of the system. /dev/cdrom and /dev/fd0 represent your CD-ROM drive and your floppy drive.

1.3.6 etc

This is the nerve center of your system; it contains all system related configuration files in here or in its sub-directories. A "configuration file" is defined as a local file used to control the operation of a program; it must be static and cannot be an executable binary.

1.3.7 var

This directory contains variable data like system logging files, mail and printer spool directories, and transient and temporary files. Some portions of /var are not shareable between different systems.

1.3.8 usr

This directory contains user applications and a variety of other things for them, like their source codes, and pictures, docs, or config files they use. /usr is the largest directory on a Linux system, and some people like to have it on a separate partition.

1.3.9 tmp

This directory contains mostly files that are required temporarily. Many programs use this to create lock files and for temporary storage of data. Do not remove files from this directory unless you know exactly what you are doing! Many of these files are important for currently running programs and deleting them may result in a system crash.

1.3.10 home

This is directory where users keep their personal files. Every user has their own directory under /home, and usually it's the only place where normal users are allowed to write files.

1.3.11 lib

In this directory Linux keeps the shared libraries for programs that are dynamically linked. The shared libraries are similar to DLL's on Winblows.

1.3.12 lost+found

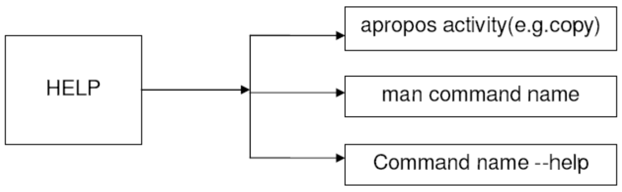
In this directory Linux keeps the files that it restores after a system crash or when a partition hasn't been unmounted before a system shutdown. This way you can recover files that would otherwise have been lost.

1.3.13 mnt

This is a generic mount point under which you mount your filesystems or devices. Mounting is the process by which you make a filesystem available to the system. After mounting your files will be accessible under the mount-point. This directory usually contains mount points or sub-directories where you mount your floppy and your CD. You can also create additional mount-points here if you wish.

2. Basic LINUX Commands

2.1 Getting HELP from LINUX



Example:

Apropos is a command to search the man page files in UNIX and Unix-like operating systems. [apropos<command_name>]
>apropos php

Man is command it is use To read a manual page for a Unix command
[man <command_name>]
>man php

Andthird is –help [<command_name> –help]
>php–help

2.2 Shell

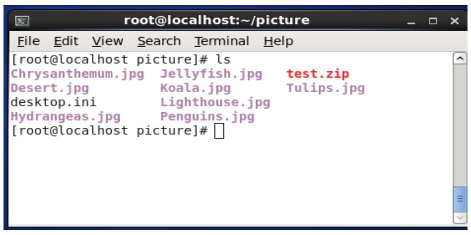
The shell is a program that takes your commands from the keyboard and gives them to the operating system to perform. In the old days, it was the only user interface available on a Unix computer. Nowadays, we have graphical user interfaces (GUIs) in addition to command line interfaces (CLIs) such as the shell.

On most Linux systems a program called bash (which stands for Bourne Again SHell, an enhanced version of the original Bourne shell program, sh, written by Steve Bourne) acts as the shell program. There are several additional shell programs available on a typical Linux system. These include: ksh, tcsh and zsh.

Why Use a Shell Prompt

Graphical environments for Linux have come a long way in the past few years. Linux functions can be completed faster from the shell prompt than from a graphical user interface (GUI). In less time than it takes to open a file manager, locate a directory, and then create, delete, or modify files from a GUI, a task can be finished with just a few commands at a shell prompt.

A shell prompt looks similar to other command line interfaces with which you might be familiar. Users type commands at a shell prompt, the shell interprets these commands, and then the shell tells the OS what to do.



```
[root@localhost picture]# ls
Chrysanthemum.jpg  Jellyfish.jpg    test.zip
Desert.jpg         Koala.jpg        Tulips.jpg
desktop.ini        Lighthouse.jpg
Hydrangeas.jpg     Penguins.jpg
[root@localhost picture]# 
```

Useful Keystrokes

While working from the command line, there are a few useful keystrokes that can help you with your session.   The following keystrokes are useful shortcuts at a shell prompt.

Keystroke	Action
[Ctrl]-[K]	While editing a command on the command line, this key combination deletes everything that has been typed in from the cursor's current position forward.
[Ctrl]-[D]	Pressing this key combination once ends the current application or process. Pressing it twice exits the shell.
[Ctrl]-[R]	At the command line, [Ctrl]-[R] searches through the command history to find the entry that starts with the letters you type.
[Ctrl]-[Z]	Suspends the current application. Entering **bg** after [Ctrl]-[Z] causes a program to run in the background.
[Ctrl]-[C]	"Kills" a program. This should be a last resort. Try stopping a program with other methods first.

2.3 Basic LINUX Commands

Following are the basic LINUX commands useful to day to day operations

No	Command	Description	Example	Descriptions
1	cd	Type cd followed by the name of a directory to access that directory. Keep in mind that you are always in a directory and allowed access to any directories hierarchically.	1. cd 2. cd - 3. cd ~ 4. cd / 5. cd /root 6. cd /home 7. cd .. 8. cd ~otheruser	1. Returns you to your login directory 2. Returns you to your previous working directory 3. Also returns you to your login directory 4. Takes you to the entire system's root directory. 5. Takes you to the home directory of the root user. You must be the root user to access this directory. 6. Takes you to the home directory, where user login directories are usually stored 7. Takes you to the directory one level up. 8. Takes you to otheruser's home directory, if otheruser has granted you permission.
2	pwd	To print current working directory on screen	pwd	It will show full path of your current dir

3	clear	To clear screen	clear	It will clear console screen
4	hostname	To print your host name screen	hostname	It will show your server name
5	date	To print current date an time		Display system date
6	cal	To print calendar on scr	1. cal 2. cal 7 2012	1. Display calendar of current month 2. Display calendar of July 20
7	bc	To start a calculator		
8	uname –	To know your kernel- machinearchitecture	uname –mi	It will show machine architecture and os architecture
9	ifconfig	To know your host's TC Address	ifconfig –a	It will display all network devices and their TCP/IP Addresses
10	ls	To list files in a director Options -l (long) -a (all) — -F (file type) -r (reverse) -R (recursive) -S (size)	1. ls –l 2. ls –Sl 3. ls –ahl 4. ls –ld 5. ls –t 6. ls –tlr 7. ls –il	1. Long list of files in a dir 2. Long list of files in a dir sorted by size Long list of archive and 3. hidden files in a directory 4. To show only directory 5. Sort by access/modify time 6. Time vise sorting of dir 7. Inode vise sorting of dir
11	touch	To create an empty file	1. touch example 2. touch file{1,2,3,4,5}	1. Create file example with 0 byte 2. Create five files with 0 bytes
12	file	It will show type of file	file example	classify a file's contents
13	cat	To read file	cat filename cat –n filename	Read file with all line nos.
14	more	To read long file with pagination	1. more filename 2. cat filename | more	
15	less	To read long file with scrolling facility	less filename cat filename | less	view text files
16	cp	Copy a file	1. cp 2. cp –I 3. cp –r 4. cp –r –P 5. cp –a 6. cp -av	1. Copy 2. Copy interactively 3. Copy recursively 4. Copy recursively & Preserve the permission 5. Copy all files with retaining permission 6. Copy all files & directories recursively with retaining permission
17	mv	Move / rename file -i (interactive) -f (force) -v (verbose)	mv file1 file2 mv file1 /home/user1/	Rename file1 with file2 Move file1 to /home/user1/
18	rm	Remove files -i (interactive) -f (force) -v (verbose) -r (recursive)	rm –i rm –r rm –f rm –rf	Remove interactively Remove recursively Remove forcefully Remove recursively & force fully

19	mkdir	To create an empty dire	mkdir –p /var/www/html/example mkdir –p /tmp/1/2/3/4/5/6/7/ mkdir –p user/{Jan,Feb,March}/{1,2,3	Create dir with permission of current user To create multiple dir inside a dir To create dir inside dir
20	rmdir	To remove an empty d	rmdirdirname	
21	tail	To read end of the file	tail filename tail –f /var/log/maillog tail -100 /var/log/messages	Read end of file Read end of file in realtime Read last 100 line of file
22	head	To read top of the file	head filename head -100 /var/log/message	Read top of the file Read 100 lines from top of the file
23	stat	Display file attribute	stat filename	
24	grep	Looks for pattern in file	grep root /etc/passwd grep –v root /etc/passwd cat /etc/passwd | grep root grep –lr password /etc/ grep –rn password /etc/ or filename	Show all matches of root Show all lines that do not match root Show all matches of root show all files in /etc which contails word password show line no & word password in /etc/ dir or filename
25	wc	Word count	wc –l filename	Shows no of lines in filename
26				
27	locate	To find files and dir fas	locate filename	Require updatedb to build index
28	sort	to sort file in alphabeticalnumerical order	sort /etc/passwd | less sort –n /etc/passwd | less sort –n –t : -k 3 /etc/passwd less	sort alphabetically sort numerically sort by position & field separator
29	cut	to cut require informati	cut –d":" –f4 /etc/passwd> uid.txt cut –d":" –f1 /etc/passwd> user.txt	to cut uid no & write to uid.txt to cut user name & write to user.txt
30	paste	to paste files	paste –d= user.txt uid.txt > new.txt	combined both files & write new.txt with separator

2.4 I/O Redirection

By using some special notation we can redirect the output of many commands to files, devices, and even to the input of other commands.

2.4.1 Standard Output

Most command line programs that display their results do so by sending their results to a facility calledstandard output. By default, standard output directs its contents to the display. To redirect standard output to a file, the ">" character is used like this:

```
ls> file_list.txt
```

In this example, the ls command is executed and the results are written in a file named file_list.txt. Since the output of ls was redirected to the file, no results appear on the display.

Each time the command above is repeated, file_list.txt is overwritten (from the beginning) with the output of the command ls. If you want the new results to be appended to the file instead, use ">>" like this:

```
ls>> file_list.txt
```

When the results are appended, the new results are added to the end of the file, thus making the file longer each time the command is repeated. If the file does not exist when you attempt to append the redirected output, the file will be created.

```
cat > foo.txt
```

Writing/creating into file using Unix command cat
cat > foo.txt. Enter a few more lines of text, and use the [Ctrl]-[D] key combination to quit cat.
Its redirect written lines of text into the file foo.txt.
To read file use cat foo.txt

Using Multiple Commands

touch foo.txt; echo “test file” >> foo.txt; cat foo.txt
touch : create new empty file
echo : print given text
>> redirect and append output in foo.txt

2.4.2 Standard Input

Many commands can accept input from a facility called standard input. By default, standard input gets its contents from the keyboard, but like standard output, it can be redirected. To redirect standard input from a file instead of the keyboard, the "<" character is used like this:

```
sort< file_list.txt
```

In the above example we used the sort command to process the contents of file_list.txt. The results are output on the display since the standard output is not redirected in this example. We could redirect standard output to another file like this:

```
sort< file_list.txt > sorted_file_list.txt
```

As you can see, a command can have both its input and output redirected. Be aware that the order of the redirection does not matter. The only requirement is that the redirection operators (the "<" and ">") must appear after the other options and arguments in the command.

2.4.3 Pipes

By far, the most useful and powerful thing you can do with I/O redirection is to connect multiple commands together with what are called pipes. With pipes, the standard output of one command is fed into the standard input of another. Here is my absolute favorite:

```
ls -l | less
```

In this example, the output of the ls command is fed into less. By using this "| less" trick, you can make any command have scrolling output. I use this technique all the time.

By connecting commands together, you can accomplish amazing feats. Here are some examples you'll want to try:

Examples of commands used together with pipes

Command	What it does
ls -lt | head	Displays the 10 newest files in the current directory.
du | sort -nr	Displays a list of directories and how much space they consume, sorted from the largest to the smallest.
find . -type f -print | wc -l	Displays the total number of files in the current working directory and all of its subdirectories.

```
who | wx -l
who | wc -l
ls | wc -l
ls | more
cat *.c | wc –c
echo "2+3" | bc
```

2.5 Advance LINUX Commands

2.5.1 Finding/Filter Files

which

The 'which' is a useful command for finding the full path of the executable program that would be executed if the name of the executable program is entered on the command line. The command:

```
which startx
```

locate

Locate command is use to find the locations of files and directories on Linux, to use the locate command; you will need to have aupdateddatabase on your system. (use'updated' command)

```
locate whereis
```

whereis

This command will locate binary (or executable) programs and their respective man pages. The command:

```
whereis mysql
```

will find all binaries and manpages with the name mysql

grep

grep searches the named input *FILE*s (or standard input if no files are named, or if a single hyphen-minus (**-**) is given as file name) for lines containing a match to the given *PATTERN*. By default, **grep** prints the matching lines.

- **Find word (string)from file**
  grep<word><file>
- **Find all files which contain a word**
  grep -l <word> *
  grep "this" demo_*
- **Find abstract pattern: ab 2 digits cd**
  grep 'ab[0-9][0-9]cd' <file>
- **Match regular expression in files**
  grep "lines.*empty" demo_file
  o/p => Two lines above this line is empty.
- **Highlighting the search using GREP_OPTIONS**
  As grep prints out lines from the file by the pattern / string you had given, if you wanted it to highlight which part matches the line, then you needs to follow the following way.
  Run belowcommand first:
  ```
  $ export GREP_OPTIONS='--color=auto' GREP_COLOR='100;8'
  $ grep this demo_file
  ```
  o/p
  this line is the 1st lower case line in this file.
  Two lines above **this** line is empty.
  And **this** is the last line.
- **Searching in all files recursively using grep–r**
  When you want to search in all the files under the current directory and its sub directory. -r option is the one which you need to use. The following example will look for the string “bhavesh” in all the files in the current directory and all its subdirectory.
  ```
  $ grep -r "bhavesh" *
  ```
- **Invert match using grep –v**
  option -v to do invert match. When you want to display the lines which do not matches the given string/pattern, use the option -v as shown below. This example will display all the lines that did not match the word “go”.
  ```
  $ grep -v "go" demo_text
  ```

find

To find files and dir

```
$ find / -name filename                    --   Finds the file filename in
$ find  /root -name  demo_text
$ find -name 'mypage.htm'
```

In the above command the system would search for any file named mypage.htm in the current directory and any subdirectory.

```
find / -name 'mypage.htm'
```

In the above example the system would search for any file named mypage.htm on the root and all subdirectories from the root.

```
find -name 'file*'
```

In the above example the system would search for any file beginning with file in the current directory and any subdirectory.

```
find -name '*' -size +1000k
```

In the above example the system would search for any file that is larger then 1000k.

```
find . -size +500000 -print
```

Next, similar to the above example, just formatted differently this command would find anything above 500MB.

awk

Short for Aho, Weinberger, and Kernighan, awk is one of the most powerful tools in UNIX used for processing the rows and columns in a file. Awk has built in string functions and associative arrays. Awk supports most of the operators, conditional blocks, and loops available in C language.

One of the good things is that you can convert Awk scripts into Perl scripts using a2p utility.

The basic syntax of AWK:

```
awk 'BEGIN {start_action} {action} END {stop_action}' filename
```

Here the actions in the begin block are performed before processing the file and the actions in the end block are performed after processing the file. The rest of the actions are performed while processing the file.

Examples:

Create a file input_file with the following data. This file can be easily created using the output of ls -l

```
-rw-r--r-- 1 center center  0 Dec  8 21:39 p1
-rw-r--r-- 1 center center 17 Dec  8 21:15 t1
-rw-r--r-- 1 center center 26 Dec  8 21:38 t2
-rw-r--r-- 1 center center 25 Dec  8 21:38 t3
-rw-r--r-- 1 center center 43 Dec  8 21:39 t4
-rw-r--r-- 1 center center 48 Dec  8 21:39 t5
```

From the data, you can observe that this file has rows and columns. The rows are separated by a new line character and the columns are separated by a space characters. We will use this file as the input for the examples discussed here.

1. `awk '{print $1}' input_file`

Here $1 has a meaning. $1, $2, $3... represents the first, second, third columns... in a row respectively. This
awk command will print the first column in each row as shown below.

```
-rw-r--r--
-rw-r--r--
-rw-r--r--
-rw-r--r--
```

```
-rw-r--r--
-rw-r--r--
```

To print the 4th and 6th columns in a file use awk '{print $4,$5}' input_file

Here the Begin and End blocks are not used in awk. So, the print command will be executed for each row it reads from the file.

2. awk 'BEGIN {sum=0} {sum=sum+$5} END {print sum}' input_file

This will prints the sum of the value in the 5th column. In the Begin block the variable sum is assigned with value 0. In the next block the value of 5th column is added to the sum variable. This addition of the 5th column to the sum variable repeats for every row it processed. When all the rows are processed the sum variable will hold the sum of the values in the 5th column. This value is printed in the End block.

3. In this example we will see how to execute the awk script written in a file. Create a file sum_column and paste the below script in that file

```
#!/usr/bin/awk -f
BEGIN {sum=0}
{sum=sum+$5}
END {print sum}
```

Now execute the the script using awk command as

awk -f sum_column input_file.

This will run the script in sum_column file and displays the sum of the 5th column in the input_file.

4. awk '{ if($9 == "t4") print $0;}' input_file

This awk command checks for the string "t4" in the 9th column and if it finds a match then it will print the entire line. The output of this awk command is

```
-rw-r--r-- 1 pcenter pcenter 43 Dec  8 21:39 t4
```

5. awk 'BEGIN { for(i=1;i<=5;i++) print "square of", i, "is",i*i; }'

This will print the squares of first numbers from 1 to 5. The output of the command is

```
square of 1 is 1
square of 2 is 4
square of 3 is 9
square of 4 is 16
square of 5 is 25
```

Notice that the syntax of “if” and “for” are similar to the C language.

Awk Built in Variables:

You have already seen $0, $1, $2... which prints the entire line, first column, second column... respectively. Now we will see other built in variables with examples.

FS - Input field separator variable:
So far, we have seen the fields separated by a space character. By default Awk assumes that fields in a file are separted by space characters. If the fields in the file are separated by any other character, we can use the FS variable to tell about the delimiter.

6. awk 'BEGIN {FS=":"} {print $2}' input_file
OR
awk -F: '{print $2} input_file

This will print the result as

```
39 p1
15 t1
38 t2
38 t3
39 t4
39 t5
```

OFS - Output field separator variable:

By default whenever we printed the fields using the print statement the fields are displayed with space character as delimiter. For example

7. awk '{print $4,$5}' input_file
The output of this command will be

```
center 0
center 17
center 26
center 25
center 43
center 48
```

We can change this default behavior using the OFS variable as
awk 'BEGIN {OFS=":"} {print $4,$5}' input_file

```
center:0
center:17
center:26
center:25
center:43
center:48
```

Note: print $4,$5 and print $4$5 will not work the same way. The first one displays the output with space as delimiter. The second one displays the output without any delimiter.
NF - Number of fileds variable:

The NF can be used to know the number of fields in line

8. awk '{print NF}' input_file
This will display the number of columns in each row.

NR - number of records variable:
The NR can be used to know the line number or count of lines in a file.

9. awk '{print NR}' input_file
This will display the line numbers from 1.

10. awk 'END {print NR}' input_file
This will display the total number of lines in the file.

diff
Displays two files and prints the lines that are different.
diff<file1><file2>

cmp
Compares two files and tells you which line numbers are different.
cmp file1.txt file2.txt

comm
Select or reject lines common to two files.
comm myfile1.txt myfile2.txt

2.5.2 Info about UNIX System

Description	Example(s)
See who is logged on	who
Get the date	date, date +%m, date +%h , date +%s date +"%h %m"
See who logged in lately	last -20
See what operating system is thei	uname-auname -r, uname -n
See who you are	whoami
Get the name of your computer	hostname
See the disk space used	df -k
See how much space all your file	du -k

2.5.2 Compressing and Archiving Files

It is useful to store a group of files in one file for easy backup, for transfer to another directory, or for transfer to another computer. It is also useful to compress large files; compressed files take up less disk space and download faster via the Internet.

Both Linux and UNIX include various commands for Compressing and decompresses (read as expand compressed file). To compress files you can use gzip, bzip2 and zip commands. To expand compressed file (decompresses) you can use and gzip -d, bunzip2 (bzip2 -d), unzip commands.

Compressing files

Syntax	Description	Example(s)
gzip {filename}	Gzip compress the size of the given files using Lempel-Ziv coding (LZ77). Whenever possible, each file is replaced by one with the extension .gz.	gzip mydata.doc gzip *.jpg ls -l
bzip2 {filename}	bzip2 compresses files using the Burrows-Wheeler block sorting text compression algorithm, and Huffman coding. Compression is generally considerably better than that achieved by bzip command (LZ77/LZ78-based compressors). Whenever possible, each file is replaced by one with the extension .bz2.	bzip2 mydata.doc bzip2 *.jpg ls -l
zip {.zip-filename} {filename-to-compress}	zip is a compression and file packaging utility for Unix/Linux. Each file is stored in single .zip {.zip-filename} file with the extension .zip.	zip mydata.zip mydata.doc zip data.zip *.doc ls -l

tar -zcvf {.tgz-file} {files}
tar -jcvf {.tbz2-file} {files}
	The GNU tar is archiving utility but it can be use to compressing large file(s). GNU tar supports both archive compressing through gzip and bzip2. If you have more than 2 files then it is recommended to use tar instead of gzip or bzip2.
	-z: use gzip compress
	-j: use bzip2 compress
		tar -zcvf data.tgz *.doc
		tar -zcvf pics.tar.gz *.jpg *.png
		tar -jcvf data.tbz2 *.doc
		ls -l

Decompressing files

Syntax	Description	Example(s)

gzip -d {.gz file}
gunzip {.gz file}
	Decompressed a file that is created using gzip command. File is restored to their original form using this command.	gzip -d mydata.doc.gz
		gunzip mydata.doc.gz

bzip2 -d {.bz2-file}
bunzip2 {.bz2-file}
	Decompressed a file that is created using bzip2command. File is restored to their original form using this command.	bzip2 -d mydata.doc.bz2
		gunzip mydata.doc.bz2

unzip {.zip file}	Extract compressed files in a ZIP archive.	unzip file.zip unzip data.zip resume.doc

tar -zxvf {.tgz-file}
tar -jxvf {.tbz2-file}
	Untar or decompressed a file(s) that is created using tar compressing through gzip and bzip2filter	tar -zxvf data.tgz
		tar -zxvf pics.tar.gz *.jpg
		tar -jxvf data.tbz2

2.5.4 Mail

Description	Example(s)
Check for mail	mailx
Read mail	mail
Compose mail	Mail -s <subject><mailaddress> or Mailx kbhavesh Subject : Hello This is test message [ctrl + d] EOT
Mail a whole file (one "<" is real)	Mail -s <subject><mailaddr><<file> mailx -s hello kbhavesh<output.txt

2.5.6 Other Commands

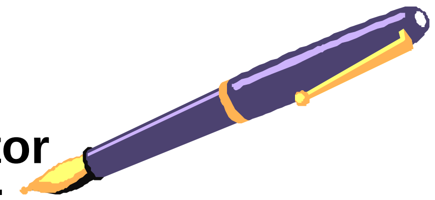
Description	Example(s)
Viewing Processes	ps ps –a, ps –f, ps –u user, ps –e
Directing output to file	ls> list
Counting Number of Line in File	wc list, wc –l, wc -w, wc -c ls | wc
Chat via terminal	talk kbhavesh
make typescript of terminal session(save session)	Script my.log [for Write in file give exit]
Calculator	bc
Change password	passwd username
Who are users	who
To know your Machine's characteristics	uname, uname -r, uname -n
	script my.log [for Write in file give exit]
Terminate process	kill -9 3522 kill -s KILL 3522
Hard link	lnemp.list employee
Soft link	ln –s emp.list employee
Read data from Terminal	read a echo $a
Run job in Background	nohup sort out.txt (note: Sending output in nohup.out)
Evaluate arguments as expressions and print the results.	expr 5 + 10 / 2 i=`expr&dollar;i + 1` echo $i

exec - Invoke subprocess

The exec command replaces the current shell process with the specified command. Normally, when you run a command a new process is spawned (forked). The exec command does not spawn a new process. Instead, the current process is overlaid with the new command. In other words the exec command is executed in place of the current shell without creating a new process.

1. exec ls
2. exec ls | cat
3. { exec ls; echo this too; } | cat
4. #!/bin/bash
   exec ls | cat
   echo “End of File”
   exit 0
5.
```
while true
do
echo "Go to Show files[mM],
Editor[eE],
or eXit[xX]:"
read ANS
case "$ANS" in
    [mM]) exec ls;;
    [eE]) exec ${EDITOR:=vi} ;;
    [xX]) exec exit ;;
esac
done
```

3. Working with Editors - The vim Editor



If you're working in command line mode, you may want to become familiar with at least one text editor that operates in the Linux console and to crate script file. The vi editor is the original editor used on Unix systems.

It is text editor software that helps you create and modify text files. There are a lot of text editors, but vi is the best of them. If most of your time is devoted to text editing, perhaps you should consider an editor of power, which is vi/vim available. It uses the console graphics mode to emulate a text-editing window, allowing you to visually see the lines of your file, move around within the file, and insert, edit, and replace text.

What is good in vim as an editor?

- ➢ vim is efficient and elegant. It does things quickly by the best way.
- ➢ vim is healthful. It doesn't make your fingers and arms painful.
- ➢ vim is small and fast. It fits in one or two floppies and starts instantly.
- ➢ vim is powerful. No feature useful for text editing is missing.
- ➢ vim is simple and clean. It doesn't offer to do things irrelevant to text editing.
- ➢ vim is running everywhere. If you're willing to use it, you can, no matter what computers and operating systems you use.
- ➢ vim is free.

Working with vim/vi

If you type vi on your shell prompt it will run vim as alias is defined in operating system like alias vi='vim' or
you can start by giving command vim. To start vim and edit a file (new or existing)

$vi filename

For Example $vi hello.sh

You can type the stuff in a file and when you completed, you can save it as a file.

❖ To Exit vim

save your file before you exit: press ESC, then type**: wq**
save your file as *newname*before you exit: press ESC, then type **:wq*newname***
Exit without saving, press ESC, then type **: q**
Forced quit_If**:q**doesn't work, it's probably because you didn't save the change. If you want to save, use **:wq**.
If you don't want to save the changes, type **: q!**
Forced save _if :wq doesn't work and if you want to save it forcefully, type **: wq!**

❖ Enter TEXT in to file
As soon as you open a file using vi it starts in command mode so you cannot insert text. To begin typing new text into the file, you need to switch to **insert mode** by press **i**. Then you can see each character you type appears on the screen.
1. From command mode to insert mode, press i

2. to come back after typing text from **insert mode** to **command mode**, press **ESC**
3. Best practice is to use **A (SHIFT+a)** which will take you to insert mode at the end of line.

❖ Delete Text

First I will tell you that the DELETE key always works, and the BACKSPACE key works with the newly typed text in the **insert mode**. Other ways to delete a character is

- ➢ Make sure you are in **command mode** by press **ESC**
- ➢ move the cursor to the character you want to delete
- ➢ press **x** , this character disappears
- ➢ **Delete the whole current line – dd:**You must learn this command. It makes you feel power. If you have several lines of text to delete, you can keep hitting **dd**and watch the lines disappearing. If you overdid it and removed some useful stuff, don't panic—keep pressing **u** the removed lines will reappear in reverse order. **u**means "undo". **vim**has multiple levels of undo.

xis just one of the many powerful deletion commands. Usually you need to move the cursor to the character you want to delete and then press **x** to remove it. Remember using the cursor motion commands **j k h l** to locate your target, and don't leave the **command mode**.

❖ COPY, CUT and PASTE

- ➢ Enter the **command mode** by pressing **ESC**
- ➢ Move the cursor to the line which you want to make a copy, by pressing j or k and press **yy //it will copy that line**
- ➢ to make a copy of the line, or press **dd**to cut it and make a copy
- ➢ now move cursor (by pressing k or j) to the the location where you want to put this copy and press **p**
- ➢ If you want to copy or cut several lines, put a number before the **yy**or **dd**command, like **3yy // to copy 3 lines.**

Important Keys for vi Editor:

<u>Input mode command</u>

i: Insert the text to the left of Cursor.
a: append the text to the right of Cursor.
I: Insert the text to the left of the line.
A: Insert the text to the right of the line.
o: Open the line below.
O: Open line above.
r: Replaces the character under.
R: Replaces the text to the right.
s: Replaces the character under the cursor with any no of characters.
S: Replaces entire line.

<u>Ex mode command</u>

:w – saving your work.
:x and :wq save and quit
:q! – abort
:w new.sh same as save as.
:w! new.sh to overwrite the existing file.

:$w copy the last line to the file
:sh to move to the shell from the program.

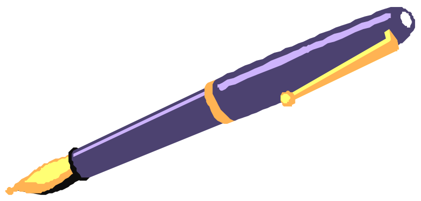
Navigation.

k: moves cursor up.
j: down.
h: left.
l: right.

Word Navigation.

b: moves back to the beginning of the word.
w: moves forward to the beginning of the word.
e: moves forward to the end of the word.

4. Shell Programming



4.1 Writing your first bash shell script

Let us write a shell program to print knowledge is power on screen.
Type the following command to open file:

```
   $ vi hello.sh
```

The first line should be as follows:

```
   #!/bin/bash   or   #!/bin/sh
```

It is called a shebang. It consists of a number sign and an exclamation point character (#!), followed by the full path to the interpreter such as /bin/bash. All scripts under UNIX execute using the interpreter specified on a first line.

Next append code as follows:

```
   # A shell script to print message
   echo "Knowledge is power"
```

Save and close the file (Esc + : + wq). At the end your script should look like as follows:

```
   #!/bin/bash
   # A shell script to print message
   # ------------------------------
   echo "Knowledge is power"
```

Set executable permission

Type the following command to set executable permission:

```
   chmod +x hello.sh or chmod 777 hello.sh
```

Run your bash shell script

Type the following command:

```
   $ ./hello.sh or sh hello.sh
```

Sample output: `Knowledge is power`

4.2 Passing arguments to a shell script

Any shell script you run has access to (inherit) the environment variables accessible to its parent shell. In addition, any arguments you type after the script name on the shell command line are passed to the script as a series of variables.

The following parameters are recognized:

$*

Returns a single string (``**$1**, **$2** ... **$n**'') comprising all of the positional parameters separated by the internal field separator character (defined by the **IFS** environment variable).

$@

Returns a sequence of strings (``**$1**'', ``**$2**'', ... ``**$n**'') wherein each positional parameter remains separate from the others.

$1, **$2** ... **$*n***

Refers to a numbered argument to the script, where ***n*** is the position of the argument on the command line. In the Korn shell you can refer directly to arguments where ***n*** is greater than 9 using braces. For example, to refer to the 57th positional parameter, use the notation **${57}**. In the other shells, to refer to parameters with numbers greater than 9, use the **shift** command; this shifts the parameter list to the left. **$1** is lost, while **$2** becomes **$1**, **$3** becomes **$2**, and so on. The inaccessible tenth parameter becomes **$9** and can then be referred to.

Examples:

```
echo There are $# arguments to $0: $*
echo first argument: $1
echo second argument: $2
echo third argument: $3
echo here they are again: $@
```

When the file is executed, you will see something like the following:

```
$ mytest foo bar bam
There are 3 arguments to mytest: foo bar bam
first argument: foo
second argument: bar
third argument: bam
here they are again: foo bar bam
```

4.3 Variables

Variables are "words" that hold a *value*. The shell enables you to create, assign, and delete variables. Although the shell manages some variables, it is mostly up to the programmer to manage variables in shell scripts

Variables are defined as follows:
`name=value`
The shell enables you to store any value you want in a variable. For example,
`FRUIT=apple`

Accessing Values
To access the value stored in a variable, prefix its name with the dollar sign ($). For example, the command
`echo $FRUIT`

4.4 Array Variables

Arrays provide a method of grouping a set of variables. Instead of creating a new name for each variable that is required, you can use a single array variable that stores all the other variables.

The simplest method of creating an array variable is to assign a value to one of its indices. This is expressed
as follows:
`name[index]=value`
Here *name* is the name of the array, *index* is the index of the item in the array that you want

to set, and *value* is the value you want to set for that item.

As an example, the following commands

```
#!/bin/sh
NAME[0]="mscit"
NAME[1]="mca"
NAME[2]="bscit"
NAME[3]="bca"

echo "First Method: ${NAME[1]}"
echo "First Method: ${NAME[*]}"
echo "Second Method: ${NAME[@]}"
```

Various array operations

```
#!/bin/bash
# array-ops.sh: More fun with arrays.
array=( zero one two three four five )
# Element 0   1   2    3     4    5

echo ${array[0]}       #  zero
echo ${array:0}        #  zero
                       #  Parameter expansion of first element,
                       #+ starting at position # 0 (1st character).
echo ${array:1}        #  ero
                       #  Parameter expansion of first element,
                       #+ starting at position # 1 (2nd character).

echo "--------------"

echo ${#array[0]}      #  4
                       #  Length of first element of array.
echo ${#array}         #  4
                       #  Length of first element of array.
                       #  (Alternate notation)

echo ${#array[1]}      #  3
                       #  Length of second element of array.
                       #  Arrays in Bash have zero-based indexing.

echo ${#array[*]}      #  6
                       #  Number of elements in array.
echo ${#array[@]}      #  6
                       #  Number of elements in array.

echo "--------------"

array2=([0]="first element" [1]="second element" [3]="fourth element" )
#            ^     ^       ^     ^      ^       ^     ^      ^       ^
# Quoting permits embedding whitespace within individual array elements.

echo ${array2[0]}      # first element
echo ${array2[1]}      # second element
echo ${array2[2]}      # Skipped in initialization, and therefore null.
echo ${array2[3]}       # fourth element
echo ${#array2[0]}      # 13    (length of first element)
echo ${#array2[*]}      # 3     (number of elements in array)
exit
```

4.5 If/Else Statement

In order for a script to be very useful, you will need to be able to test the conditions of variables. Most programming and scripting languages have some sort of if/else expression and so does the bourne shell. Unlike most other languages, spaces are very important when using an if statement. Let's do a simple script that will ask a user for a password before allowing him to continue. This is obviously not how you would implement such security in a real system, but it will make a good example of using if and else statements.

```
#!/bin/bash

VALID_PASSWORD="secret"

echo "Please enter the password:"
read PASSWORD

if [ "$PASSWORD" == "$VALID_PASSWORD" ]; then
       echo "You have access!"
else
       echo "ACCESS DENIED!"
fi
```

Remember that the spacing is very important in the if statement. Notice that the termination of the if statement is fi. You will need to use the fi statement to terminate an if whether or not use use an else as well. You can also replace the "==" with "!=" to test if the variables are NOT equal. There are other tokens that you can put in place of the "==" for other types of tests. The following table shows the different expressions allowed.

4.6 Elif Statement

```
#!/bin/sh
# Prompt for a user name...
echo "Please enter your age:"
read AGE

if [ "$AGE" -lt 20 ] || [ "$AGE" -ge 50 ]; then
       echo "Sorry, you are out of the age range."
elif [ "$AGE" -ge 20 ] && [ "$AGE" -lt 30 ]; then
       echo "You are in your 20s"
elif [ "$AGE" -ge 30 ] && [ "$AGE" -lt 40 ]; then
       echo "You are in your 30s"
elif [ "$AGE" -ge 40 ] && [ "$AGE" -lt 50 ]; then
       echo "You are in your 40s"
fi
```

<u>OutPut</u>

```
[root@localhost ~]# sh mpro.sh
Please enter your age:
25
You are in your 20s
```

```
[root@localhost ~]# sh mpro.sh
Please enter your age:
31
You are in your 30s
[root@localhost ~]# sh mpro.sh
Please enter your age:
15
Sorry, you are out of the age range.
```

4.7 TEST or [command with if statement

Table File Test Options for the `test` Command

Option	Description
`-b file`	True if `file` exists and is a block special file.
-c file	True if file exists and is a character special file.
-d file	True if file exists and is a directory.
-e file	True if file exists.
-f file	True if file exists and is a regular file.
-g file	True if file exists and has its SGID bit set.
-h file	True if file exists and is a symbolic link.
-k file	True if file exists and has its "sticky" bit set.
-p file	True if file exists and is a named pipe.
-r file	True if file exists and is readable.
-s file	True if file exists and has a size greater than zero.
-u file	True if file exists and has its SUID bit set.
-w file	True if file exists and is writable.
-x file	True if file exists and is executable.
`-O file`	True if `file` exists and is owned by the effective user ID.

```
if test -f abc.c
then
echo "File exist"
fi

if [ -f abc.c ]
then
echo "File exist"
fi

if test -f /bin/bash
then
echo "File /bin/bash exist"
fi

if [ -d /bin/bash ]
then
echo "/bin/bash is directory"
else
echo "/bin/bash is NOT Directory"
fi
if [ -z "$FRUIT_BASKET" ] ; then
echo "Your fruit basket is empty" ;
else
```

```
echo "Your fruit basket has the following fruit: $FRUIT_BASKET"
fi
```

OR

```
read FRUIT_BASKET
if [ -z "$FRUIT_BASKET" ] ; then
echo "Your fruit basket is empty" ;
else
echo "Your fruit basket has the following fruit: $FRUIT_BASKET"
fi
```

OR

```
if [ -n "$FRUIT_BASKET" ] ; then
echo "Your fruit basket has the following fruit: $FRUIT_BASKET"
else
echo "Your fruit basket is empty" ;
fi
```

The test Numeric Comparisons

Comparison	Description
n1 –eq *n2*	Check if *n1* is equal to *n2*.
n1 –ge *n2*	Check if *n1* is greater than or equal to *n2*.
n1 –gt *n2*	Check if *n1* is greater than *n2*.
n1 -le *n2*	Check if *n1* is less than or equal to *n2*.
n1 –lt *n2*	Check if *n1* is less than *n2*.
n1 -ne *n2*	Check if *n1* is not equal to *n2*.

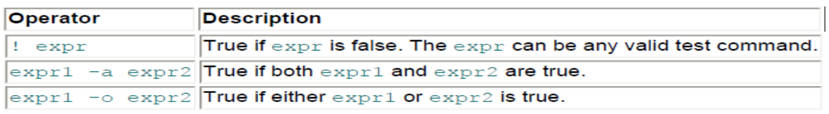
The test Command String Comparisons

Comparison	Description
str1 = *str2*	Check if *str1* is the same as string *str2*.
str1 != *str2*	Check if *str1* is not the same as *str2*.
str1 < *str2*	Check if *str1* is less than *str2*.
str1 > *str2*	Check if *str1* is greater than *str2*.
-n *str1*	Check if *str1* has a length greater than zero.
-z *str1*	Check if *str1* has a length of zero.

The basic syntax for checking whether two strings are equal is

```
test string1 = string2
or
[ string1 = string2 ]

test int1 operator int2
or
[ int1 operator int2 ]
```

The test Command with *expr*



Operator	Description
`! expr`	True if `expr` is false. The `expr` can be any valid test command.
`expr1 -a expr2`	True if both `expr1` and `expr2` are true.
`expr1 -o expr2`	True if either `expr1` or `expr2` is true.

```
test expr1 operator expr2
or
[ expr1 operator expr2 ]
```

4.8 For Loop

The for statement is used when you want to loop through a list of items. The body of the loop is put between do and done. Let's say that we want to write a program that will validate numbers in a given list. These numbers can be loaded from a file, hard coded, or manually entered by the user. For our example, we will ask the user for a list of numbers separated with spaces. We will validate each number and make sure that it is between 1 and 100. The best way to write a program like this would be to use a for loop.

```
#!/bin/bash

fori in  1 2 3 4 5
do
echo -n "$i"
done


#!/bin/bash

echo "Can you see the following:"

for (( i=1; i<=5; i++ ))
do
for (( j=1; j<=i;  j++ ))
do
echo -n "$i"
done
echo ""
done

#!/bin/bash
# Validate numbers...

echo "Please enter a list of numbers between 1 and 100. "
read NUMBERS

for NUM in $NUMBERS
do
       if [ "$NUM" -lt 1 ] || [ "$NUM" -gt 100 ]; then
       echo "Invalid Number ($NUM) - Must be between 1 and 100!"
       else
              echo "$NUM is valid."
       fi
done
```

4.9 While/ Do-While Loop

The while statement is used when you want to loop while a statement is true. This is the same in many other programming and scripting languages. The body of the loop is put between do and done. Suppose that we want to `write` a script to have the user guess what number we are thinking of. The best way to use this would be to use a while loop.

```
     while [ condition ]
```

```
do
  command1
  command2
  commandN
done

#!/bin/bash
c=1
while [ $c -le 5 ]
do
  echo "Welcome $c times"
  (( c++ ))
done
```

```
#!/bin/bash
while read f
do
   case $f in
         hello)          echo English    ;;
         howdy)          echo American   ;;
         gday)           echo Australian ;;
         bonjour)        echo French     ;;
         "guten tag")    echo German     ;;
         *)              echo Unknown Language: $f
                 ;;
    esac
done
```

4.10 Until Loop

```
until who | grep "$1" > /dev/null
do
echo "Waiting to Login…"
sleep 60
done
echo -e '\a'
echo "Logged Now"
```

```
x=1
until [ $x -ge 10 ]
do
echo $x
x=`expr $x + 1`
done
```

4.11 break statement

```
#!/bin/sh

a=0

while [ $a -lt 10 ]
do
echo $a
if [ $a -eq 5 ]
then
break
fi
   a=`expr $a + 1`
done
```

4.12 Continue statement

```
#!/bin/sh

NUMS="1 2 3 4 5 6 7"

for NUM in $NUMS
do
   Q=`expr $NUM % 2`
if [ $Q -eq 0 ]
then
echo "Number is an even number!!"
continue
fi
echo "Found odd number"
done
```

4.13 Function

```
#!/bin/sh
a=0
stop1()
{
echo "stop";    exit;
}

while [ $a -lt 10 ]
do
echo $a
if [ $a -eq 5 ]
then
      stop1;
fi
   a=`expr $a + 1`
done
```

4.14 Case Statement

Many programming languages and scripting languages have the concept of a case or select statement. This is generally used as a shortcut for writing if/else statements. The case statement is always preferred when there are many items to select from instead of using a large if/elif/else statement. It is usually used to implement menus in a script. The case statement is terminated with esac (case backwards). Here is a simple example of using a case statement:

```
#!/bin/sh

echo "Enter a number between 1 and 10. "
read NUM

case $NUM in
  1) echo "one" ;;
  2) echo "two" ;;
  3) echo "three" ;;
  4) echo "four" ;;
  5) echo "five" ;;
  6) echo "six" ;;
  7) echo "seven" ;;
  8) echo "eight" ;;
  9) echo "nine" ;;
  10) echo "ten" ;;
  *) echo "INVALID NUMBER!" ;;
esac
```

exit

Exit status when our scripts finish. To do this, use the exit command. The exit command causes the script to terminate immediately and set the exit status to whatever value is given as an argument. For example:

```
exit 0
```

Exits your script and sets the exit status to 0 (success), whereas

```
exit 1
```

Exits your script and sets the exit status to 1 (failure).

```
if [ x -ne 0 ]
then
echo "x value  failed"
exit 1
fi
echo "x value ucceeded"
exit 0
```

expr

```
a=`expr $1 + $1`
    b=`expr $a - 1`
echo $b
```

return

```
#!/bin/bash
functionX () {
if [ $1 = "yes" ]
then
   return 1
else
   return 2
fi
}

functionX yes
echo functionX returned $?
functionX no

echofunctionX returned $?

Output

./myscript.sh
functionX returned: 1
functionX returned: 2
```

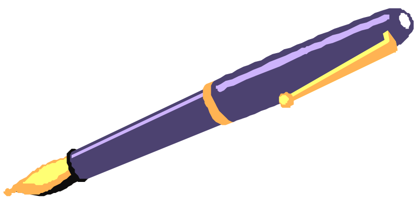
Shift

```
while [ "$2" != "" ];
do
echo "$2"
shift
done
echo "$1"
exit 0
```

trap

```
function settrap {
trap "echo 'You hit control-C'" INT
}

settrap
while true; do
sleep 20
done
```

5. LINUX Booting Sequence & Services



5.1 Booting Sequence

BIOS Initialization:
- POST(Power on Self-Test)
  BIOS will run POST which will detect all the devices and peripherals connected to the server.
- Boot Device Selected
  At the end of the POST, a boot device is selected from the list of detected boot

Boot Loader Components:
- Boot Loader (GRUB / LILO)
  BIOS passes control to an initial program loader (IPL = GRUB / LILO) installed on Master Boot
  Record of the Hard drive. The main job of the OS Loader is to locate the kernel on the disk, load it

Kernel Startup:
- When the kernel is loaded, it initializes the devices (via their drivers), starts the swapper (it is a "kernel process", called kswapd) and mounts the root file system (/).
- Only then the kernel creates the first process which is numbered 1 and known as

INIT and INITTAB:
- When init starts it reads */etc/inittab* for further instructions. This file defines what should be run in different *run-levels*.
- Then it processes /etc/rc.d/rc.sysinit which first of all mount all file system as per /etc/fstab file and further processes all scripts located in /etc/rc.d/rc(runlevel as per inittab file).d/
- After Running all the scripts of /etc/rc.d/rc (runlevel no.).d/ it successfully boots to the runlevel defined as per /etc/inittab file and will ask for login.

5.2 System Services

A Linux service is an application (or set of applications) that runs in the background waiting to be used, or carrying out essential tasks. I've already mentioned a couple of typical ones (Apache and MySQL). You will generally be unaware of services until you need them.

Let's start by looking at how the system is set up, and in particular at the directory /etc/rc.d. Here you will find either a set of files named rc.0, rc.1, rc.2, rc.3, rc.4, rc.5, and rc.6, or a set of directories named rc0.d, rc1.d, rc2.d, rc3.d, rc4.d, rc5.d, and rc6.d. You will also find a file named **/etc/inittab**. The system uses these files (and/or directories) to control the services to be started.

If you look in the file /etc/inittab you will see something like:

```
id:5:initdefault:

.
.
.
# Run xdm in runlevel 5

x:5:respawn:/etc/X11/prefdm –nodaemon
```

The boot process uses these parameters to identify the default runlevel and the files that will be used by that runlevel. In this example, runlevel 5 is the default and the scripts that define runlevel 5 can be found in /etc/rc.d/rc.5

And what is a runlevel? You might assume that this refers to different levels that the system goes through during a boot up. Instead, think of the runlevel as the point at which the system is entered. Runlevel 1 is the most basic configuration (simple single user access using an text interface), while runlevel 5 is the most advanced (multi-user, networking, and a GUI front end). Runlevels 0 and 6 are used for halting and rebooting the system.

5.3 RUNLEVEL

A RUN LEVEL is a system state which allows only selected services to be executed as a special group / user. The processes spawned by init command/process for each of these run levels are defined in /etc/inittab file. Run levels 0, 1, and 6 are reserved / common amongst all the flavors of LINUX. Run level 0 is used to halt / shutdown / power down the system, run level 6 is used to reboot / restart the system, and run level 1 is used to get the system into single user / Administration mode. in order to find out your system's current run level, you need to use who or run level command at your shell prompt.

```
# who –r

# runlevel
```

Below is the list of RUN Levels used with LINUX Operating System.

Run Level	Red Hat LINUX
0	Halt / Shutdown / Power down
1 / s / S	Single User / System Administrator mode
2	Multi-user without Network
3	Multi-user with Network
4	N/A / Customizable
5	GUI-GNOME, KDE, X-Windows with Multi-user & Network
6	reboot / restart to default run level

Init Level	Comments
0	Runlevel 0 is reserved for the "shutdown" phase. Entering init 0 from the shell prompt will shutdown the system and usually power off the machine.
1	Runlevel 1 is usually for very basic commands. This is the equivalent to "safe mode" used by Windows. This level is usually only used to asses repairs or maintenance to the system. This is a single-user mode and does not allow other users to login to the machine.
2	Runlevel 2 is used to start most of the machines services. However, it does not start the network file sharing service (SMB, NFS). This will allows multiple users to login to the machine.
3	Runlevel 3 is commonly used by servers. This loads all services except the X windows system. This means the system will boot to the equivalent of DOS. No GUIs (KDE, Gnome) will start. This level allows multiple users to login to the machine.
4	Runlevel 4 is usually a "custom" level. By default it will start a few more services than level 3. This level is usually only used under special circumstances.
5	Runlevel 5 is everything! This will start any GUIs, extra services for printing, and 3rd party services. Full multi-users support also. This runlevel is generally used on by workstations.
6	Runlevel 6 is reserved for "reboot" only. Be carefully when running this command. Once you have entered init 6, there is no stopping it!

# /etc/rc.d/init.d/ ==or== /etc/init.d/

# service servicename start/stop/restart/status
services mysqld start

# chkconfig - used to add service in specific / all run levels
chkconfig --level 3 httpd on/off

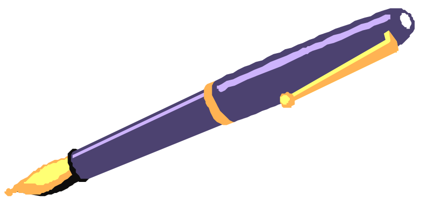
# ntsysv - simple interface for configuring run-levels

# system-config-services – GUI interface for configuring run-levels

Exercise:

(1) Boot system in run level 3
(2) Change Run Level of System to 3 by default
(3) Change Run Level of System to 5 by default
(4) Troubleshooting on Server Start up

6. Permissions



This will cover the following commands:

- chmod - modify file access rights
- su - temporarily become the superuser
- chown - change file ownership
- chgrp - change a file's group ownership

6.1 File permissions

Linux uses the same permissions scheme as Unix. Each file and directory on your system is assigned access rights for the owner of the file, the members of a group of related users, and everybody else. Rights can be assigned to read a file, to write a file, and to execute a file (i.e., run the file as a program).

To see the permission settings for a file, we can use the ls command as follows:

```
[me@localhost me]$ ls -l some_file

-rw-rw-r-- 1 me   me   1097374 Jan 26 18:48 some_file
```

We can determine a lot from examining the results of this command:

- The file "some_file" is owned by user "me"
- User "me" has the right to read and write this file
- The file is owned by the group "me"
- Members of the group "me" can also read and write this file
- Everybody else can read this file

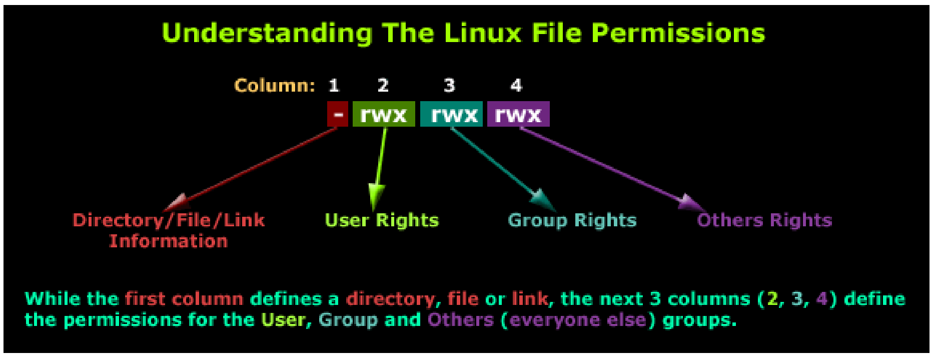
Let's try another example. We will look at the bash program which is located in the /bin directory:

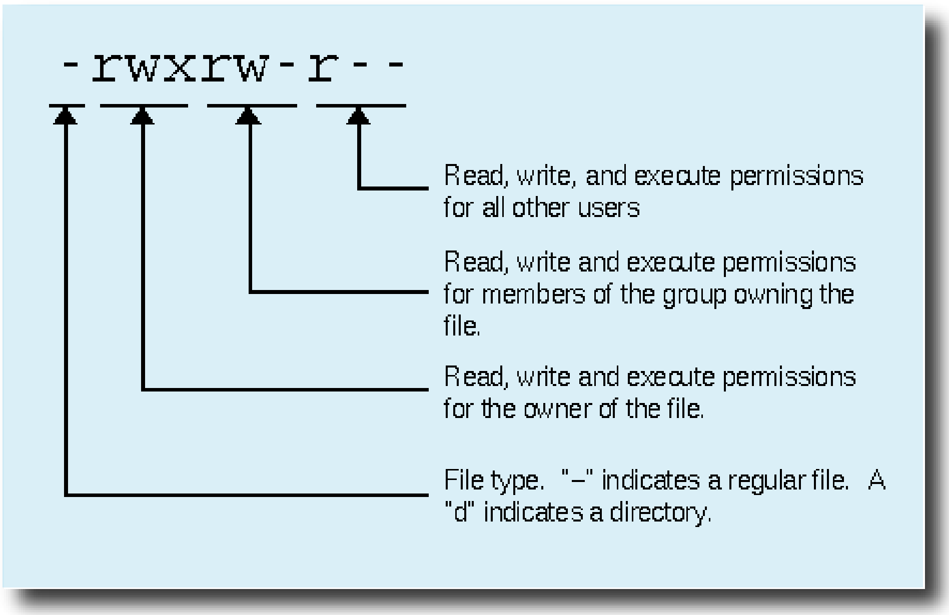
```
[me@localhost me]$ ls -l /bin/bash

-rwxr-xr-x 1 root root  316848 Jan 26 20:00 /bin/bash
```

Here we can see:

- The file "/bin/bash" is owned by user "root"
- The superuser has the right to read, write, and execute this file
- The file is owned by the group "root"
- Members of the group "root" can also read and execute this file
- Everybody else can read and execute this file

In the diagram below, we see how the first portion of the listing is interpreted. It consists of a character indicating the file type, followed by three sets of three characters that convey the reading, writing and execution permission for the owner, group, and everybody else.


Understanding The Linux File Permissions
Column: 1 2 3 4
- rwx rwx rwx
Directory/File/Link Information
User Rights
Group Rights
Others Rights
While the first column defines a directory, file or link, the next 3 columns (2, 3, 4) define the permissions for the User, Group and Others (everyone else) groups.


-rwxrw-r--
Read, write, and execute permissions for all other users
Read, write and execute permissions for members of the group owning the file.
Read, write and execute permissions for the owner of the file.
File type. "–" indicates a regular file. A "d" indicates a directory.

Permission Table

Binary rwx	Digital	Permission
000	0	No
001	1	Execute
010	2	Write
011	3	Write & Execute
100	4	Read
101	6	Read & Execute
110	6	Read & Write
111	7	Read, Write & Execute

6.2 Change the permissions - chmod

The `chmod` command is used to change the permissions of a file or directory. To use it, you specify the desired permission settings and the file or files that you wish to modify. There are two ways to specify the permissions, but I am only going to teach one way.

It is easy to think of the permission settings as a series of bits (which is how the computer thinks about them). Here's how it works:

```
rwx rwx rwx = 111 111 111
rw- rw- rw-   = 110 110 110
rwx --- ---     = 111 000 000

and so on...

rwx = 111 in binary = 7
rw- = 110 in binary = 6
r-x = 101 in binary = 5
r-- = 100 in binary = 4
```

Now, if you represent each of the three sets of permissions (owner, group, and other) as a single digit, you have a pretty convenient way of expressing the possible permissions settings. For example, if we wanted to set `some_file` to have read and write permission for the owner, but wanted to keep the file private from others, we would:

```
[me@localhost me]$ chmod 600 some_file
```

Here is a table of numbers that covers all the common settings. The ones beginning with "7" are used with programs (since they enable execution) and the rest are for other kinds of files.

Value	*Meaning*
777	*(rwxrwxrwx)* No restrictions on permissions. Anybody may do anything. Generally not a desirable setting.
755	*(rwxr-xr-x)* The file's owner may read, write, and execute the file. All others may read and execute the file. This setting is common for programs that are used by all users.
700	*(rwx------)* The file's owner may read, write, and execute the file. Nobody else has any rights. This setting is useful for programs that only the owner may use and must be kept private from others.
666	*(rw-rw-rw-)* All users may read and write the file.
644	*(rw-r--r--)* The owner may read and write a file, while all others may only read the file. A common setting for data files that everybody may read, but only the owner may change.

600	*(rw-------)* The owner may read and write a file. All others have no rights. A common setting for data files that the owner wants to keep private.

Directory permissions

The chmod command can also be used to control the access permissions for directories. In most ways, the permissions scheme for directories works the same way as they do with files. However, the execution permission is used in a different way. It provides control for access to file listing and other things. Here are some useful settings for directories:

Value	*Meaning*
777	*(rwxrwxrwx)* No restrictions on permissions. Anybody may list files, create new files in the directory and delete files in the directory. Generally not a good setting.
755	*(rwxr-xr-x)* The directory owner has full access. All others may list the directory, but cannot create files nor delete them. This setting is common for directories that you wish to share with other users.
700	*(rwx------)* The directory owner has full access. Nobody else has any rights. This setting is useful for directories that only the owner may use and must be kept private from others.

Becoming the superuser for a short while

It is often useful to become the superuser to perform important system administration tasks, but as you have been warned (and not just by me!), you should not stay logged on as the superuser. In most distributions, there is a program that can give you temporary access to the superuser's privileges. This program is called su (short for substitute user) and can be used in those cases when you need to be the superuser for a small number of tasks. To become the superuser, simply type the su command. You will be prompted for the superuser's password:

```
[me@localhost me]$ su
Password:
[root@localhost me]#
```

After executing the su command, you have a new shell session as the superuser. To exit the superuser session, type exit and you will return to your previous session.

In some distributions, most notably Ubuntu, an alternate method is used. Rather than using su, these systems employ the sudo command instead. With sudo, one or more users are granted superuser privileges on an as needed basis. To execute a command as the superuser, the desired command is simply proceeded with the sudo command. After the command is entered, the user is prompted for the user's password rather than the superuser's:

```
[me@localhost me]$ sudo some_command
Password:
[me@localhost me]$
```

```
[root@localhost me]# chmod 777 –R /home/guest/some_folder
```

Above command is to give full permission to entire directory (all folders/files and sub folders)

6.3 Changing file ownership - chown

You can change the owner of a file by using the chown command. Here's an example: Suppose I wanted to change the owner of some_file from "me" to "you". I could:

```
[me@localhost me]$ su
Password:
[root@localhost me]# chown you some_file
[root@localhost me]# chown kbhavesh:kbhavesh some_file/some_folder
[root@localhost me]# exit
[me@localhost me]$
```

Notice that in order to change the owner of a file, you must be the superuser. To do this, our example employed the su command, then we executed chown, and finally we typed exit to return to our previous session.

chown works the same way on directories as it does on files.

6.4 Changing group ownership

The group ownership of a file or directory may be changed with chgrp. This command is used like this:

```
[me@localhost me]$ chgrp new_group some_file
```

In the example above, we changed the group ownership of some_file from its previous group to "new_group". You must be the owner of the file or directory to perform a chgrp.

To change the group name of a file or directory

You can change the group of a file as shown below,

```
$ ls -l
-rw-r--r-- 1 kbhavesh kbhavesh 210 2012-01-04 16:01 /home/kbhavesh/sample.txt

$ chgrp testgrp /home/kbhavesh/sample.txt

$ ls -l
-rw-r--r-- 1 kbhavesh testgrp 210 2012-01-04 16:01 /home/kbhavesh/sample.txt
```

To change the group name of link files

Using –derefence option, the group of referent of symbolic link file can be changed and with –no-dereference option, the group of symbolic link file can be changed as shown below.

```
# the group name of actual file pointed by sym_link gets changed.
$ chgrp --dereference guest sym_link

# the group name of sym_link gets changed.
$ chgrp --no-dereference user sym_link

$ ls -l sym_link richard_readme.txt
lrw-r--r--  1 test user    4  2011-01-04 15:52 sym_link -> readme.txt
-rw-r--r-- 1 test guest  20 2011-01-04 15:52 readme.txt
```

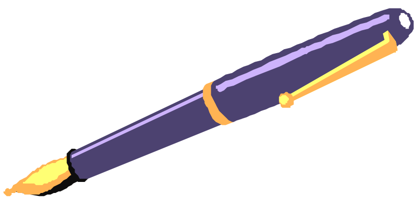
Note : –dereference is the default option.

To change the group for list of files

Using -R option, the list of file's group name can be modified as shown below,

```
$ chgrp -R guest dir/

$ ls -l dir/
total 8
-rw-r--r-- 1 test guest 20 2011-01-04 16:50 readme.txt
-rw-r--r-- 1 test guest 20 2011-01-04 16:50 changes.txt
```

7. Scheduling Tasks with Cron



cron is a Linux system process that will execute a program at a preset time. To use cron you must prepare a text file that describes the program that you want executed and the times that cron should execute them. Then you use the crontab program to load the text file that describes the cron jobs into cron.

- Cron is used to automate periodic tasks (eg: backup the hard drive at every Sunday at 3 a.m. or email yourself a reminder about the Linux Users Group meeting the first day of every month)

- Users setup the configuration file for cron using the command: crontab
    - to create or modify the configuration file use: crontab -e
      This will allow you to edit your own crontab using vi
    - to view the contents of the file use: crontab -l
    - to delete the file use: crontab -r

- Your configuration file is stored in the directory: /var/spool/crontabs under your username.
- The crond daemon checks the crontab files every minute to determine if a task should be launched in that minute.
- Each line in a crontab file has 6 fields:
    - minute
    - hour
    - day of the month
    - month
    - day of the week
    - command
- If the command produces output, the output will be sent to you in an email message.

- **Example 1:**
  19   20   *   *   Sun   echo Study Linux
  Every Sunday, at 20:19, echo Study Linux (you will receive this as an email message)

- **Example 2:**
  13,18,23,28   5   *   *   Mon-Fri   /usr/bin/who
  Monday to Friday, at 5:13, 5:18, 5:23 and 5:28 execute /usr/bin/who

- **Example 3:**
  43   */2   1   *   *   echo Pay Bills
  On the first day of every month, every 2 hours at 43 minutes past the hour, echo Pay Bills.

Syntax of Cron Job

Syntax: minute   hour   day   month   dayofweek   command

1. **minute** — any integer from 0 to 59
2. **hour** — any integer from 0 to 23
3. **day** — any integer from 1 to 31 (must be a valid day if a month is specified)
4. **month** — any integer from 1 to 12 (or the short name of the month such as jan or feb)
5. **dayofweek** — any integer from 0 to 7, where 0 or 7 represents Sunday (or the short name of the week such as sun or mon)

6. **command** — Full path of binary with command options or the command to execute (the command can either be a command such as ls /proc >> /tmp/proc or the command to execute a custom script)

Execute every minute
If you leave the star, or asterisk, it means **every**. Maybe that's a bit unclear. Let's use the the previous example again:

```
* * * * * /home/kbhavesh/script.sh
```

They are all still asterisks! So this means execute /home/kbhavesh/script.sh:

1. **every** minute
2. of **every** hour
3. of **every** day of the month
4. of **every** month
5. and **every** day in the week.

Execute every Friday 1AM
So if we want to schedule the script to run at 1AM every Friday, we would need the following cronjob:

```
0 1 * * 5 /home/kbhavesh/script.sh
```

The script is now being executed when the system clock hits:

1. minute: 0
2. of hour: 1
3. of day of month: * (every day of month)
4. of month: * (every month)
5. and weekday: 5 (=Friday)

Execute 10 past after every hour on the 1st of every month

```
10 * 1 * * /home/kbhavesh/script.sh
```

Every Sunday, at 20:19, echo Study Linux (you will receive this as an email message)

```
19   20   *   *   Sun   echo Study Linux
```

Here are some examples, just for practicing

To run a cron on every half hour

```
1-31 * * * * command
==OR==
*/31 * * * * command
```

To run a cron on every two minutes

```
*/2 * * * * command
```

To run custom script the first day of every month at 4:10AM

```
10 4 1 * * /root/scripts/backup.sh
```

To run custom script the first day January of every year at 4:10AM

```
10 4 1 Jan * sh /home/guest/backup.sh
```

To run a cron on every 30 minutes

```
*/30 * * * * tar -cvvf /root/cron/cron.tar /root/cron/
```

Before going to start, it's important to ensure that crond daemon is running and not hanged in the system. To manage crond daemon in Linux, we can make use of "service" command.

To check the status of crond daemon:
service crond status
To stop and terminate crond process:
service crond stop
To start and run crond daemon:
service crond start
To restart crond service:
service crond restart

Steps to create Cron Job

<1> Create txt File for scheduling tasks: cron

Example: mscitdata.txt

```
$ vi mscitdata.txt
# run custom script the first day of every month at 4:10AM
*/1 * * * * /root/cron/abc
```

<2> Add task to crontab

```
$ crontab mscitdata.txt
```

<3> check whether your take has been added to crontab or not

```
$ crontab –l
```

Storing the crontab output

By default cron saves the output of /home/kbhavesh/script.sh in the user's mailbox (root in this case). But it's prettier if the output is saved in a separate logfile. Here's how:

```
*/10 * * * * /home/kbhavesh/script.sh 2>&1 >> /var/log/script_output.log
```

Mailing the crontab output

By default cron saves the output in the user's mailbox (root in this case) on the local system. But you can also configure crontab to forward all output to a real email address by starting your crontab with the following line:

```
MAILTO="yourname@yourdomain.com"
```

Mailing the crontab output of just one cronjob

If you'd rather receive only one cronjob's output in your mail, make sure this package is installed:

```
Yum/aptitude install mailx
```

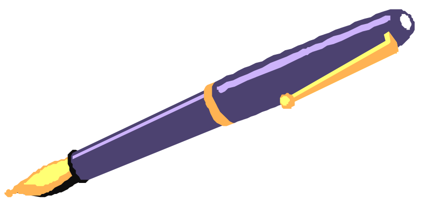
And change the cronjob like this:

```
*/10 * * * * /home/kbhavesh/script.sh 2>&1 | mail -s "Cronjob ouput" yourname@yourdomain.com
```

Exercises:

1. Setup a cron job to print memory status in file on first day of every month. (hint : cat /proc/meminfo >> /home/guest/meminfo)
2. Setup a cron job to echo Study Linux every 30 minutes, Monday to Friday.
3. Setup a cron job to echo Study Linux every hour on Saturday and Sunday.
4. Setup a script that displays the date and then a list of everyone who is logged on. Run your script and make sure that it works. Setup cron to run the script you created on the first day of every month at 08:02.
5. Setup a script called backup that echoes the message "You should backup now". Run your script and make sure that it works. Setup cron to run the script you created in question two the first day of every month at 07:17.
6. When you are sure that this is working correctly, store the output from crontab -l in the file cronlist.
7. Run crontab -r to delete all of your cron tasks.
8. Setup a cron job to take backup of guest home on the first day January of every year at 4:10AM
9. When you are sure that this is working correctly, store the output from crontab -l in the file cronlist.
10. Setup a cron job to say(echo) “Good Morning”, and “Good Evening” Monday to Friday (use system time to decide what message to print)

8. Linux Networking



Commands: ping, ftp , ssh, telnet, finger, traceroute, host scp, rsync

8.1 PING

ping [*options*] *host*

It is administrator command. Confirm that a remote host is online and responding. Ping is intended for use in network testing, measurement, and management. Because of the load it can impose on the network, it is unwise to use ping during normal operations or from automated scripts.

```
[root@localhost ~]# ping 172.16.16.3
```

8.2 FTP

ftp [*options*] [*hostname*]

Transfer files to and from remote network site *hostname*. ftp prompts the user for a command. The commands are listed after the options. Some of the commands are toggles, meaning they turn on a feature when it is off and vice versa. Note that versions may have different options.

First step is to start vsftpd service
1. service vsftpd start

And then
2. add ftp user into ftp group

The following examples illustrate typical uses of the command ftp for remotely copying, renaming, and deleting files.

ftp abc.xyz.edu
This command will attempt to connect to the ftp server at abc.xyz.edu. If it succeeds, it will ask you to log in using a username and password. Public ftp servers often allow you to log in using the username "anonymous" and your email address as password. Once you are logged in you can get a list of the available ftp commands using the help function:

ftp> help
This lists the commands that you can use to show the directory contents, transfer files, and delete files.

ftp> ls
This command prints the names of the files and subdirectories in the current directory on the remote computer.

ftp> cd customers
This command changes the current directory to the subdirecotry "customers", if it exists.

ftp> cd ..
Changes the current directory to the parent direcotry.

ftp> lcd images
Changes the current directory [em]on the local computer[/em] to "images", if it exists.

ftp> ascii
Changes to "ascii" mode for transferring text files.

ftp> binary
Changes to "binary" mode for transferring all files that are not text files.

ftp> get image1.jpg
Downloads the file image1.jpg from the remote computer to the local computer. Warning: If there already is file with the same name it will be overwritten.

ftp> put image2.jpg
Uploads the file image2.jpg from the local computer to the remote computer. Warning: If there already is file with the same name it will be overwritten.

ftp> !ls
A '!' in front will execute the specified command on the local computer. So '!ls' lists the file names and directory names of the current directory on the local computer.

ftp> mget *.jpg
With mget you can download multiple images. This command downloads all files that end with ".jgp".

ftp> mput *.jpg
Uploads all files that end with ".jgp".

ftp> mdelete *.jpg
Deletes all files that end with ".jgp".

ftp> prompt
Turns iteractive mode on or off so that commands on multiple files are executed without user confirmation.

ftp> quit
Exits the ftp program.

8.3 SSH

sh [*options*] hostname [*command*]

Securely log a user into a remote system and run commands on that system. The version of ssh described here is the OpenSSH client. ssh can use either Version 1 (SSH1) or Version 2 (SSH2) of the SSH protocol. SSH2 is preferable, as it provides stronger encryption methods and greater connection integrity. The hostname can be specified either as *hostname* or as user@hostname. If a command is specified, the user is authenticated, the command is executed, and the connection is closed. Otherwise, a terminal session is opened on the remote system. See Escape characters," later in this section, for functions that can be supported through an escape character. The default escape character is a tilde (~). The exit status returned from ssh is the exit status from the remote system, or 255 if there was an error.

Commonly, authentication is handled with standard username/password credentials, but it can also be useful to authenticate with a key exchange. This is done by generating a key on the client with ssh-keygen and populating the known_hosts file on the remote host.

```
[root@localhost ~]# ssh root@172.16.16.3
password *******
```

8.4 TELNET

telnet [*options*] [*host* [*port*]]

Access remote systems. telnet is the user interface that communicates with another host using the Telnet protocol. If telnet is invoked without *host*, it enters command mode, indicated by its prompt, telnet>, and accepts and executes commands. Type ? at the command prompt to see the available commands. If invoked with arguments, telnet performs an open command (shown in the following list) with those arguments. *host* indicates the host's official name, alias, or Internet address. *port* indicates a port number (default is the Telnet port).

The Telnet protocol is often criticized because it uses no encryption and makes it easy for snoopers to pick up user passwords. Most sites now use ssh instead.

```
[root@localhost ~]# telnet 172.16.16.3
password *******
```

8.5 FINGER

finger [*options*] *users*

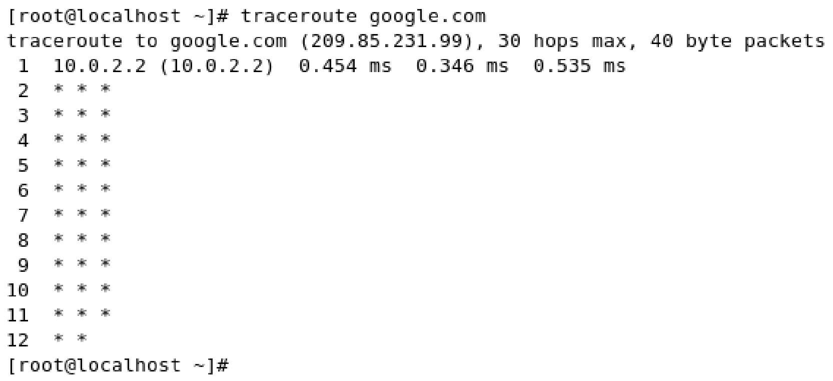
Display data about one or more *users*, including information listed in the files *.plan* and *.project* in each user's home directory. You can specify each user either as a login name (exact match) or as a first or last name (display information on all matching names). Networked environments recognize arguments of the form user@host and @host.

```
[root@localhost ~]# finger guest
```

8.6 TRACEROUTE

traceroute [*options*] *host* [*packetsize*]

TCP/IP command. Trace route taken by packets to reach network host. traceroute attempts tracing by launching UDP probe packets with a small TTL (time-to-live), then listening for an ICMP "time exceeded" reply from a gateway. *host* is the destination hostname or the IP number of the host to reach. *packetsize* is the packet size in bytes of the probe datagram. Default is 40 bytes.

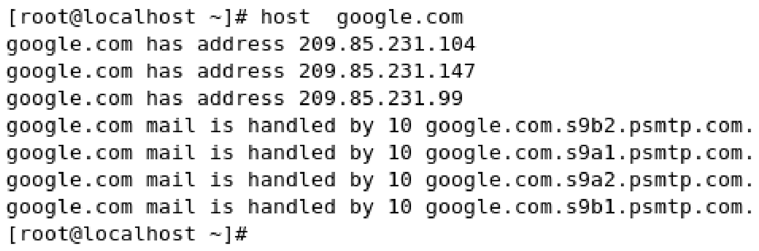


```
[root@localhost ~]# traceroute google.com
traceroute to google.com (209.85.231.99), 30 hops max, 40 byte packets
 1  10.0.2.2 (10.0.2.2)  0.454 ms  0.346 ms  0.535 ms
 2  * * *
 3  * * *
 4  * * *
 5  * * *
 6  * * *
 7  * * *
 8  * * *
 9  * * *
10  * * *
11  * * *
12  * *
[root@localhost ~]#
```

8.7 HOST

host [*options*] *name* [*server*]

It is administrator command. Print information about hosts or zones in DNS. Hosts may be IP addresses or hostnames; host converts IP addresses to hostnames by default and appends the local domain to hosts without a trailing dot. Default servers are determined in */etc/resolv.conf*.



```
[root@localhost ~]# host  google.com
google.com has address 209.85.231.104
google.com has address 209.85.231.147
google.com has address 209.85.231.99
google.com mail is handled by 10 google.com.s9b2.psmtp.com.
google.com mail is handled by 10 google.com.s9a1.psmtp.com.
google.com mail is handled by 10 google.com.s9a2.psmtp.com.
google.com mail is handled by 10 google.com.s9b1.psmtp.com.
[root@localhost ~]#
```

8.8 SCP

scp copies files between hosts on a network. It uses ssh for data transfer, and uses the same authentication and provides the same security as ssh. scp will ask for passwords or passphrases if they are needed for authentication. Any file name may contain a host and user specification to indicate that the file is to be copied to/from that host. Copies between two remote hosts are permitted.

When copying a source file to a target file which already exists, scp will replace the contents of the target file (keeping the inode). When copying a source file to a target file which already exists, scp will replace the contents of the target file (keeping the inode).

If the target file does not yet exist, an empty file with the target file name is created, and then filled with the source file contents. No attempt is made at "near-atomic" transfer using temporary files.

Usage

scp - secure copy (remote file copy program)

scp [option][[user@]host1:]file1 [...] [[user@]host2:]file2

scp [[user@]from-host:]source-file [[user@]to-host:][destination-file]

Description of options

from-host

Is the name or IP of the host where the source file is, this can be omitted if the from-host is the host where you are actually issuing the command

user

Is the user which have the right to access the file and directory that is supposed to be copied in the cas of the fromhost and the user who has the rights to write in the to-host

source-file

Is the file or files that are going to be copied to the destination host, it can be a directory but in that case you need to specify the *-r* option to copy the contents of the directory

destination-file

Is the name that the copied file is going to take in the to-host, if none is given all copied files are going to maintain its names

Example
```
scp –r /home/kbhavesh/abc test@172.16.16.15:/home/test/Desktop
password: ******
```

8.9 RSYNC

Remote synchronize, updates (synchronizes) files between two different networked machines. rsync is a program that behaves in much the same way that rcp does, but has many more options and uses the rsync remote-update protocol to greatly speed up file transfers when the destination file is being updated.

The rsync remote-update protocol allows rsync to transfer just the differences between two sets of files across the network connection, using an efficient checksum-search algorithm described in the technical report that accompanies this package.

Some of the additional features of rsync are:

- support for copying links, devices, owners, groups, and permissions
- exclude and exclude-from options similar to GNU tar
- does not require super-user privileges
- rsync is used to perform the backup operation in UNIX / Linux.
- rsync utility is used to synchronize the files and directories from one location to another in an effective way. Backup location could be on local server or on remote server
- **Speed**: First time, rsync replicates the whole content between the source and destination directories. Next time, rsync transfers only the changed blocks or bytes to the destination location, which makes the transfer really fast.
- **Security**: rsync allows encryption of data using ssh protocol during transfer.
- **Less Bandwidth**: rsync uses compression and decompression of data block by block at the sending and receiving end respectively. So the bandwidth used by rsync will be always less compared to other file transfer protocols.
- **Privileges**: No special privileges are required to install and execute rsync

Syntax

rsync command- Remote file copy (Synchronize file trees)
```
# Local file to Local file
rsync [option]... Source [Source]... Dest
```

```
# Local to Remote
rsync [option]... Source [Source]... [user@]host:Dest
# Remote to Local
rsync [option]... [user@]host::Source [Dest]
```

Examples:

Copy entire directory (all folders/files and sub folders) from one host to another

```
rsync -r /root/Desktop/abc kbhavesh@17.16.16.15:/home/kbhavesh/
password: *****
```

Synchronize Two Directories in a Local Server
To sync two directories in a local computer, use the following rsync -zvr command.

```
rsync -zvr abc/ /home/kbhavesh/abc

rsync -r /root/Desktop/abc kbhavesh@localhost:/home/kbhavesh/abc
password: *****
```

In the above rsync example:

- -z is to enable compression
- -v verbose
- -r indicates recursive

Synchronize Files from Local to Remote

```
rsync -zvr abc/ kbhavesh@172.16.16.15:/home/kbhavesh/abc
```

Synchronize Files from Remote to Local

```
rsync -avz kbhavesh@172.16.16.15:/home/kbhavesh/abc /root/Desktop/abc
```

rsync option -a indicates archive mode. -a option does the following,

- Recursive mode
- Preserves symbolic links
- Preserves permissions
- Preserves timestamp
- Preserves owner and group

Synchronize only the Directory Tree Structure

```
rsync -v -d kbhavesh@172.16.16.15:/home/kbhavesh/abc/ .
```

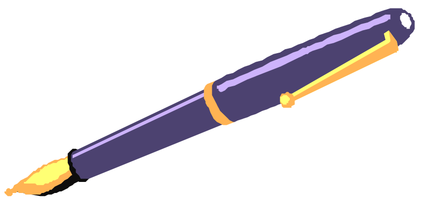
View the rsync Progress during Transfer

```
rsync -avz --progress kbhavesh@localhost:/home/kbhavesh/abc /root/Desktop/abc
```

Delete the Files Created at the Target

```
rsync -avz --delete kbhavesh@localhost:/home/kbhavesh/abc /root/Desktop/abc
```

9. Package Management



Commands: rpm, yum, wget, cpan

9.1 RPM

rpm [*options*]

The Red Hat Package Manager. A freely available packaging system for software distribution and installation. RPM packages are built, installed, and queried with the **rpm** and **rpmbuild** commands. For detailed information on RPM

RPM commands for installing, uninstalling, upgrading, querying, listing, and checking RPM packages on your Red Hat Linux system.

# rpm -ivh foo-2.0-4.i386.rpm
# rpm -i ftp://ftp.redhat.com/pub/redhat/RPMS/foo-1.0-1.i386.rpm
# rpm -i http://oss.oracle.com/projects/firewire/dist/files/kernel-2.4.20-18.10.1.i686.rpm

Used to install a RPM package. Note that RPM packages have file naming conventions like foo-2.0-4.i386.rpm, which include the package name (foo), version (2.0), release (4), and architecture (i386). Also notice that RPM understands FTP and HTTP protocols for installing and querying remote RPM files.

# **rpm -e foo**
To uninstall a RPM package. Note that we used the package name foo, not the name of the original package file foo-2.0-4.i386.rpm above.

# **rpm -Uvh foo-1.0-2.i386.rpm**
# **rpm -Uvh ftp://ftp.redhat.com/pub/redhat/RPMS/foo-1.0-1.i386.rpm**
# **rpm -Uvh http://oss.oracle.com/projects/firewire/dist/files/kernel-2.4.20-18.10.1.i686.rpm**

To upgrade a RPM package. Using this command, RPM automatically uninstall the old version of the foo package and install the new package. It is safe to always use rpm -Uvh to install and upgrade packages, since it works fine even when there are no previous versions of the package installed! Also notice that RPM understands FTP and HTTP protocols for upgrading from remote RPM files.

# **rpm -qa**
To query all installed packages. This command will print the names of all installed packages installed on your Linux system.

# **rpm -q foo**
To query a RPM package. This command will print the package name, version, and release number of the package foo only if it is installed. Use this command to verify that a package is or is not installed on your Linux system.

# **rpm -qi foo**
To display package information. This command display package information including the package name, version, and description of the installed program. Use this command to get detailed information about the installed package.

# **rpm -ql foo**

To list files in installed package. This command will list all of files in an installed RPM package. It works only when the package is already installed on your Linux system.

# **rpm -qf /usr/bin/mysql**
mysql-3.23.52-3
Which package owns a file? This command checks to determine which installed package a particular file belongs to.

# **rpm -qpl kernel-2.4.20-18.10.1.i686.rpm**
# **rpm -qpl ftp://ftp.redhat.com/pub/redhat/RPMS/foo-1.0-1.i386.rpm**
# **rpm -qpl http://oss.oracle.com/projects/firewire/dist/files/kernel-2.4.20-18.10.1.i686.rpm**

List files in RPM file. This command allows you to query a (possibly) uninstalled RPM file with the use of the the "-p" option. You can use the "-p" option to operate on an RPM file without actually installing anything. This command lists all files in an RPM file you have in the current directory. Also note that RPM can query remote files through the FTP and HTTP protocols.

9.2 YUM

Yum is software installation tool for Red hat linux and Fedora Linux. It is a complete software management system. Other option is to use up2date utility. Yum is designed to use over network/internet. It does not use CDROM to install packages.

Example

yum Fedora Linux Howto

If you don't have yum then download it from project home page And then install it

```
rpm -ivh yup*
```

Step # 1: Configure yum
You need to edit /etc/yum.conf and modify/add following code to it:

```
vi /etc/yum.conf
```

Append or edit code as follows:

```
[base]
name=Fedora Core $releasever - $basearch - Base
baseurl=http://apt.sw.be/fedora/$releasever/en/$basearch/dag
baseurl=http://mirrors.kernel.org/fedora/core/$releasever/$basearch/os
```

Save the file

Install GPG signature key with rpm command:

```
# rpm --import http://dag.wieers.com/packages/RPM-GPG-KEY.dag.txt
```

and other keys too (if any using above command)

Step # 2 Update your package list:

```
# yum check-update
```

Step # 3 start to use yum

Install a new package called foo

```
# yum install foo
```

To update packages

```
# yum update
```

To update a single package called bar

```
# yum update bar
```

To remove a package called telnet

```
# yum remove telnet
```

To list all packages

```
# yum list installed
```

You can search using grep command

```
# yum list installed | grep samba
```

Display information on a package called foo

```
# yum info foo
```

To display list of packages for which updates are available

```
# yum list updates
```

9.3 WGET

Syntax

wget [*options*] [*urls*]

Perform non-interactive file downloads from the Web. **wget** works in the background and can be used to set up and run a download without the user having to remain logged on. **wget** supports HTTP, HTTPS, FTP, as well as downloads through HTTP proxies. **wget** uses a global startup file that you may find at /etc/wgetrc or /usr/local/etc/wgetrc. In addition, users can define their own &dollar;HOME/.wgetrc files.

Example :

wget http://www.xyz.co/abc.exe

Specify Download Speed / Download Rate Using wget –limit-rate

While executing the wget, by default it will try to occupy full possible bandwidth. This might not be acceptable when you are downloading huge files on production servers. So, to avoid that we can limit the download speed using the –limit-rate as shown below. In the following example, the download speed is limited to 200k

wget --limit-rate=200k http://www.openss7.org/.../strx25-0.9.2.1.tar.bz2

Continue the Incomplete Download Using wget –c

Restart a download which got stopped in the middle using wget -c option as shown below.

```
wget -c http://www.openss7.org/repos/tarballs/strx25-0.9.2.1.tar.bz2
```

Download in the Background Using wget –b

For a huge download, put the download in background using wget option -b as shown below.

```
wget -b http://www.openss7.org/repos/tarballs/strx25-0.9.2.1.tar.bz2

Continuing in background, pid 1984.
Output will be written to `wget-log'.
```

It will initiate the download and gives back the shell prompt to you. You can always check the status of the download using tail -f as shown below.

```
$ tail -f wget-log
Saving to: `strx25-0.9.2.1.tar.bz2.4'
0K .......... .......... .......... .......... ..........    1% 65.5K 57s
50K .......... .......... .......... .......... ..........  2% 85.9K 49s
250K .......... .......... .......... .......... .......... 7% 182M 46s
300K .......... .......... .......... .......... .......... 9% 57.9K 47s
```

Test Download URL Using wget –spider

When you are going to do scheduled download, you should check whether download will happen fine or not at scheduled time. To do so, copy the line exactly from the schedule, and then add –spider option to check.

```
wget --spider DOWNLOAD-URL
```

If the URL given is correct, it will say

```
wget --spider download-url
Spider mode enabled. Check if remote file exists.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]
Remote file exists and could contain further links, but recursion is disabled -- not retrieving.
```

Increase Total Number of Retry Attempts Using wget –tries

If the internet connection has problem, and if the download file is large there is a chance of failures in the download. By default wget retries 20 times to make the download successful. If needed, you can increase retry attempts using –tries option as shown below.

```
wget --tries=75 DOWNLOAD-URL
```

Download Multiple Files / URLs Using Wget –i

First, store all the download files or URLs in a text file as:

```
$ cat > download-file-list.txt
URL1
URL2
URL3
URL4
```

Next, give the download-file-list.txt as argument to wget using -i option as shown below.

```
$ wget -i download-file-list.txt
```

Download a Full Website Using wget –mirror

Following is the command line which you want to execute when you want to download a full website and made available for local viewing.

```
$ wget --mirror -p --convert-links -P ./LOCAL-DIR WEBSITE-URL
```

- –mirror : turn on options suitable for mirroring.
- -p : download all files that are necessary to properly display a given HTML page.
- –convert-links : after the download, convert the links in document for local viewing.
- -P ./LOCAL-DIR : save all the files and directories to the specified directory.

Reject Certain File Types while Downloading Using wget –reject

You have found a website which is useful, but don’t want to download the images you can specify the following.

```
$ wget --reject=gif WEBSITE-TO-BE-DOWNLOADED
```

Download Only Certain File Types Using wget -r –A

You can use this under following situations:

- Download all images from a website
- Download all videos from a website
- Download all PDF files from a website

```
$ wget -r -A.pdf http://url-to-webpage-with-pdfs/
```

FTP Download With wget

You can use wget to perform FTP download as shown below. Anonymous FTP download using Wget

```
$ wget ftp-url
```

FTP download using wget with username and password authentication

```
$ wget --ftp-user=USERNAME --ftp-password=PASSWORD DOWNLOAD-URL
```

9.4 CPAN

cpan - easily interact with CPAN from the command line
CPAN is the **C**omprehensive **P**erl **A**rchive **N**etwork, a large collection of Perl software and documentation. You can begin exploring from either http://www.cpan.org/, http://www.perl.com/CPAN/ or any of the mirrors listed at http://www.cpan.org/SITES.html.
Example:

```
cpan> install OpenGL
OpenGL is up to date.
cpan> force install OpenGL
Running make
OpenGL-0.4/
OpenGL-0.4/COPYRIGHT
```

10. Web server

Apache Web Server

Apache is probably the most widely used open source software today. It is used to host over 50% of all websites in existence1 and is usually chosen for its maturity, stability, and flexibility. It is designed to be modular, so extra functionality can be added or removed by enabling or disabling modules. Packages are available for virtually all Linux distributions, so you can install it on your hosts via the package management system.

Apache is probably the most popular Linux-based Web server application in use. Once you have DNS correctly setup and your server has access to the Internet, you'll need to configure Apache to accept surfers wanting to access your Web site.

Download and Install the Apache Package

Most RedHat/CentOS and Fedora Linux software products are available in the RPM format. When searching for the file, remember that the Apache RPM's filename usually starts with the word httpd followed by a version number, as in httpd-2.0.48-1.2.rpm. It is best to use the latest version of Apache

When searching for the file, remember that the Redhat / CentOS / Fedora Apache RPM package's filename usually starts with the word httpd followed by a version number, as in httpd- 2.0.48-1.2.rpm. With Ubuntu / Debian the package name will have the apache prefix instead.

How To Get Apache Started

Setting up the Apache server is easy to do, but the procedure differs between Linux distributions.
Redhat / Cent OS/ Fedora

Use the chkconfig command to configure Apache to start at boot:

```
[root@bigboy tmp]# chkconfig httpd on
```

Use the httpd<code> init script in the <code>/etc/init.d directory to start,stop, and restart Apache after booting:

```
[root@bigboy tmp]# /etc/init.d/httpd start
[root@bigboy tmp]# /etc/init.d/httpd stop
[root@bigboy tmp]# /etc/init.d/httpd restart
```

Explain steps to create Virtual Host on Local Machine - http://newdomain.com

Step 1: Do Entry in Host File

```
vi /etc/hosts
```

```
# Do not remove the following line, or various programs
# that require network functionality will fail.
127.0.0.1    localhost.localdomain    localhost     newdomain
::1     localhost6.localdomain6localhost6
```

```
172.16.16.4  newdomain.com   newdomain
```

Step 2: Create Directory for Host on local PC

```
mkdir /var/www/html/newdomain.com
touch /var/www/html/newdomain.com/index.php
echo "<?php echo "Welcome"; ?> " >> /var/www/html/newdomain.com/index.php
chmod 777 –R /var/www/html/newdomain.com
```

Step 3: Open file /etc/httpd/conf/httpd.conf and do following changes

```
NameVirtualHost 172.16.16.4

# Virtual host Default Virtual Host
<VirtualHost *>
ServerSignature email
DirectoryIndex  index.php index.html index.htm index.shtml
LogLevel  warn
HostNameLookups off
</VirtualHost>

# Virtual host newdomain.com
<VirtualHost 172.16.16.4>
        DocumentRoot /var/www/html/newdomain.com
        ServerAdmin kbhavesh@newdomain.com
        ServerName newdomain.com
        ServerAlias newdomain.com
        DirectoryIndex index.html index.php index.htm index.shtml
        <Directory "/var/www/html/newdomain.com/">
        Options all
        AllowOverride all
        </Directory>
</VirtualHost>
```

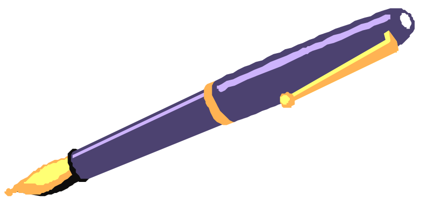
Step 4: Save httpd.conf file then restart httpd service and open http://newdomain.com in web browser

```
[root@localhost me]# service httpd restart
```

Step5 - Access domain from Windows [Optional]

C:\Windows\System32\drivers\etc\host
Add 172.16.16.4   newdomain.com
Then you can access by http://newdomain.com

11. Samba Server



You can start/stop/restart Samba after boot time using the smb initialization script as in the examples below:

```
[root@localhost tmp]# service smb start
[root@localhost tmp]# service smb stop
[root@localhost tmp]# service smb restart
```

Steps to share directory using command line

1. Add local user on Fedora Core system.

```
[root@localhost samba]# adduser kbhavesh
[root@localhost samba]#
[root@localhost samba]# passwd kbhavesh
Changing password for user kbhavesh.
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully.
```

2. Add samba user and samba password in **smbpasswd**.

```
[root@localhost samba]# smbpasswd -a kbhavesh
New SMB password:
Retype new SMB password:
Added user kbhavesh
```

3. Verify and add user to the list of samba users (**smbusers**).

```
[root@localhost samba]# cat /etc/samba/smbusers
# Unix_name = SMB_name1 SMB_name2 ...
root = administrator admin
nobody = guest pcguest smbguest
veronica = veronica

[root@localhost samba]# echo "kbhavesh = kbhavesh" >> /etc/samba/smbusers
[root@localhost samba]# cat /etc/samba/smbusers
# Unix_name = SMB_name1 SMB_name2 ...
root = administrator admin
nobody = guest pcguest smbguest
veronica = veronica
kbhavesh = kbhavesh
```

Create Samba share directory

1. Creating Samba share directory name 'smb_share' for all samba user in /home directory. The command example below shown on the creation of /home/smb_share directory with the 755 permission for all Samba users:

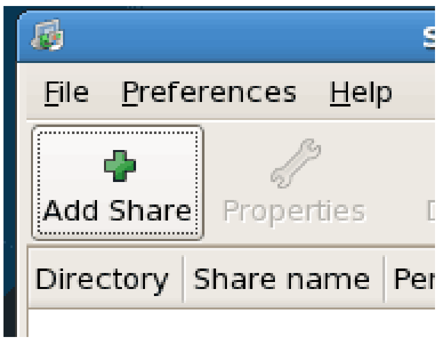
```
[root@localhost ~]# mkdir -m 4755 /home/smb_share
```

2. Verify the creating of Samba share directory

```
[root@localhost ~]# ls -al /home/
total 44
drwxr-xr-x 9 root root 4096 Apr 3 04:40 .
drwxr-xr-x 25 root root 4096 Apr 3 03:42 ..
drwsr-xr-x 2 root root 4096 Apr 3 04:40 smb_share
```

Enable Samba share directory

1. To add share directory / folder for Samba, You can click either the **Add Share** button or go to and click **File** -> **Add Share** from the menu on the **Samba Configuration** window
2.



OR

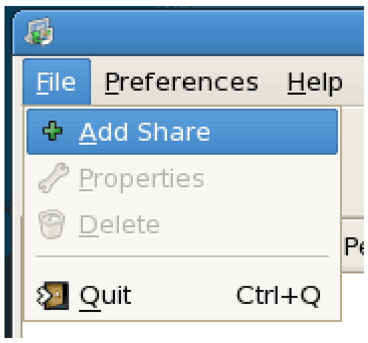
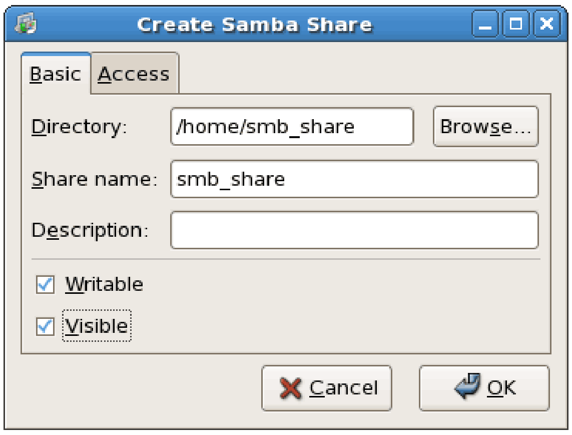


Figure: Add directory to share

2. On the Create Samba share window, on the Basic tab, you can configure the samba share directory, share name and directory description. Remember that we already create the directory named **smb_share**, now click Browse button and point to the directory that we create earlier, then point and click the Access tab to proceed.

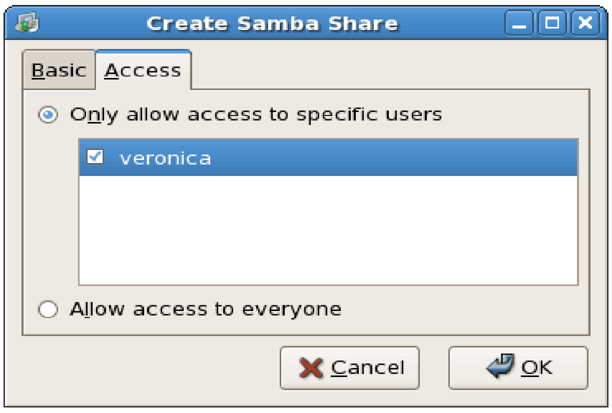
Example of the Samba share Basic configuration:



Directory: /home/smb_share
Share name: smb_share
Description:
☑ Writable
☑ Visible

Figure: Select Directory from Local System

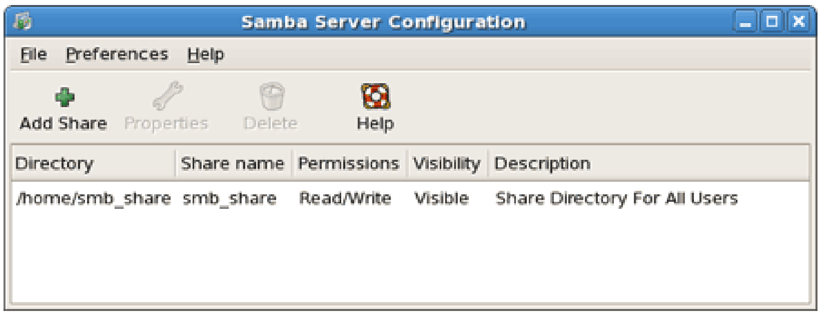
3. On the **Access** tab, you can specify the user that can be allow accessing the share directory, you can select the user from the list or you can click the radio button to allow access to everyone.



● Only allow access to specific users
☑ veronica
○ Allow access to everyone

Figure: Select user to give access permission

4. The example of samba share directory.



Directory	Share name	Permissions	Visibility	Description
/home/smb_share	smb_share	Read/Write	Visible	Share Directory For All Users

Figure: validate

5. Restart samba service

```
[root@localhost tmp]# service smb restart
```

12. FTP Server

FTP Control Channel, TCP Port 21: All commands you send and the ftp server's responses to those commands will go over the control connection, but any data sent back (such as "ls" directory lists or actual file data in either direction) will go over the data connection.

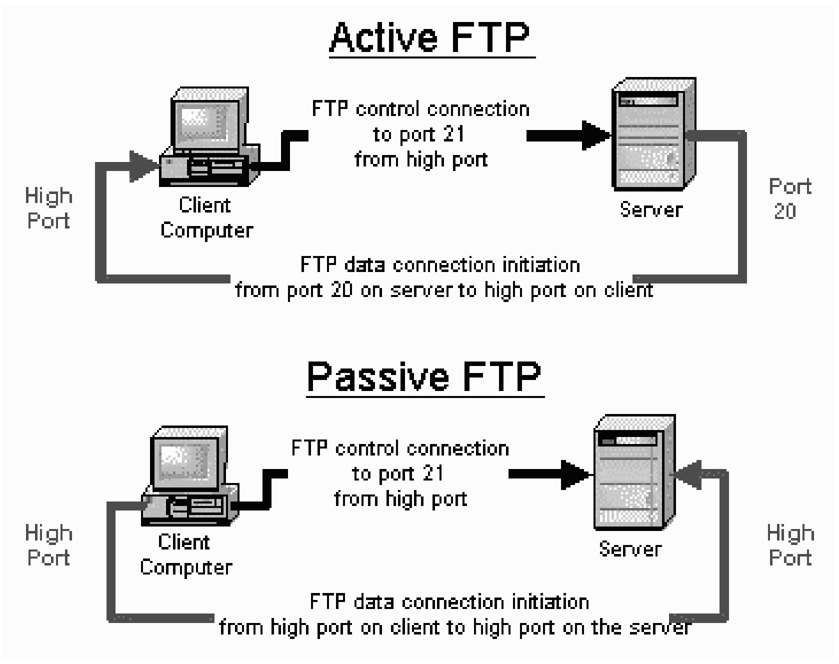
FTP Data Channel, TCP Port 20: This port is used for all subsequent data transfers between the client and server.

In addition to these channels, there are several varieties of FTP.

Types of FTP

From a networking perspective, the two main types of FTP are active and passive. In active FTP, the FTP server initiates a data transfer connection back to the client. For passive FTP, the connection is initiated from the FTP client. These are illustrated in Figure 1.

Figure 1 Active And Passive FTP Illustrated


Active FTP
FTP control connection to port 21 from high port
High Port
Client Computer
Server
Port 20
FTP data connection initiation from port 20 on server to high port on client
Passive FTP
FTP control connection to port 21 from high port
High Port
Client Computer
Server
High Port
FTP data connection initiation from high port on client to high port on the server

From a user management perspective there are also two types of FTP: regular FTP in which files are transferred using the username and password of a regular user FTP server, and anonymous FTP in which general access is provided to the FTP server using a well-known universal login method.

Take a closer look at each type.

Active FTP

The sequence of events for active FTP is:

1. Your client connects to the FTP server by establishing an FTP control connection to port 21 of the server. Your commands such as 'ls' and 'get' are sent over this connection.
2. Whenever the client requests data over the control connection, the server initiates data transfer connections back to the client. The source port of these data transfer connections is always port 20 on the server, and the destination port is a high port (greater than 1024) on the client.
3. Thus the ls listing that you asked for comes back over the port 20 to high port connection, not the port 21 control connection.

FTP active mode therefore transfers data in a counter intuitive way to the TCP standard, as it selects port 20 as it's source port (not a random high port that's greater than 1024) and connects back to the client on a random high port that has been pre-negotiated on the port 21 control connection.

Active FTP may fail in cases where the client is protected from the Internet via many to one NAT (masquerading). This is because the firewall will not know which of the many servers behind it should receive the return connection.

Passive FTP

Passive FTP works differently:

1. Your client connects to the FTP server by establishing an FTP control connection to port 21 of the server. Your commands such as ls and get are sent over that connection.
2. Whenever the client requests data over the control connection, the client initiates the data transfer connections to the server. The source port of these data transfer connections is always a high port on the client with a destination port of a high port on the server.

Passive FTP should be viewed as the server never making an active attempt to connect to the client for FTP data transfers. Because client always initiates the required connections, passive FTP works better for clients protected by a firewall.

As Windows defaults to active FTP, and Linux defaults to passive, you'll probably have to accommodate both forms when deciding upon a security policy for your FTP server.

Regular FTP

By default, the VSFTPD package allows regular Linux users to copy files to and from their home directories with an FTP client using their Linux usernames and passwords as their login credentials.

VSFTPD also has the option of allowing this type of access to only a group of Linux users, enabling you to restrict the addition of new files to your system to authorized personnel.

The disadvantage of regular FTP is that it isn't suitable for general download distribution of software as everyone either has to get a unique Linux user account or has to use a shared username and password. Anonymous FTP allows you to avoid this difficulty.

Anonymous FTP

Anonymous FTP is the choice of Web sites that need to exchange files with numerous unknown remote users. Common uses include downloading software updates and MP3s and uploading diagnostic information for a technical support engineers' attention. Unlike regular FTP where you login with a preconfigured Linux username and password, anonymous FTP requires only a username of anonymous and your email address for the password. Once logged in to a VSFTPD server, you automatically have access to only the default anonymous FTP directory (/var/ftp in the case of VSFTPD) and all its subdirectories.

Installing vsftpd

Most RedHat and Fedora Linux software product packages are available in the RPM format, When searching for these packages remember that the filename usually starts with the software package name and is followed by a version number, as in vsftpd-1.2.1-5.i386.rpm.

Starting vsftpd

The methodologies vary depending on the variant of Linux you are using as you'll see next.

Fedora / CentOS / RedHat

With these flavors of Linux you can use the chkconfig command to get vsftpd configured to start at boot:

```
[root@172.16.16.15 tmp]# chkconfig vsftpd on
```

To start, stop, and restart vsftpd after booting use the service command:

```
[root@172.16.16.15 tmp]# service vsftpd start
[root@172.16.16.15 tmp]# service vsftpd stop
[root@172.16.16.15 tmp]# service vsftpd restart
```

To determine whether vsftpd is running you can issue either of these two commands. The first will give a status message. The second will return the process ID numbers of the vsftpd daemons.

```
[root@172.16.16.15 tmp]# service vsftpd status
[root@172.16.16.15 tmp]# pgrep spam
```

Note: Remember to run the chkconfig command at least once to ensure vsftpd starts automatically on your next reboot.

Testing the Status of VSFTPD

You can always test whether the VSFTPD process is running by using the netstat -a command which lists all the TCP and UDP ports on which the server is listening for traffic. This example shows the expected output.

```
[root@172.16.16.15 root]# netstat -a | grep ftp
tcp        0        0        *:ftp         *:*        LISTEN
[root@172.16.16.15 root]#
```

If VSFTPD wasn't running, there would be no output at all.

The vsftpd.conf File

VSFTPD only reads the contents of its vsftpd.conf configuration file only when it starts, so you'll have to restart VSFTPD each time you edit the file in order for the changes to take effect. The file may be located in either the `/etc` or the `/etc/vsftpd` directories depending on your Linux distribution.

This file uses a number of default settings you need to know about.

- VSFTPD runs as an anonymous FTP server. Unless you want any remote user to log into to your default FTP directory using a username of anonymous and a password that's the same as their email address, I would suggest turning this off. The configuration file's anonymous_enable directive can be set to no to disable this feature. You'll also need to simultaneously enable local users to be able to log in by removing the comment symbol (#) before the local_enable instruction.
- If you enable anonymous FTP with VSFTPD, remember to define the root directory that visitors will visit. This is done with the anon_root directive.

```
anon_root=/opt/test
// Add this in to /etc/vsftpd/vsftpd.conf and  access through
ftp://172.16.16.15  by using gftp/ftp u can access using anonymous username
and password In both cases you will get access of /opt/test
```

- VSFTPD allows only anonymous FTP downloads to remote users, not uploads from them. This can be changed by modifying the anon_upload_enable directive shown later.
- VSFTPD doesn't allow anonymous users to create directories on your FTP server. You can change this by modifying the anon_mkdir_write_enable directive.
- VSFTPD logs FTP access to the /var/log/vsftpd.log log file. You can change this by modifying the xferlog_file directive.
- By default VSFTPD expects files for anonymous FTP to be placed in the /var/ftp directory. You can change this by modifying the anon_root directive. There is always the risk with anonymous FTP that users will discover a way to write files to your anonymous FTP directory. You run the risk of filling up your /var partition if you use the default setting. It is best to make the anonymous FTP directory reside in its own dedicated partition.

The configuration file is fairly straight forward as you can see in the snippet below where we enable anonymous FTP and individual accounts simultaneously.

```
# Allow anonymous FTP?
anonymous_enable=YES
...
# The directory which vsftpd will try to change
# into after an anonymous login. (Default = /var/ftp)
anon_root=/data/directory
...
# Uncomment this to allow local users to log in.
local_enable=YES
...
# Uncomment this to enable any form of FTP write command.
# (Needed even if you want local users to be able to upload files)
write_enable=YES
...
# Uncomment to allow the anonymous FTP user to upload files. This only
# has an effect if global write enable is activated. Also, you will
# obviously need to create a directory writable by the FTP user.
#anon_upload_enable=YES
```

```
...
# Uncomment this if you want the anonymous FTP user to be able to create
# new directories.
#anon_mkdir_write_enable=YES
...
# Activate logging of uploads/downloads.
xferlog_enable=YES
...
# You may override where the log file goes if you like.
# The default is shown below.
xferlog_file=/var/log/vsftpd.log
...
```

To activate or deactivate a feature, remove or add the # at the beginning of the appropriate line.

Other vsftpd.conf Options

There are many other options you can add to this file:

- Limiting the maximum number of client connections (max_clients)
- Limiting the number of connections by source IP address (max_per_ip)
- The maximum rate of data transfer per anonymous login. (anon_max_rate)
- The maximum rate of data transfer per non-anonymous login. (local_max_rate)

Descriptions on this and more can be found in the vsftpd.conf man pages.

FTP Security Issues

FTP has a number of security drawbacks, but you can overcome them in some cases. You can restrict an individual Linux user's access to non-anonymous FTP, and you can change the configuration to not display the FTP server's software version information, but unfortunately, though very convenient, FTP logins and data transfers are not encrypted.

The /etc/vsftpd.ftpusers File

For added security, you may restrict FTP access to certain users by adding them to the list of users in the /etc/vsftpd.ftpusers file. The VSFTPD package creates this file with a number of entries for privileged users that normally shouldn't have FTP access. As FTP doesn't encrypt passwords, thereby increasing the risk of data or passwords being compromised, it is a good idea to let these entries remain and add new entries for additional security.

Anonymous Upload

If you want remote users to write data to your FTP server, then you should create a write-only directory within /var/ftp/pub. This will allow your users to upload but not access other files uploaded by other users. The commands you need are:

```
[root@172.16.16.15 tmp]# mkdir /var/ftp/pub/upload
[root@172.16.16.15 tmp]# chmod 722 /var/ftp/pub/upload
```

FTP Greeting Banner

Change the default greeting banner in the vsftpd.conf file to make it harder for malicious users to determine the type of system you have. The directive in this file is.

```
ftpd_banner = New Banner Here
```

Using SCP As Secure Alternative To FTP

One of the disadvantages of FTP is that it does not encrypt your username and password. This could make your user account vulnerable to an unauthorized attack from a person eavesdropping on the network connection. Secure Copy (SCP) and Secure FTP (SFTP) provide encryption and could be considered as an alternative to FTP for trusted users. SCP does not support anonymous services, however, a feature that FTP does support.

Troubleshooting FTP

You should always test your FTP installation by attempting to use an FTP client to log in to your FTP server to transfer sample files.

The most common sources of day-to-day failures are incorrect usernames and passwords.

Initial setup failures could be caused by firewalls along the path between the client and server blocking some or all types of FTP traffic. Typical symptoms of this are either connection timeouts or the ability to use the ls command to view the contents of a directory without the ability to either upload or download files. Follow the firewall rule guidelines to help overcome this problem.

FTP has many uses, one of which is allowing numerous unknown users to download files. You have to be careful, because you run the risk of accidentally allowing unknown persons to upload files to your server. This sort of unintended activity can quickly fill up your hard drive with illegal software, images, and music for the world to download, which in turn can clog your server's Internet access and drive up your bandwidth charges.

FTP Users with Only Read Access to a Shared Directory

In this example, anonymous FTP is not desired, but a group of trusted users need to have read only access to a directory for downloading files. Here are the steps:

1) Disable anonymous FTP. Comment out the anonymous_enable line in the vsftpd.conf file like this:

```
# Allow anonymous FTP?
anonymous_enable=NO
```

2) Enable individual logins by making sure you have the local_enable line uncommented in the vsftpd.conf file like this:

```
# Uncomment this to allow local users to log in.
local_enable=YES
```

3) Start VSFTP.

```
[root@172.16.16.15 tmp]# service vsftpd start
```

4) Create a user group and shared directory. In this case, use /home/ftp-users and a user group name of ftp-users for the remote users

```
[root@172.16.16.15 tmp]# groupadd ftp-users
[root@172.16.16.15 tmp]# mkdir /home/ftp-docs
```

5) Make the directory accessible to the ftp-users group.

```
[root@172.16.16.15 tmp]# chmod 750 /home/ftp-docs
[root@172.16.16.15 tmp]# chown root:ftp-users /home/ftp-docs
```

6) Add users, and make their default directory /home/ftp-docs

```
[root@172.16.16.15 tmp]# useradd -g ftp-users -d /home/ftp-docs user1
[root@172.16.16.15 tmp]# useradd -g ftp-users -d /home/ftp-docs user2
[root@172.16.16.15 tmp]# useradd -g ftp-users -d /home/ftp-docs user3
[root@172.16.16.15 tmp]# useradd -g ftp-users -d /home/ftp-docs user4
[root@172.16.16.15 tmp]# passwd user1
[root@172.16.16.15 tmp]# passwd user2
[root@172.16.16.15 tmp]# passwd user3
[root@172.16.16.15 tmp]# passwd user4
```

7) Copy files to be downloaded by your users into the /home/ftp-docs directory

8) Change the permissions of the files in the /home/ftp-docs directory for read only access by the group

```
[root@172.16.16.15 tmp]# chown root:ftp-users /home/ftp-docs/*
[root@172.16.16.15 tmp]# chmod 740 /home/ftp-docs/*
```

Users should now be able to log in via FTP to the server using their new usernames and passwords. If you absolutely don't want any FTP users to be able to write to any directory, then you should set the write_enable line in your vsftpd.conf file to no:

```
write_enable = NO
```

Remember, you must restart VSFTPD for the configuration file changes to take effect.

Sample Login Session to Test Functionality

Here is a simple test procedure you can use to make sure everything is working correctly:

1) Check for the presence of a test file on the ftp client server.

```
[root@smallfry tmp]# ll
total 1
-rw-r--r-- 1 root root 0 Jan 4 09:08 testfile
[root@smallfry tmp]#
```

2) Connect to 172.16.16.15 via FTP

```
[root@smallfry tmp]# ftp 172.16.16.15
Connected to 172.16.16.15 (172.16.16.15)
220 ready, dude (vsFTPd 1.1.0: beat me, break me)
Name (172.16.16.15:root): user1
331 Please specify the password.
Password:
230 Login successful. Have fun.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

As expected, we can't do an upload transfer of testfile to 172.16.16.15.

```
ftp> put testfile
local: testfile remote: testfile
227 Entering Passive Mode (172,16,16,15,181,210)
553 Could not create file.
ftp>
```

But we can view and download a copy of the VSFTPD RPM located on the FTP server 172.16.16.15.

```
ftp> ls
227 Entering Passive Mode (172,16,16,15,35,173)
150 Here comes the directory listing.
-rwxr----- 1 0 502 76288 Jan 04 17:06 vsftpd-1.1.0-1.i386.rpm
226 Directory send OK.
ftp> get vsftpd-1.1.0-1.i386.rpm vsftpd-1.1.0-1.i386.rpm.tmp
local: vsftpd-1.1.0-1.i386.rpm.tmp remote: vsftpd-1.1.0-1.i386.rpm
227 Entering Passive Mode (172,16,16,15,44,156)
150 Opening BINARY mode data connection for vsftpd-1.1.0-1.i386.rpm (76288
bytes).
226 File send OK.
76288 bytes received in 0.499 secs (1.5e+02 Kbytes/sec)
ftp> exit
221 Goodbye.
[root@smallfry tmp]#
```

As expected, anonymous FTP fails.

```
[root@smallfry tmp]# ftp 172.16.16.15
Connected to 172.16.16.15 (172.16.16.15)
220 ready, dude (vsFTPd 1.1.0: beat me, break me)
Name (172.16.16.15:root): anonymous
331 Please specify the password.
Password:
530 Login incorrect.
Login failed.
ftp> quit
221 Goodbye.
[root@smallfry tmp]#
```

Now that testing is complete, you can make this a regular part of your FTP server's operation.

FTP is a very useful software application that can have enormous benefit to a Web site or to collaborative computing in which files need to be shared between business partners. Although insecure, it is universally accessible, because FTP clients are a part of all operating systems and Web browsers.