UNIX Basics and Commands, PuTTY V1.0

Preface

This document will take you through UNIX architecture, directory structure and commands description.

Also it will give overview of PuTTY.

Table of Contents

1. What is UNIX?

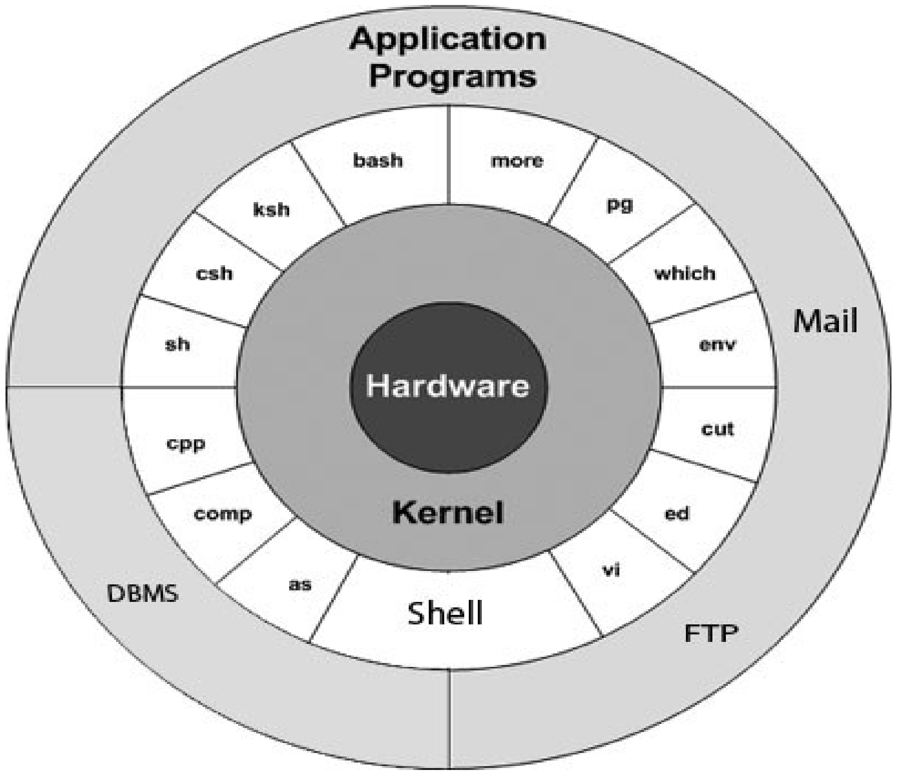
UNIX **(UNiplexed Information and Computing System)** is a powerful computer command line Operating System based on **"C "** language Originally developed in **1969** by **Ken Thompson, Dennis Ritchie** at **AT&T Bell Labs.** It is very popular among the scientific, engineering, and academic communities due to its multi-user and multi-tasking environment, flexibility and portability, electronic mail and networking capabilities, and the numerous programming, text processing and scientific utilities available.

Unix was one of the first "modern" operating systems developed for computers. Because of advantages in security, speed and cost, Unix serves as the operating system of choice for many companies around the world. Unix systems (and Unix-like variants such as FreeBSD, Solaris and Linux) maintain prevalence on large computers such as mainframes and supercomputers, while being equally suited for desktops, laptops and netbooks.

2. Features of UNIX

- Multi-User Operating System
- Multi-Tasking Operating System
- Portability
- Job Control
- Hierarchical Structure (Single Root)
- Case sensitive
- Device Independence
- Security

3. Unix Architecture



Kernal:
The kernel is the heart of the UNIX system which manages the system resources and it comunicates between hardware and software.It also handles the scheduling and execution of commands.

Shell:
It is a special program which provides you to interact with a UNIX system.
It gathers input from the user executes it and then displays output.

4. Types of Shells

- Bourne Shell (Default Prompt **#** Character)

- Bourne Shell (sh)
- Korn Shell (ksh)
- Bourne Again Shell (bash)
- Posix Shell (sh)

- C Shell (Default Prompt **%** Character)

- C Shell (csh)
- Tenex C Shell (tesh)

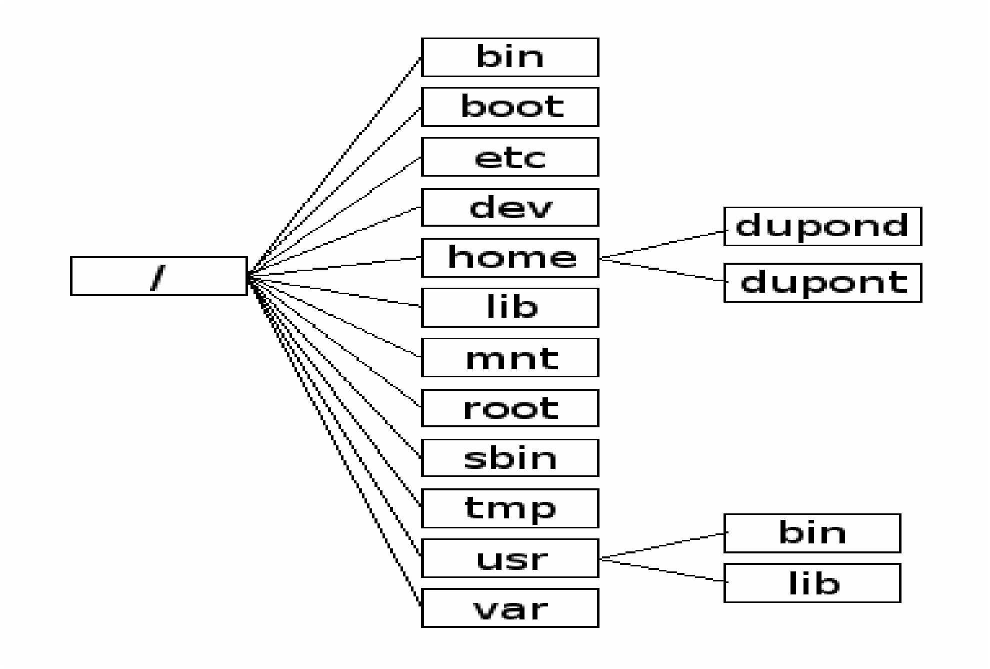
5. UNIX Directory Structure

Unix uses a hierarchical file system structure, much like an upside-down tree, with root (/) at the base of the file system and all other directories spreading from there.

A UNIX filesystem is a collection of files and directories that has the following properties:

- It has a root directory (/) that contains other files and directories.
- Each file or directory is uniquely identified by its name, the directory in which it resides, and a unique identifier, typically called an inode.
- By convention, the root directory has an inode number of 2 and the lost+found directory has an inode number of 3. Inode numbers 0 and 1 are not used. File inode numbers can be seen by specifying the -i option to ls command.
- It is self contained. There are no dependencies between one filesystem and any other.

The directories have specific purposes and generally hold the same types of information for easily locating files. Following are the directories that exist on the major versions of Unix:



/	This is the root directory which should contain only the directories needed at the top level of the file structure.
/bin	This is where the executable files are located. They are available to all user.
/dev	These are device drivers.
/etc	Supervisor directory commands, configuration files, disk configuration files, valid user lists, groups, Ethernet, hosts, where to send critical messages.
/lib	Contains shared library files and sometimes other kernel-related files.
/boot	Contains files for booting the system.
/home	Contains the home directory for users and other accounts.
/mnt	Used to mount other temporary file systems, such as cdrom and floppy for the CD-ROM drive and floppy diskette drive, respectively
/proc	Contains all processes marked as a file by process number or other information that is dynamic to the system.
/tmp	Holds temporary files used between system boots
/usr	Used for miscellaneous purposes, or can be used by many users. Includes administrative commands, shared files, library files, and others
/var	Typically contains variable-length files such as log and print files and any other type of file that may contain a variable amount of data
/sbin	Contains binary (executable) files, usually for system administration. For example fdisk and ifconfig utlities.

6. Unix Based Operating Systems

(Linux, HP-UX, MINIX, Sun Solaris, IBM AIX, FreeBSD)

7. PuTTY

PuTTY is a client program for the SSH, Telnet and Rlogin network protocols.

These protocols are all used to run a remote session on a computer, over a network. PuTTY implements the client end of that session: the end at which the session is displayed, rather than the end at which it runs.

In really simple terms: you run PuTTY on a Windows machine, and tell it to connect to (for example) a Unix machine. PuTTY opens a window. Then, anything you type into that window is sent straight to the Unix machine, and everything the Unix machine sends back is displayed in the window. So you can work on the Unix machine as if you were sitting at its console, while actually sitting somewhere else.

8. Types of Commands

- Simple : A simple command is one that you can execute by just giving its name at the prompt.

```
$ ls
```

- Complex : A compound command is a command that consists of a command name and a list of arguments (Arguments are command modifiers that change the behavior of a command).

```
$ ls -a
```

- Compound : One of the most powerful features of UNIX is the capability to combine simple and complex commands together to obtain compound commands.

A compound command consists of a list of simple and complex commands separated by the semicolon character (**;**).

```
$ date ; who am i ;
Wed Dec 9 10:10:10 PST 1998
miracle pts/0 Dec 9 08:49
```

9. Important Unix Commands

S.No	Commands Category	Commands
1	File Manipulation	ls, cat, mv, cp, rm, diff, nl, touch
2	Directory	cd, mkdir, mv, cp -r, rmdir, rm -r, pwd
3	File System & Administration	du, df, chown, free, chgrp, passwd, ln, chmod
4	Network/Communication	ping, ifconfig, set, env, uname, hostname, netstat, nslookup, ssh, scp, mail
5	Filters/Redirection	wc, more, head, tail, sort, |, tee
6	Archive/Compression	tar, gzip, gunzip, zcat, compress, uncompress
7	Process/Job Control	ps, kill, &, bg, fg, top, cron, at, ctrl + z, nice
8	Text Manipulation / Searching	grep, awk, sed, cut, vi, find
9	Information	who, w, who am i, man, whatis, finger, history, id, logname, tty
10	Miscellaneous	date, cal, exit, clear, banner
11	Shell Programming	read, expr, echo, sh, source, (.), sleep, test

10. Getting Help

Every version of UNIX comes with an extensive collection of online help pages called manual pages. These are often referred to as man pages .

```
$ man ls
```

11. File Manipulation Commands

All data in UNIX is organized into files. All files are organized into directories. These directories are organized into a tree-like structure called the filesystem.

When you work with UNIX, one way or another you spend most of your time working with files. This tutorial would teach you how to create and remove files, copy and rename them, create links to them etc.

In UNIX there are three basic types of files:

1. **Ordinary Files:** An ordinary file is a file on the system that contains data, text, or program instructions. In this tutorial, you look at working with ordinary files.
2. **Directories:** Directories store both special and ordinary files. For users familiar with Windows or Mac OS, UNIX directories are equivalent to folders.
3. **Special Files:** Some special files provide access to hardware such as hard drives, CD-ROM drives, modems, and Ethernet adapters. Other special files are similar to aliases or shortcuts and enable you to access a single file using different names.

11.1 Listing Files

To list the files and directories stored in the current directory. Use the following command:

```
$ ls
```

Here is the sample output of the above command:

```
$ ls

bin        hosts  lib     res.03
ch07       hw1    pub     test_results
ch07.bak   hw2    res.01  users
docs       hw3    res.02  work
```

The command ls supports the -l option which would help you to get more information about the listed files:

```
$ls -l
total 1962188

drwxrwxr-x  2 amrood amrood      4096 Dec 25 09:59 uml
-rw-rw-r--  1 amrood amrood      5341 Dec 25 08:38 uml.jpg
drwxr-xr-x  2 amrood amrood      4096 Feb 15  2006 univ
drwxr-xr-x  2 root   root        4096 Dec  9  2007 urlspedia
-rw-r--r--  1 root   root      276480 Dec  9  2007 urlspedia.tar
drwxr-xr-x  8 root   root        4096 Nov 25  2007 usr
drwxr-xr-x  2    200    300      4096 Nov 25  2007 webthumb-1.01
-rwxr-xr-x  1 root   root        3192 Nov 25  2007 webthumb.php
-rw-rw-r--  1 amrood amrood     20480 Nov 25  2007 webthumb.tar
-rw-rw-r--  1 amrood amrood      5654 Aug  9  2007 yourfile.mid
-rw-rw-r--  1 amrood amrood    166255 Aug  9  2007 yourfile.swf
drwxr-xr-x 11 amrood amrood      4096 May 29  2007 zlib-1.2.3
$
```

Here is the information about all the listed columns:

1. First Column: represents file type and permission given on the file. Below is the description

of all type of files.

2. Second Column: represents the number of memory blocks taken by the file or directory.
3. Third Column: represents owner of the file. This is the Unix user who created this file.
4. Fourth Column: represents group of the owner. Every Unix user would have an associated group.
5. Fifth Column: represents file size in bytes.
6. Sixth Column: represents date and time when this file was created or modified last time.
7. Seventh Column: represents file or directory name.

In the ls -l listing example, every file line began with a d, -, or l. These characters indicate the type of file that's listed.

Prefix	**Description**
-	Regular file, such as an ASCII text file, binary executable, or hard link.
b	Block special file. Block input/output device file such as a physical hard drive.
c	Character special file. Raw input/output device file such as a physical hard drive
d	Directory file that contains a listing of other files and directories.
l	Symbolic link file. Links on any regular file.
p	Named pipe. A mechanism for interprocess communications
s	Socket used for interprocess communication.

11.2 Viewing The Content Of A File

To view the content of a file we use ***cat*** command.

```
$ cat sample.txt
$ cat > sample.txt
$ cat >> sample.txt
```

11.3 Copying Files:

To make a copy of a file use the **cp** command. The basic syntax of the command is:

```
$ cp source_file destination_file
```

Following is the example to create a copy of existing file **filename**.

```
$ cp filename copyfile
```

Now you would find one more file **copyfile** in your current directory. This file would be exactly same as original file **filename**.

11.4 Renaming Files

To change the name of a file use the **mv** command. Its basic syntax is:

```
$ mv old_file new_file
```

Following is the example which would rename existing file **filename** to **newfile**:

```
$ mv filename newfile
```

The **mv** command would move existing file completely into new file. So in this case you would fine only **newfile** in your current directory.

11.5 Deleting Files:

To delete an existing file use the **rm** command. Its basic syntax is:

```
$ rm filename
```

Caution: It may be dangerous to delete a file because it may contain useful information. So be careful while using this command. It is recommended to use **-i** option along with **rm** command. Following is the example which would completely remove existing file **filename**:

```
$ rm filename
```

You can remove multiple files at a tile as follows:

```
$ rm filename1 filename2 filename3
```

11.6 Difference of Files

To compare the contents of two files use diff command:

```
diff email addresses
2a3,4
> Jean JRS@pollux.ucs.co
> Jim  jim@frolix8
```

This displays a line by line difference between the file email and addresses.

To make these files match you need to add (a) lines 3 and 4 (3,4) of the file addresses (>) after line 2 in the file email.

Here are the contents of files email and addresses used in this example. Line numbers are shown at the beginning of each line to help you follow this example.

```
email                               addresses
1 John erpl08@ed                    1 John erpl08@ed
2 Joe  CZT@cern.ch                  2 Joe  CZT@cern.ch
3 Kim  ks@x.co                      3 Jean JRS@pollux.ucs.co
4 Keith keith@festival             4 Jim  jim@frolix8
                                    5 Kim  ks@x.co
                                    6 Keith keith@festival
```

To compare two files ignoring differences in the case of the letters and blank spaces:

```
$ diff -iw part1 part_one
```

Lines in the two files that differ only in case and by the number of spaces they contain are considered to be identical.

11.7 Numbering Files

nl is used for numbering lines, either from a file or from standard input, reproducing output on standard output.

```
$ nl sort.txt
     1  UK
     2  Australia
     3  Newzealand
     4  Brazil
```

11.8 Updating Date of Modification

Every file in Linux is associated with timestamps, which specifies the last access time, last modification time and last change time.

Whenever we create a new file, or modify an existing file or its attributes, these timestamps will be updated automatically.

touch command is used to change these timestamps (access time, modification time, and change time of a file).

"touch" command options:

-a	change the access time only
-c	if the file does not exist, do not create it
-d	update the access and modification times
-m	change the modification time only
-r	use the access and modification times of file
-t	creates a file using a specified time

1. How to Create an Empty File

The following touch command creates an empty (zero byte) new file called **samp1**.

```
$ touch samp1
```

2. How to Create Multiple Files

By using touch command, you can also create more than one single file. For example the following command will create 3 files named, **samp1**, **samp2** and **samp3**.

```
$ touch samp1 samp2 samp3
```

3. How to Change File Access and Modification Time

To change or update the last access and modification times of a file called **samp3**, use the **-a** option as follows. The following command sets the current time and date on a file. If the **samp3** file does not exist, it will create the new empty file with the name.

```
$ touch -a samp3
```

The most popular Linux commands such as find command and ls command uses timestamps for listing and finding files.

4. How to Avoid Creating New File

Using **-c** option with touch command avoids creating new files. For example the following command will not create a file called **samp3** if it does not exists.

```
$ touch -c samp3
```

5. How to Change File Modification Time

If you like to change the only modification time of a file called **samp3**, then use the **-m** option with touch command. Please note it will only updates the last modification times (not the access times) of the file.

```
$ touch -m samp3
```

6. Explicitly Set the Access and Modification times

You can explicitly set the time using **-c** and **-t** option with touch command. The format would be as follows.

```
$ touch -c -t YYDDHHMM samp3
```

For example the following command sets the access and modification date and time to a file **samp3** as **17:30** (**17:30 p.m**.) **December 10** of the current year (**2012**).

```
$ touch -c -t 12101730 samp3
```

Next verify the access and modification time of file **samp3**, with **ls -l** command.

```
$ ls -l

total 2
-rw-r--r--.  1 root    root    0 Dec 10 17:30 samp3
```

7. How to Use the time stamp of another File

The following touch command with **-r** option, will update the time-stamp of file **samp3** with the time-stamp of **samp2** file. So, both the file holds the same time stamp.

```
$ touch -r samp3 samp2
```

8. Create a File using a specified time

If you would like to create a file with specified time other than the current time, then the format should be.

```
$ touch -t YYMMDDHHMM.SS tecmint
```

For example the below command touch command with **-t** option will gives the **tecmint** file a time stamp of **18:30:55 p.m**. on **December 10**, **2012**.

```
$ touch -t 201212101830.55 tecmint
```

12. Directory Commands

A directory is a file whose sole job is to store file names and related information. All files, whether ordinary, special, or directory, are contained in directories.

UNIX uses a hierarchical structure for organizing files and directories. This structure is often referred to as a directory tree . The tree has a single root node, the slash character (/), and all other directories are contained below it.

12.1 Home Directory:

The directory in which you find yourself when you first login is called your home directory.

You will be doing much of your work in your home directory and subdirectories that you'll be creating to organize your files.

You can go in your home directory anytime using the following command:

```
$ cd ~
```

Here **~** indicates home directory. If you want to go in any other user's home directory then use the following command:

```
$cd ~username
```

To go in your last directory you can use following command:

```
$cd -
```

12.2 Absolute/Relative Pathnames:

Directories are arranged in a hierarchy with root (/) at the top. The position of any file within the hierarchy is described by its pathname.

Elements of a pathname are separated by a /. A pathname is absolute if it is described in relation to root, so absolute pathnames always begin with a /.

These are some example of absolute filenames.

```
/etc/passwd
/users/sjones/chem/notes
/dev/rdsk/Os3
```

A pathname can also be relative to your current working directory. Relative pathnames never begin with /. Relative to user amrood' home directory, some pathnames might look like this:

```
chem/notes
personal/res
```

To determine where you are within the filesystem hierarchy at any time, enter the command **pwd** to print the current working directory:

```
$pwd
/user0/home/amrood
```

12.3 Listing Directories:

To list the files in a directory you can use the following syntax:

```
$ ls dirname
```

12.4 Creating Directories:

Directories are created by the following command:

```
$ mkdir dirname
```

Here, directory is the absolute or relative pathname of the directory you want to create. For example, the command:

```
$ mkdir mydir
```

Creates the directory mydir in the current directory. Here is another example:

```
$mkdir /tmp/test-dir
```

This command creates the directory test-dir in the /tmp directory. The **mkdir** command produces no output if it successfully creates the requested directory.

If you give more than one directory on the command line, mkdir creates each of the directories.

For example:

```
$ mkdir docs pub
```

Creates the directories docs and pub under the current directory.

Creating Parent Directories:

Sometimes when you want to create a directory, its parent directory or directories might not exist. In this case, mkdir issues an error message as follows:

```
$ mkdir /tmp/amrood/test
mkdir: Failed to make directory "/tmp/amrood/test";
No such file or directory
```

In such cases, you can specify the **-p** option to the **mkdir** command. It creates all the necessary directories for you. For example:

```
$ mkdir -p /tmp/amrood/test
```

Above command creates all the required parent directories.

12.5 Removing Directories:

Directories can be deleted using the **rmdir** command as follows:

```
$ rmdir dirname
```

Note: To remove a directory make sure it is empty which means there should not be any file or sub-directory inside this directory.

You can create multiple directories at a time as follows:

```
$ rmdir dirname1 dirname2 dirname3
```

Above command removes the directories dirname1, dirname2, and dirname2 if they are empty. The rmdir command produces no output if it is successful.

12.6 Changing Directories:

You can use the **cd** command to do more than change to a home directory: You can use it to change to any directory by specifying a valid absolute or relative path. The syntax is as follows:

```
$ cd dirname
```

Here, dirname is the name of the directory that you want to change to. For example, the command:

```
$ cd /usr/local/bin
```

Changes to the directory /usr/local/bin. From this directory you can cd to the directory /usr/home/amrood using the following relative path:

```
$ cd ../../home/amrood
```

12.7 Renaming Directories:

The mv (move) command can also be used to rename a directory. The syntax is as follows:

```
$mv olddir newdir
```

You can rename a directory **mydir** to **yourdir** as follows:

```
$ mv mydir yourdir
```

The directories . (dot) and .. (dot dot)

The filename . (dot) represents the current working directory; and the filename .. (dot dot) represent the directory one level above the current working directory, often referred to as the parent directory.
If we enter the command to show a listing of the current working directories files and use the -a option to list all the files and the -l option provides the long listing, this is the result.

```
$ls -la
drwxrwxr-x    4    teacher   class   2048  Jul 16 17.56 .
drwxr-xr-x    60   root              1536  Jul 13 14:18 ..
----------    1    teacher   class   4210  May 1 08:27 .profile
-rwxr-xr-x    1    teacher   class   1948  May 12 13:42 memo
```

12.8 Print Working Directory:

To find out what the current directory is, use the **pwd** (print working directory) command, which prints the name of the directory in which you are currently located. For example

```
$ pwd
/home/miracle/pub
```

13. File System and Administration Commands

13.1 Disk Usage:

The Linux "**du**" (**Disk Usage**) is a standard **Unix/Linux** command, used to check the information of disk usage of files and directories on a machine. The **du** command has many parameter options that can be used to get the results in many formats. The **du** command also displays the files and directory sizes in a recursively manner.

1. To find out the disk usage summary of a **/home/tecmint** directory tree and each of its sub directories. Enter the command as:

```
[root@tecmint]# du  /home/tecmint

40       /home/tecmint/downloads
4        /home/tecmint/.mozilla/plugins
4        /home/tecmint/.mozilla/extensions
12       /home/tecmint/.mozilla
12       /home/tecmint/.ssh
689112   /home/tecmint/Ubuntu-12.10
689360   /home/tecmint
```

The output of the above command displays the number of disk blocks in the **/home/tecmint** directory along with its sub-directories.

2. Using "**-h**" option with "**du**" command provides results in "**Human Readable Format**". Means

you can see sizes in **Bytes**, **Kilobyte**s, **Megabytes**, **Gigabytes** etc.

```
[root@tecmint]# du -h /home/tecmint

40K     /home/tecmint/downloads
4.0K    /home/tecmint/.mozilla/plugins
4.0K    /home/tecmint/.mozilla/extensions
12K     /home/tecmint/.mozilla
12K     /home/tecmint/.ssh
673M    /home/tecmint/Ubuntu-12.10
674M    /home/tecmint
```

```
[root@tecmint]# du -h /home/tecmint

40K     /home/tecmint/downloads
4.0K    /home/tecmint/.mozilla/plugins
4.0K    /home/tecmint/.mozilla/extensions
12K     /home/tecmint/.mozilla
12K     /home/tecmint/.ssh
673M    /home/tecmint/Ubuntu-12.10
674M    /home/tecmint
```

3. To get the summary of a grand total disk usage size of an directory use the option "**-s**" as follows.

```
[root@tecmint]# du -sh /home/tecmint

674M    /home/tecmint
```

4. Using "**-a**" flag with "**du**" command displays the disk usage of all the files and directories.

```
[root@tecmint]# du -a /home/tecmint

4       /home/tecmint/.bash_logout
12      /home/tecmint/downloads/uploadprogress-1.0.3.1.tgz
24      /home/tecmint/downloads/Phpfiles-org.tar.bz2
40      /home/tecmint/downloads
12      /home/tecmint/uploadprogress-1.0.3.1.tgz
4       /home/tecmint/.mozilla/plugins
4       /home/tecmint/.mozilla/extensions
12      /home/tecmint/.mozilla
4       /home/tecmint/.bashrc
689108  /home/tecmint/Ubuntu-12.10/ubuntu-12.10-server-i386.iso
689112  /home/tecmint/Ubuntu-12.10
689360  /home/tecmint
```

5. Using "**-a**" flag along with "**-h**" displays disk usage of all files and folders in human readeable

format. The below output is more easy to understand as it shows the files in **Kilobytes**, **Megabytes** etc.

```
[root@tecmint]# du -ah /home/tecmint

4.0K    /home/tecmint/.bash_logout

12K     /home/tecmint/downloads/uploadprogress-1.0.3.1.tgz
24K     /home/tecmint/downloads/Phpfiles-org.tar.bz2
40K     /home/tecmint/downloads
12K     /home/tecmint/uploadprogress-1.0.3.1.tgz
4.0K    /home/tecmint/.mozilla/plugins
4.0K    /home/tecmint/.mozilla/extensions
12K     /home/tecmint/.mozilla
4.0K    /home/tecmint/.bashrc
673M    /home/tecmint/Ubuntu-12.10/ubuntu-12.10-server-i386.iso
673M    /home/tecmint/Ubuntu-12.10
674M    /home/tecmint
```

6. Find out the disk usage of a directory tree with its subtress in **Kilobyte** blcoks. Use the "**-k**" (displays size in **1024** bytes units).

```
[root@tecmint]# du -k /home/tecmint
40      /home/tecmint/downloads
4       /home/tecmint/.mozilla/plugins
4       /home/tecmint/.mozilla/extensions
12      /home/tecmint/.mozilla
12      /home/tecmint/.ssh
689112  /home/tecmint/Ubuntu-12.10
689360  /home/tecmint
```

7. To get the summary of disk usage of directory tree along with its subtrees in **Megabytes** (**MB**) only. Use the option "**-mh**" as follows. The "**-m**" flag counts the blocks in **MB** units and "**-h**" stands for human readable format.

```
[root@tecmint]# du -mh /home/tecmint

40K     /home/tecmint/downloads
4.0K    /home/tecmint/.mozilla/plugins
4.0K    /home/tecmint/.mozilla/extensions
12K     /home/tecmint/.mozilla
12K     /home/tecmint/.ssh
673M    /home/tecmint/Ubuntu-12.10
674M    /home/tecmint
```

8. The "**-c**" flag provides a grand total usage disk space at the last line. If your directory taken **674MB** space, then the last last two line of the output would be.

```
[root@tecmint]# du -ch /home/tecmint

40K     /home/tecmint/downloads
4.0K    /home/tecmint/.mozilla/plugins
4.0K    /home/tecmint/.mozilla/extensions
```

```
12K      /home/tecmint/.mozilla
12K      /home/tecmint/.ssh
673M     /home/tecmint/Ubuntu-12.10
674M     /home/tecmint
674M     total
```

9. The below command calculates and displays the disk usage of all files and directories, but excludes the files that matches given pattern. The below command excludes the "**.txt**" files while calculating the total size of diretory. So, this way you can exclude any file formats by using flag "**--exclude**". See the output there is no **txt** files entry.

```
[root@tecmint]# du -ah --exclude="*.txt" /home/tecmint

4.0K     /home/tecmint/.bash_logout
12K      /home/tecmint/downloads/uploadprogress-1.0.3.1.tgz
24K      /home/tecmint/downloads/Phpfiles-org.tar.bz2
40K      /home/tecmint/downloads
12K      /home/tecmint/uploadprogress-1.0.3.1.tgz
4.0K     /home/tecmint/.bash_history
4.0K     /home/tecmint/.bash_profile
4.0K     /home/tecmint/.mozilla/plugins
4.0K     /home/tecmint/.mozilla/extensions
12K      /home/tecmint/.mozilla
4.0K     /home/tecmint/.bashrc
24K      /home/tecmint/Phpfiles-org.tar.bz2
4.0K     /home/tecmint/geoipupdate.sh
4.0K     /home/tecmint/.zshrc
120K     /home/tecmint/goaccess-0.4.2.tar.gz.1
673M     /home/tecmint/Ubuntu-12.10/ubuntu-12.10-server-i386.iso
673M     /home/tecmint/Ubuntu-12.10
674M     /home/tecmin
```

10. Display the disk usage based on modification of time, use the flag "**–time**" as shown below.

```
[root@tecmint]# du -ha --time /home/tecmint

4.0K     2012-10-12 22:32        /home/tecmint/.bash_logout
12K      2013-01-19 18:48        /home/tecmint/downloads/uploadprogress-
1.0.3.1.tgz
24K      2013-01-19 18:48        /home/tecmint/downloads/Phpfiles-org.tar.bz2
40K      2013-01-19 18:48        /home/tecmint/downloads
12K      2013-01-19 18:32        /home/tecmint/uploadprogress-1.0.3.1.tgz
4.0K     2012-10-13 00:11        /home/tecmint/.bash_history
4.0K     2012-10-12 22:32        /home/tecmint/.bash_profile
0        2013-01-19 18:32        /home/tecmint/xyz.txt
0        2013-01-19 18:32        /home/tecmint/abc.txt
4.0K     2012-10-12 22:32        /home/tecmint/.mozilla/plugins
4.0K     2012-10-12 22:32        /home/tecmint/.mozilla/extensions
12K      2012-10-12 22:32        /home/tecmint/.mozilla
4.0K     2012-10-12 22:32        /home/tecmint/.bashrc
24K      2013-01-19 18:32        /home/tecmint/Phpfiles-org.tar.bz2
4.0K     2013-01-19 18:32        /home/tecmint/geoipupdate.sh
4.0K     2012-10-12 22:32        /home/tecmint/.zshrc
120K     2013-01-19 18:32        /home/tecmint/goaccess-0.4.2.tar.gz.1
673M     2013-01-19 18:51        /home/tecmint/Ubuntu-12.10/ubuntu-12.10-server-
i386.iso
673M     2013-01-19 18:51        /home/tecmint/Ubuntu-12.10
674M     2013-01-19 18:52        /home/tecmint
```

13.2 Disk Free Space:

On the internet you will find plenty of tools for checking disk space utilization in Linux. However, Linux has a strong built in utility called '**df**'. The '**df**' command stand for "**disk filesystem**", it is used to get full summary of available and used disk space usage of file system on Linux system.

Using '**-h**' parameter with (**df -h**) will shows the file system disk space statistics in "**human readable**" format, means it gives the details in bytes, mega bytes and gigabyte.

13.2.1. Check File System Disk Space Usage:

The "**df**" command displays the information of device name, total blocks, total disk space, used disk space, available disk space and mount points on a file system.

```
[root@tecmint ~]# df

Filesystem            1K-blocks      Used Available Use% Mounted on
/dev/cciss/c0d0p2      78361192  23185840  51130588  32% /
/dev/cciss/c0d0p5      24797380  22273432   1243972  95% /home
/dev/cciss/c0d0p3      29753588  25503792   2713984  91% /data
/dev/cciss/c0d0p1        295561     21531    258770   8% /boot
tmpfs                    257476         0    257476   0% /dev/shm
```

2. Display Information of all File System Disk Space Usage.

The same as above, but it also displays information of dummy file systems along with all the file system disk usage and their memory utilization.

```
[root@tecmint ~]# df -a

Filesystem            1K-blocks      Used Available Use% Mounted on
/dev/cciss/c0d0p2      78361192  23186116  51130312  32% /
proc                          0         0         0   -  /proc
sysfs                         0         0         0   -  /sys
devpts                        0         0         0   -  /dev/pts
/dev/cciss/c0d0p5      24797380  22273432   1243972  95% /home
/dev/cciss/c0d0p3      29753588  25503792   2713984  91% /data
/dev/cciss/c0d0p1        295561     21531    258770   8% /boot
tmpfs                    257476         0    257476   0% /dev/shm
none                          0         0         0   -  /proc/sys/fs/binfmt_misc
sunrpc                        0         0         0   -  /var/lib/nfs/rpc_pipefs
```

3. Show Disk Space Usage in Human Readable Format

Have you noticed that above commands displays information in bytes, which is not readable yet all, because we are in a habit of reading the sizes in megabytes, gigabytes etc. as it makes very easy to understand and remember.

The **df** command provides an option to display sizes in **Human Readable** formats by using '-h' (prints the results in human readable format (e.g., 1K 2M 3G)).

```
[root@tecmint ~]# df -h

Filesystem             Size  Used Avail Use% Mounted on
/dev/cciss/c0d0p2       75G   23G   49G  32% /
/dev/cciss/c0d0p5       24G   22G  1.2G  95% /home
```

```
/dev/cciss/c0d0p3       29G   25G  2.6G  91% /data
/dev/cciss/c0d0p1      289M   22M  253M   8% /boot
tmpfs                  252M     0  252M   0% /dev/shm
```

4. Display Information of /home File System

To see the information of only device **/home** file system in human readable format use the following command.

```
[root@tecmint ~]# df -hT /home

Filesystem        Type    Size  Used Avail Use% Mounted on
/dev/cciss/c0d0p5 ext3     24G   22G  1.2G  95% /home
```

5. Display Information of File System in Bytes

To display all file system information and usage in **1024-byte** blocks, use the option '**-k**' (e.g. –block-size=1K) as follows.

```
[root@tecmint ~]# df -k

Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/cciss/c0d0p2      78361192  23187212  51129216  32% /
/dev/cciss/c0d0p5      24797380  22273432   1243972  95% /home
/dev/cciss/c0d0p3      29753588  25503792   2713984  91% /data
/dev/cciss/c0d0p1        295561     21531    258770   8% /boot
tmpfs                    257476         0    257476   0% /dev/shm
```

6. Display Information of File System in MB

To display information of all file system usage in **MB** (**Mega Byte**) use the option as '**-m**'.

```
[root@tecmint ~]# df -m

Filesystem           1M-blocks      Used Available Use% Mounted on
/dev/cciss/c0d0p2         76525     22644     49931  32% /
/dev/cciss/c0d0p5         24217     21752      1215  95% /home
/dev/cciss/c0d0p3         29057     24907      2651  91% /data
/dev/cciss/c0d0p1           289        22       253   8% /boot
tmpfs                       252         0       252   0% /dev/shm
```

7. Display Information of File System in GB

To display information of all file system statistics in **GB** (**Gigabyte**) use the option as '**df -h**'.

```
[root@tecmint ~]# df -h

Filesystem            Size  Used Avail Use% Mounted on
/dev/cciss/c0d0p2      75G   23G   49G  32% /
/dev/cciss/c0d0p5      24G   22G  1.2G  95% /home
/dev/cciss/c0d0p3      29G   25G  2.6G  91% /data
/dev/cciss/c0d0p1     289M   22M  253M   8% /boot
tmpfs                 252M     0  252M   0% /dev/shm
```

8. Display File System Inodes

Using '**-i**' switch will display the information of number of used inodes and their percentage for the file system.

```
[root@tecmint ~]# df -i

Filesystem             Inodes   IUsed   IFree IUse% Mounted on
/dev/cciss/c0d0p2    20230848  133143 20097705    1% /
/dev/cciss/c0d0p5    6403712  798613 5605099   13% /home
/dev/cciss/c0d0p3    7685440 1388241 6297199   19% /data
/dev/cciss/c0d0p1      76304      40   76264    1% /boot
tmpfs                  64369       1   64368    1% /dev/shm
```

9. Display File System Type

If you notice all the above commands output, you will see there is no file system type mentioned in the results. To check the file system type of your system use the option '**T**'. It will display file system type along with other information.

```
[root@tecmint ~]# df -T

Filesystem        Type   1K-blocks  Used      Available Use% Mounted on
/dev/cciss/c0d0p2 ext3    78361192  23188812  51127616  32%   /
/dev/cciss/c0d0p5 ext3    24797380  22273432  1243972   95%   /home
/dev/cciss/c0d0p3 ext3    29753588  25503792  2713984   91%   /data
/dev/cciss/c0d0p1 ext3    295561     21531    258770    8%    /boot
tmpfs             tmpfs   257476         0    257476    0%   /dev/shm
```

10. Include Certain File System Type

If you want to display certain file system type use the '**-t**' option. For example, the following command will only display **ext3** file system.

```
[root@tecmint ~]# df -t ext3

Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/cciss/c0d0p2     78361192  23190072  51126356  32% /
/dev/cciss/c0d0p5     24797380  22273432   1243972  95% /home
/dev/cciss/c0d0p3     29753588  25503792   2713984  91% /data
/dev/cciss/c0d0p1       295561     21531    258770   8% /boot
```

11. Exclude Certain File System Type

If you want to display file system type that doesn't belongs to **ext3** type use the option as '**-x**'. For example, the following command will only display other file systems types other than **ext3**.

```
[root@tecmint ~]# df -x ext3

Filesystem           1K-blocks      Used Available Use% Mounted on
tmpfs                   257476         0    257476   0% /dev/shm
```

12. Display Information of df Command.

Using '**–help**' switch will display a list of available option that are used with **df** command.

```
[root@tecmint ~]# df –help
Usage: df [OPTION]... [FILE]...
Show information about the file system on which each FILE resides,
or all file systems by default.

Mandatory arguments to long options are mandatory for short options too.
  -a, --all              include dummy file systems
  -B, --block-size=SIZE use SIZE-byte blocks
  -h, --human-readable  print sizes in human readable format (e.g., 1K 234M 2G)
  -H, --si               likewise, but use powers of 1000 not 1024
  -i, --inodes           list inode information instead of block usage
  -k                     like --block-size=1K
  -l, --local            limit listing to local file systems
      --no-sync          do not invoke sync before getting usage info (default)
  -P, --portability      use the POSIX output format
      --sync             invoke sync before getting usage info
  -t, --type=TYPE        limit listing to file systems of type TYPE
  -T, --print-type       print file system type
  -x, --exclude-type=TYPE   limit listing to file systems not of type TYPE
  -v                      (ignored)
      --help     display this help and exit
      --version  output version information and exit
```

SIZE may be (or may be an integer optionally followed by) one of following:

kB 1000, K 1024, MB 1000*1000, M 1024*1024, and so on for G, T, P, E, Z, Y.

Report bugs to <bug-coreutils@gnu.org>.

13.3 Changing File Permissions :

You can change file and directory permissions with the chmod command. The basic syntax is as follows:

chmod expression files

Here, expression is a statement of how to change the permissions. This expression can be of the following types: 1) Symbolic 2) Octal

The symbolic expression method uses letters to alter the permissions, and the octal expression method uses numbers. The numbers in the octal method are base-8 (octal) numbers ranging from 0 to 7.

Table-1: Basic Permissions

Letter	Permission	Definition
r	Read	The user can view the contents of the file.
w	Write	The user can alter the contents of the file.
x	Execute	The user can run the file, which is likely a program. For directories, the execute permission must be set in order for users to access the directory.

Symbolic Method :

The symbolic expression has the syntax of

(who)(action)(permissions)

Table : who

Letter	**Represents**
u	Owner
g	Group
o	Other
a	All

Table : actions

Symbol	**Represents**
+	Adding permissions to the file
-	Removing permission from the file
=	Explicitly set the file permissions

Table : permissions

Letter	Represents
r	Read
w	Write
x	Execute

```
$ chmod a=r *

$ chmod guo=r *

$ chmod go-w .profile
```

Octal Method :

By changing permissions with an octal expression, you can only explicitly set file permissions. This method uses a single number to assign the desired permission to each of the three categories of users (owner, group, and other).

The values of the individual permissions are the following:

Read permission has a value of 4

Write permission has a value of 2

Execute permission has a value of 1

```
$ chmod 444 .profile
```

13.4 Changing Ownership

The **chown** command changes the ownership of a file. The basic syntax is as follows:

chown options user:group files

```
$ chown miracle: /home/httpd/html/users/details
```

13.5 Changing Group

You can change group ownership of a file with the chgrp command. Its basic syntax is as follows:

```
$ chgrp options group files

$ chgrp authors /home/ranga/docs/ch5.doc
```

13.6 Checking Physical and Swap Memory

1. Display System Memory

Free command used to check the used and available space of **physical memory** and **swap memory** in **KB**. See the command in action below.

```
# free

             total       used       free     shared    buffers     cached
Mem:       1021628     912548     109080          0     120368     655548
-/+ buffers/cache:     136632     884996
Swap:      4194296          0    4194296
```

2. Display Memory in Bytes

Free command with option **-b**, display the size of memory in **Bytes**.

```
             total       used       free     shared    buffers     cached
Mem:    1046147072  934420480  111726592          0  123256832  671281152
-/+ buffers/cache:  139882496  906264576
Swap:   4294959104          0 4294959104
```

3. Display Memory in Kilo Bytes

Free command with option **-k**, display the size of memory in (**KB**) **Kilobytes**.

```
# free -k

             total       used       free     shared    buffers     cached
Mem:       1021628     912520     109108          0     120368     655548
-/+ buffers/cache:     136604     885024
Swap:      4194296          0    4194296
```

4. Display Memory in Megabytes

To see the size of the memory in (**MB**) **Megabytes** use option as **-m**.

```
# free -m

             total       used       free     shared    buffers     cached
Mem:           997        891        106          0        117        640
-/+ buffers/cache:        133        864
Swap:         4095          0       4095
```

5. Display Memory in Gigabytes

Using **-g** option with free command, would display the size of the memory in **GB**(**Gigabytes**).

```
# free -g
             total       used       free     shared    buffers     cached
Mem:             0          0          0          0          0          0
-/+ buffers/cache:          0          0
Swap:            3          0          3
```

6. Display Total Line

Free command with **-t** option, will list the total line at the end.

```
# free -t

            total       used       free     shared    buffers     cached
Mem:      1021628     912520     109108          0     120368     655548
-/+ buffers/cache:     136604     885024
Swap:     4194296          0    4194296
Total:  5215924 912520  4303404
```

7. Disable Display of Buffer Adjusted Line

By default the free command display "**buffer adjusted**" line, to disable this line use option as -o.

```
# free -o

            total       used       free     shared    buffers     cached
Mem:      1021628     912520     109108          0     120368     655548
Swap:     4194296          0    4194296
```

8. Dispaly Memory Status for Regular Intervals

The **-s** option with number, used to update free command at regular intervals. For example, the below command will update free command every **5 seconds**.

```
# free -s 5

             total       used       free     shared    buffers     cached
Mem:       1021628     912368     109260          0     120368     655548
-/+ buffers/cache:     136452     885176
Swap:     4194296        0   4194296
```

9. Show Low and High Memory Statistics

The **-l** switch displays detailed high and low memory size statistics.

```
# free -l
             total       used       free     shared    buffers     cached
Mem:       1021628     912368     109260          0     120368     655548
Low:        890036     789064     100972
High:       131592     123304       8288
-/+ buffers/cache:     136452     885176
Swap:      4194296          0    4194296
```

10. Check Free Version

The **-V** option, display free command version information.

```
# free -V
procps version 3.2.8
```

13.7 Changing Password

The Linux passwd command is used to change the password for a user account. A user can only change the password of his/her account but the superuser can change the password of any account. Besides changing password, this command can change other information like password validity etc.

1. Change password using passwd command

The passwd command can be used to simply used to change the password for an account.

Consider the example below :

```
$ passwd guest
Changing password for guest.
(current) UNIX password:
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

So we see that password was changed successfully.

2. Display the account status information using -S option

The status information related to an individual account can be retrieved by using -S with this command.

Consider the following example :

```
$ passwd -S  himanshu
himanshu P 03/12/2011 0 99999 7 -1
```

So we see that the status information for my account was displayed. The individual fields in the output as described in the man page are :

The first field is the users login name. The second field indicates if the user account has a locked password (L), has no password (NP), or has a usable password (P). The third field gives the date of the last password change. The next four fields are the minimum age, maximum age, warning period, and inactivity period for the password. These ages are expressed in days.

3. Delete an account password using the -d option

The passwd command can be used to delete an account passwd in order to make it password-less. This can be done using the -d option with this command.

Consider the following example :

```
$ sudo passwd -d guest
```

```
passwd: password expiry information changed.
```

So we see that the -d option deletes the password by changing the password expiry information.

4. Immediately expire the account password using -e option

This can be done using the -e option. Once this is done, user will be forced to change the password the next time they log in.

Consider the example below.

```
$ sudo passwd -e guest
passwd: password expiry information changed.
```

So we see that the expiry information was changed.

5. Perform actions quietly by using -q option

If we do not want the passwd command to output any extra information as output then we can choose to do so by using the -q option.

Consider the example below :

```
$ sudo passwd -eq guest
$
```

So we see that we used the -q option along with the expiry changer -e option. And when the command was run, there was no information produced in the output. So we see that -q made the passwd command to work quietly.

13.8 Creating Links

There are two types of links available in Linux — **Soft Link and Hard Link**.

Linux ln command is used to create either soft or hard links.

This article explains how to create soft link, how to create hard link, and various link tips and tricks with 10 practical examples.

```
$ ls -l
total 4
lrwxrwxrwx 1 chris chris 10 2010-09-17 23:40 file1 -> sample.txt
-rw-r--r-- 1 chris chris 22 2010-09-17 23:36 sample.txt
```

The 1st character in each and every line of the ls command output indicates one of the following file types. If the 1st character is l (lower case L), then it is a link file.

- regular file
- **l** link file
- **d** directory
- **p** pipe
- **c** character special device
- **b** block special device

1. What is Soft Link and Hard Link?

Soft Link

Linux OS recognizes the data part of this special file as a reference to another file path. The data in the original file can be accessed through the special file, which is called as Soft Link.

To create a soft link, do the following (ln command with -s option):

```
$ ln -s /full/path/of/original/file /full/path/of/soft/link/file
```

Hard Link

With Hard Link, more than one file name reference the same inode number. Once you create a directory, you would see the hidden directories "." and ".." . In this, "." directory is hard linked to the current directory and the ".." is hard linked to the parent directory.

When you use link files, it helps us to reduce the disk space by having single copy of the original file and ease the administration tasks as the modification in original file reflects in other places.

To create a hard link, do the following (ln command with no option):

```
$ ln /full/path/of/original/file /full/path/of/hard/link/file
```

2. Create Symbolic Link for File or Directory

Create a symbolic link for a File

The following examples creates a symbolic link library.so under /home/chris/lib, based on the library.so located under /home/chris/src/ directory.

```
$ cd /home/chris/lib

$ ln -s /home/chris/src/library.so library.so

$ ls -l library.so
lrwxrwxrwx  1 chris chris       21 2010-09-18 07:23 library.so ->
/home/chris/src/library.so
```

Create a symbolic link for a Directory

Just like file, you can create symbolic link for directories as shown below.

```
$ mkdir /home/chris/obj

$ cd tmp

$ ln -s /home/chris/obj objects

$ ls -l objects
lrwxrwxrwx 1 chris chris       6 2010-09-19 16:48 objects -> /home/chris/obj
```

Note: The inode of the original file/directory and the soft link should not be identical.

3. Create Hard Link for Files

The inode number for the hard linked files would be same. The hard link for files can be created as follows,

```
$ ln src_original.txt dst_link.txt

$ ls -i dst_link.txt
253564 dst_link.txt

$ ls -i src_original.txt
253564 src_original.txt
```

Note: Unix / Linux will not allow any user (even root) to create hard link for a directory.

4. Create Links Across Different Partitions

When you want to create the link across partitions, you are allowed to create only the symbolic links. Creating hard link across partitions is not allowed, as Unix can't create/maintain same inode numbers across partitions.

You would see the **"Invalid cross-device link"** error when you are trying to create a hard link file across partitions.

```
# mount /dev/sda5 /mnt

# cd /mnt

# ls
main.c Makefile

# ln Makefile /tmp/Makefile
ln: creating hard link `/tmp/Makefile' to `Makefile': Invalid cross-device link
```

And the symbolic link can be created in the same way as we did in the above.

5. Backup the Target Files If it Already Exists

When you create a new link (if another file exist already with the same name as the new link name), you can instruct ln command to take a backup of the original file before creating the new link using the –backup option as shown below.

```
$ ls
ex1.c  ex2.c

$ ln --backup -s ex1.c ex2.c

$ ls -lrt
total 8
-rw-r--r-- 1 chris chris 20 2010-09-19 16:57 ex1.c
-rw-r--r-- 1 chris chris 20 2010-09-19 16:57 ex2.c~
lrwxrwxrwx 1 chris chris  5 2010-09-19 17:02 ex2.c -> ex1.c
```

Note: If you don't want the backup and overwrite the existing file then use -f option.

6. Create Link Using "No-Deference" ln Command Option

While creating a new soft link, normally OS would de-reference the destination path before it creates the new soft link.

Sometimes you might not want ln command to create the new link, if the destination path is already a symbolic link that is pointing to a directory.

Following examples shows a normal way of creating soft link inside a directory.

```
$ cd ~

$ mkdir example

$ ln -s /etc/passwd example

$ cd example/

$ ls -l
total 0
lrwxrwxrwx 1 root root 16 2010-09-19 17:24 passwd -> /etc/passwd
```

In case the "example" directory in the above code-snippet is a symbolic link pointing to some other directory (for example second-dir), the ln command shown will still create the link under second-dir. If you don't want that to happen, use ln -n option as shown below.

```
$ cd ~

$ rm -rf example
$ mkdir second-dir

$ ln -s second-dir example

$ ln -n -s /etc/passwd example
ln: creating symbolic link `example': File exists
```

Note: In the above example, if you don't use the -n option, the link will be created under ~/second-dir directory.

7. Create Link for Multiple Files at the Same Time

In the following example, there are two directories — first-dir and second-dir. The directory first-dir contains couple of C program files. If you want to create soft links for these files in second-dir, you'll typically do it one by one. Instead, you can create soft list for multiple files together using -t option as shown below.

```
$ ls
first-dir second-dir

$ ls first-dir
ex1.c  ex2.c

$ cd second-dir

$ ln -s ../first-dir/*.c -t .

$ ls -l
total 0
lrwxrwxrwx 1 chris chris 14 2010-09-19 15:20 ex1.c -> ../first-dir/ex1.c
lrwxrwxrwx 1 chris chris 14 2010-09-19 15:20 ex2.c -> ../first-dir/ex2.c
```

Keep in mind that whenever you are creating link files with -t option, it is better to go into target directory and perform the link creation process. Otherwise, you would face the broken link files as shown below.

```
$ cd first-dir

$ ln -s *.c /home/chris/second-dir

$ cd /home/chris/second-dir
$ ls -l
total 0
lrwxrwxrwx 1 chris chris 5 2010-09-19 15:26 ex1.c -> ex1.c
lrwxrwxrwx 1 chris chris 5 2010-09-19 15:26 ex2.c -> ex2.c
```

Instead, you might also use actual path for source files to create the link properly.

8. Removing the Original File When a Soft Link is pointing to it

When the original file referred by a soft-link is deleted, the soft link will be broken as shown below.

```
$ ln -s file.txt /tmp/link

$ ls -l /tmp/link
lrwxrwxrwx 1 chris chris 9 2010-09-19 15:38 /tmp/link -> file1.txt
```

```
$ rm file.txt

$ ls -l /tmp/link
lrwxrwxrwx 1 chris chris 9 2010-09-19 15:38 /tmp/link -> file1.txt
```

9. Links Help You to Increase the Partition Size Virtually

Let us assume that you have two partitions – 5GB and 20GB. The first partition does not have too much free space available in it. If a program located on the first partition needs more space (For example, for it's log file), you can use some of the space from the second partition by creating a link for the log files as shown below.

Consider that partition1 is mounted on /, and partition2 is mounted to /mnt/. Let us assume that the logs that are located on partition1 is running out of space, and you've decided to move them to partition2. You can achieve this as shown below.

```
$ mkdir /mnt/logs

$ cd /logs

$ mv * /mnt/logs

$ cd /; rmdir logs

$ ln -s /mnt/logs logs
```

10. Removing the Hard Linked Files

When you delete a file that is hard linked, you would be still able to access the content of the file until you have the last file which is hard linked to it, as shown in the example below.

Create a sample file.

```
$ vim src_original.txt
Created this file to test the hard link.
```

Create a hard link to the sample file.

```
$ ln src_original.txt dst_link.txt
```

Delete the original file.

```
$ rm src_original.txt
```

You can still access the original file content by using the hard link you created.

```
$ cat dst_link.txt
Created this file to test the hard link.
```

14. Network/Communication Commands

When you work in a distributed environment then you need to communicate with remote users and you also need to access remote Unix machines.

There are several Unix utilities which are especially useful for users computing in a networked, distributed environment. This tutorial lists few of them:

14.1 The ping Utility:

The ping command sends an echo request to a host available on the network. Using this command you can check if your remote host is responding well or not.

The ping command is useful for the following:

- Tracking and isolating hardware and software problems.
- Determining the status of the network and various foreign hosts.
- Testing, measuring, and managing networks.

Syntax:

Following is the simple syntax to use **ping** command:

$ping hostname or ip-address

Above command would start printing a response after every second. To come out of the command you can terminate it by pressing CNTRL + C keys.

Example:

Following is the example to check the availability of a host available on the network:

```
$ping google.com
PING google.com (74.125.67.100) 56(84) bytes of data.
64 bytes from 74.125.67.100: icmp_seq=1 ttl=54 time=39.4 ms
64 bytes from 74.125.67.100: icmp_seq=2 ttl=54 time=39.9 ms
64 bytes from 74.125.67.100: icmp_seq=3 ttl=54 time=39.3 ms
64 bytes from 74.125.67.100: icmp_seq=4 ttl=54 time=39.1 ms
64 bytes from 74.125.67.100: icmp_seq=5 ttl=54 time=38.8 ms
--- google.com ping statistics ---
22 packets transmitted, 22 received, 0% packet loss, time 21017ms
rtt min/avg/max/mdev = 38.867/39.334/39.900/0.396 ms
$
If a host does not exist then it would behave something like this:
$ping giiiiiigle.com
ping: unknown host giiiiigle.com
$
```

14.2 The ifconfig Utility:

ifconfig in short "**interface configuration**" utility for system/network administration in **Unix/Linux** operating systems to configure, manage and query network interface parameters via command line interface or in a system configuration scripts.

The "**ifconfig**" command is used for displaying current network configuration information, setting up an ip address, netmask or broadcast address to an network interface, creating an alias for network interface, setting up hardware address and enable or disable network interfaces.

The "**ifconfig**" command with no arguments will display all the active interfaces details. The **ifconfig** command also used to check the assigned IP address of an server.

```
[root@tecmint ~]# ifconfig

eth0      Link encap:Ethernet  HWaddr 00:0B:CD:1C:18:5A
          inet addr:172.16.25.126  Bcast:172.16.25.63  Mask:255.255.255.224
          inet6 addr: fe80::20b:cdff:fe1c:185a/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2341604 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2217673 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:293460932 (279.8 MiB)  TX bytes:1042006549 (993.7 MiB)
          Interrupt:185 Memory:f7fe0000-f7ff0000

lo        Link encap:Local Loopback
```

```
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:5019066 errors:0 dropped:0 overruns:0 frame:0
          TX packets:5019066 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:2174522634 (2.0 GiB)  TX bytes:2174522634 (2.0 GiB)

tun0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-
00
          inet addr:10.1.1.1  P-t-P:10.1.1.2  Mask:255.255.255.255
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
```

14.3 The set Utility

Linux **set** command is use to **set values of shell options and positional parameters** or **unset values of shell options and positional parameters** on Linux system. The Linux **set** command is one of bash shell built in command. The command example below show how to get **set command manual** or help on Linux **set** command using shell command, this help show information on how to use the Linux **set** command, the **set** command options and the arguments that can be use with Linux **set command**.

14.4 The unset Utility

This command removes one or more variables. Each *name* is a variable name, specified in any of the ways acceptable to the **set** command. If a *name* refers to an element of an array then that element is removed without affecting the rest of the array. If a *name* consists of an array name with no parenthesized index, then the entire array is deleted. The **unset** command returns an empty string as result. An error occurs if any of the variables doesn't exist, and any variables after the non-existent one are not deleted.

14.6 The env Command

env is a shell command for Unix and Unix-like operating systems. It is used to either print a list of environment variables or run another utility in an altered environment without having to modify the currently existing environment. Using env, variables may be added or removed, and existing variables may be changed by assigning new values to them.

In practice, env has another common use. It is often used by shell scripts to launch the correct interpreter. In this usage, the environment is typically not changed.

14.7 The nslookup Command

nslookup is a network administration tool for querying the Domain Name System (DNS) to obtain domain name or IP address mapping or any other specific DNS record.

It is also used to troubleshoot DNS related problems. This article provides few examples on using the nslookup command.

nslookup can operate on both “Interactive mode” and “Non-Interactive mode”. Interactive mode allows the user to query the DNS-Server about various host, and domains. Non-Interactive mode allows the user to query the information for a host or domain. In this article, all the commands explained are “Non-Interactive mode”.

1. nslookup – Simple Example

nslookup followed by the domain name will display the "A Record" (IP Address) of the domain.

```
$ nslookup redhat.com

Server:           192.168.19.2
Address:    192.168.19.2#53

Non-authoritative answer:
Name: redhat.com
Address: 209.132.183.181
```

In the above output, server refers to the IP address of the DNS server. Then the below section provides the "A Record" (IP Address) of the domain "redhat.com".

14.8 The hostname Command

You can view your system name (hostname) or assign a new name to your system using hostname command.

1. View your hostname

hostname command without any argument will return the name of your system.

```
# hostname
ramesh-laptop
```

14.9 The netstat Command

Netstat command displays various network related information such as network connections, routing tables, interface statistics, masquerade connections, multicast memberships etc.,

14.10 The ssh Client Utility

With the ssh command you can log in to a remote computer and work on it remotely. That is, it enables you to connect to another computer on the Internet, that could be physically located anywhere in the world, and open a window on your local machine that lets you run programs interactively on the remote machine.

For example the following command would be used to login to a computer with the network id "comp.org.net":

ssh jdoe@comp.org.net

assuming "jdoe" is the username of the account we are tying log into on this machine. The system will ask you for the password before you get the prompt that allows you to run programs on the remote computer.

If the user name of the remote machine is the same as on the local machine you can omit the user name:

ssh comp.org.net

You can also use ssh to run a command on a remote machine without login in. For example,

ssh jdoe@comp.org.net ps

will execute the command "ps" on the computer "comp.org.net" and show the results in your local window.

14. 11 The scp Client Utility

scp: Secure copy (remote file copy program).

Usage:

```
scp source_file username@ipaddress:destination_path
```

15. Filter Commands

15.1 Counting Words in a File

You can use the **wc** command to get a count of the total number of lines, words, and characters contained in a file. Following is the simple example to see the information about above created file:

```
$ wc filename
2  19 103 filename
$
```

Here is the detail of all the four columns:

1. First Column: represents total number of lines in the file.
2. Second Column: represents total number of words in the file.
3. Third Column: represents total number of bytes in the file. This is actual size of the file.
4. Fourth Column: represents file name.

You can give multiple files at a time to get the information about those file. Here is simple syntax:

$ wc filename1 filename2 filename3

You can connect two commands together so that the output from one program becomes the input of the next program. Two or more commands connected in this way form a pipe.

To make a pipe, put a vertical bar (|) on the command line between two commands.

When a program takes its input from another program, performs some operation on that input, and writes the result to the standard output, it is referred to as a *filter*.

15.2 The sort Command:

The **sort** command arranges lines of text alphabetically or numerically. The example below sorts the lines in the food file:

```
$sort food
Afghani Cuisine
Bangkok Wok
Big Apple Deli
Isle of Java
Mandalay
Sushi and Sashimi
Sweet Tooth
Tio Pepe's Peppers
$
```

The **sort** command arranges lines of text alphabetically by default. There are many options that control the sorting:

Option	**Description**
-n	Sort numerically (example: 10 will sort after 2), ignore blanks and tabs.
-r	Reverse the order of sort.
-f	Sort upper- and lowercase together.
+x	Ignore first x fields when sorting.

More than two commands may be linked up into a pipe. Taking a previous pipe example using **grep**, we can further sort the files modified in August by order of size.
The following pipe consists of the commands **ls, grep,** and **sort**:

```
$ls -l | grep "Aug" | sort +4n
-rw-rw-r--  1 carol doc      1605 Aug 23 07:35 macros
-rw-rw-r--  1 john  doc      2488 Aug 15 10:51 intro
-rw-rw-rw-  1 john  doc      8515 Aug  6 15:30 ch07
-rw-rw-rw-  1 john  doc     11008 Aug  6 14:10 ch02
$
```

This pipe sorts all files in your directory modified in August by order of size, and prints them to the terminal screen. The sort option +4n skips four fields (fields are separated by blanks) then sorts the lines in numeric order.

15.3 The pg and more Commands:

A long output would normally zip by you on the screen, but if you run text through more or pg as a filter, the display stops after each screenful of text.

Let's assume that you have a long directory listing. To make it easier to read the sorted listing, pipe the output through **more** as follows:

```
$ls -l | grep "Aug" | sort +4n | more
-rw-rw-r--  1 carol doc      1605 Aug 23 07:35 macros
-rw-rw-r--  1 john  doc      2488 Aug 15 10:51 intro
-rw-rw-rw-  1 john  doc      8515 Aug  6 15:30 ch07
-rw-rw-r--  1 john  doc     14827 Aug  9 12:40 ch03
        .
        .
        .
-rw-rw-rw-  1 john  doc     16867 Aug  6 15:56 ch05
--More--(74%)
```

The screen will fill up with one screenful of text consisting of lines sorted by order of file size. At the bottom of the screen is the **more** prompt where you can type a command to move through the sorted text.

When you're done with this screen, you can use any of the commands listed in the discussion of the more program.

15.4 Pipe (|)

Pipes and Filters

The purpose of this lesson is to introduce you to the way that you can construct powerful Unix command lines by combining Unix commands.

Concepts

Unix commands alone are powerful, but when you combine them together, you can accomplish complex tasks with ease. The way you combine Unix commands is through using pipes and filters.

Using a Pipe

The symbol | is the Unix pipe symbol that is used on the command line. What it means is that the standard output of the command to the left of the pipe gets sent as standard input of the command to the right of the pipe. Note that this functions a lot like the > symbol used to redirect the standard output of a command to a file. However, the pipe is different because it is used to pass the output of a command to another command, not a file.

Here is an example:

```
$ cat apple.txt
core
worm seed
jewel
$ cat apple.txt | wc
            3       4      21
$
```

In this example, at the first shell prompt, I show the contents of the file apple.txt to you. In the next shell prompt, I use the cat command to display the contents of the apple.txt file, but I sent the display not to the screen, but through a pipe to the wc (word count) command. The wc command then does its job and counts the lines, words, and characters of what it got as input.

You can combine many commands with pipes on a single command line. Here's an example where I count the characters, words, and lines of the apple.txt file, then mail the results to nobody@december.com with the subject line "The count."

```
$ cat apple.txt | wc | mail -s "the count"
nobody@december.com
```

15.5 About tee

Reads from standard input, and writes to standard output and to files.

Syntax

```
tee [OPTION]... [FILE]...
```

Description

The **tee** command is named after the T-splitter in plumbing, which splits water into two directions and is shaped like an uppercase T.

tee copies data from standard input to each FILE, and also to standard output. In effect, **tee** duplicates its input, routing it to multiple outputs at once.

Examples

```
ls -1 *.txt | wc -l | tee count.txt
```

In the above example, the **ls** command lists all files in the current directory that have the filename extension **.txt**, one file per line; this output is piped to **wc**, which counts the lines and outputs the number; this output is piped to **tee**, which writes the output to the terminal, and writes the same information to the file **count.txt**. If **count.txt** already exists, it is overwritten.

15.6 The head Command

The *head* command reads the first few lines of any text given to it as an input and writes them to *standard output* (which, by default, is the display screen).

head's basic syntax is:

head [options] [file(s)]

The square brackets indicate that the enclosed items are optional. By default, head returns the first ten lines of each file name that is provided to it.

For example, the following will display the first ten lines of the file named *aardvark* in the current directory (i.e., the directory in which the user is currently working):

head aardvark

If more than one input file is provided, head will return the first ten lines from each file, precede each set of lines by the name of the file and separate each set of lines by one vertical space. The following is an example of using head with two input files:

head aardvark armadillo

If it is desired to obtain some number of lines other than the default ten, the *-n* option can be used followed by an integer indicating the number of lines desired. For example, the above example could be modified to display the first 15 lines from each file:

head -n15 aardvark armadillo

15.7 About tail

tail outputs the last part, or "tail", of files.

Syntax

```
tail [OPTION]... [FILE]...
```

Description

tail prints the last 10 lines of each FILE to standard output. With more than one FILE, it precedes each set of output with a header giving the file name. If no FILE is specified, or if FILE is specified as a dash ("**-**"), **tail** reads from standard input.

Examples

```
tail myfile.txt
```

Outputs the last 10 lines of the file **myfile.txt**.

```
tail myfile.txt -n 100
```

Outputs the last **100** lines of the file **myfile.txt**.

```
tail -f myfile.txt
```

Outputs the last 10 lines of **myfile.txt**, and monitors **myfile.txt** for updates; **tail** then continues to output any new lines that are added to **myfile.txt**.

```
tail -f access.log | grep 24.10.160.10
```

This is a useful example of using **tail** and **grep** to selectively monitor a log file in real time.

In this command, **tail** monitors the file **access.log**. It pipes **access.log**'s final ten lines, and any new lines added, to the **grep** utility. **grep** reads the output from **tail**, and outputs only those lines which contain the IP address **24.10.160.10**.

16. Archive/Compression Commands

16.1 About tar

The **tar** program is used to create, modify, and access files archived in the **tar** format.

Description

"**tar**" stands for *tape archive*. It is an archiving file format.

tar was originally developed in the early days of Unix for the purpose of backing up files to tape-based storage devices. It was later formalized as part of the POSIX standard, and today is used to collect, distribute, and archive files, while preserving file system attributes such as user and group permissions, access and modification dates, and directory structures.

This documentation covers the GNU version of **tar**, which is included with most modern variants of the Linux operating system.

Creating an archive using tar command

Creating an uncompressed tar archive using option cvf

This is the basic command to create a tar archive.

```
$ tar cvf archive_name.tar dirname/
```

In the above command:

- c – create a new archive
- v – verbosely list files which are processed.
- f – following is the archive file name

Creating a tar gzipped archive using option cvzf

The above tar cvf option, does not provide any compression. To use a gzip compression on the tar archive, use the z option as shown below.

```
$ tar cvzf archive_name.tar.gz dirname/
```

- z – filter the archive through gzip

Note: .tgz is same as .tar.gz

Note: I like to keep the 'cvf' (or tvf, or xvf) option unchanged for all archive creation (or view, or extract) and add additional option at the end, which is easier to remember. i.e cvf for archive creation, cvfz for compressed gzip archive creation, cvfj for compressed bzip2 archive creation etc., For this method to work properly, don't give – in front of the options.

Creating a bzipped tar archive using option cvjf

Create a bzip2 tar archive as shown below:

```
$ tar cvfj archive_name.tar.bz2 dirname/
```

- j – filter the archive through bzip2

gzip vs bzip2: bzip2 takes more time to compress and decompress than gzip. bzip2 archival size is less than gzip.

Note: .tbz and .tb2 is same as .tar.bz2

2. Extracting (untar) an archive using tar command

Extract a *.tar file using option xvf

Extract a tar file using option x as shown below:

```
$ tar xvf archive_name.tar
3166 apache       15    0 29444 6112 1524 S   6.6       2.4 0:00.79 httpd
```

- x – extract files from archive

Extract a gzipped tar archive (*.tar.gz) using option xvzf

Use the option z for uncompressing a gzip tar archive.

```
$ tar xvfz archive_name.tar.gz
```

Extracting a bzipped tar archive (*.tar.bz2) using option xvjf

Use the option j for uncompressing a bzip2 tar archive.

```
$ tar xvfj archive_name.tar.bz2
```

Note: In all the above commands v is optional, which lists the file being processed.

3. Listing an archive using tar command

View the tar archive file content without extracting using option tvf

You can view the *.tar file content before extracting as shown below.

```
$ tar tvf archive_name.tar
```

View the *.tar.gz file content without extracting using option tvzf

You can view the *.tar.gz file content before extracting as shown below.

```
$ tar tvfz archive_name.tar.gz
```

View the *.tar.bz2 file content without extracting using option tvjf

You can view the *.tar.bz2 file content before extracting as shown below.

```
$ tar tvfj archive_name.tar.bz2
```

4. Listing out the tar file content with less command

When the number of files in an archive is more, you may pipe the output of tar to less. But, you can also use less command directly to view the tar archive output, as explained in one of our previous article Open & View 10 Different File Types with Linux Less Command — The Ultimate Power of Less.

16.2 About gzip, gunzip, and zcat

gzip, **gunzip**, and **zcat** are used to compress or expand files.

Syntax

```
gzip [ -acdfhlLnNrtvV19 ] [-S suffix] [ name ...  ]
gunzip [ -acfhlLnNrtvV ] [-S suffix] [ name ...  ]
zcat [ -fhLV ] [ name ...  ]
```

Description

gzip reduces the size of the named files using Lempel-Ziv coding (LZ77). Whenever possible, each file is replaced by one with the extension **.gz**, while keeping the same ownership modes, access and modification times. (The default extension is **-gz** for VMS, **z** for MSDOS, OS/2 FAT, Windows NT FAT and Atari.) If no files are specified, or if a file name is "**-**", the standard input is compressed to the standard output. **gzip** will only attempt to compress regular files. In particular, it will ignore symbolic links.

gunzip takes a list of files on its command line and replaces each file whose name ends with **.gz**, **-gz**, **.z**, **-z**, or **_z** (ignoring case) and which begins with the correct magic number with an uncompressed file without the original extension. **gunzip** also recognizes the special extensions **.tgz** and **.taz** as shorthands for **.tar.gz** and **.tar.Z** respectively. When compressing, **gzip** uses the **.tgz** extension if necessary instead of truncating a file with a **.tar** extension.

gunzip can currently decompress files created by **gzip**, zip, compress, **compress -H** or pack. The detection of the input format is automatic. When using the first two formats, **gunzip** checks a 32-bit CRC. For **pack**, **gunzip** checks the uncompressed length. The standard compress format was not designed to allow consistency checks. However **gunzip** is sometimes able to detect a bad **.Z** file. If you get an error when uncompressing a **.Z** file, do not assume that the **.Z** file is correct simply because the standard uncompress does not complain. This generally means that the standard uncompress does not check its input, and happily generates garbage output. The SCO **compress -H** format (lzh compression method) does not include a CRC but also allows some consistency checks.

zcat is identical to **gunzip -c**. (On some systems, **zcat** may be installed as **gzcat** to preserve the original link to **compress**.) **zcat** uncompresses either a list of files on the command line or its standard input and writes the uncompressed data on standard output. **zcat** will uncompress files that have the correct magic number whether they have a **.gz** suffix or not.

Examples

```
zcat myfiles.tar.gz | more
```

Prints the contents of all files within the gzipped tar archive **myfiles.tar.gz**, piping the output to the more command, which pages the output (pauses at the end of each page).

```
gzip backup.tar
```

Compresses the tar archive **backup.tar**, and renames it **backup.tar.gz**.

```
gunzip backup.tar.gz
```

Uncompresses the gzipped file **backup.tar.gz**, renaming it **backup.tar**.

```
gzip -r backupfolder
```

Recursively compresses all files in the folder **backupfolder**, and all files in any subdirectories of that folder, and gives them the extension **.gz**.

```
gunzip -r backupfolder
```

Recursively uncompresses all gzipped files in the folder **backupfolder**, and removes the extension **.gz** from the filenames.

```
gzip -c myfile.txt > myfile.txt.gz
```

Compresses the file **myfile.txt**, writing the compressed file data to standard output, which in this case is redirected to a file, **myfile.txt.gz**. The result is that the original file and the gzipped file will both exist after the command is run.

```
zcat myfile.txt.gz
```

Prints the uncompressed contents of the compressed file **myfile.txt.gz**. The output is identical to printing the contents of the uncompressed file with the command **cat myfile.txt**.

```
gunzip -c myfile.txt.gz > myfile.txt
```

Uncompresses the gzipped file **myfile.txt.gz** to standard output, which in this case is redirected to the file **myfile.txt**. If **myfile.txt** already exists, the shell will prompt you to overwrite it (or not).

16.3 About compress

compress compacts a file so that it becomes smaller. When compressing a file it will be replaced with a file with the extension **.Z**, while keeping all the same ownership modes.

Description

The **compress** utility reduces the size of files using adaptive Lempel-Ziv coding. Each file is renamed to the same name plus the extension **.Z**. A file argument with a **.Z** extension will be ignored except it will cause an error exit after other arguments are processed. If compression would not reduce the size of a file, the file is ignored.

As many of the modification time, access time, file flags, file mode, user ID, and group ID as allowed by permissions are retained in the new file.

If no files are specified or a file argument is a single dash ('**-**'), the standard input is compressed to the standard output. If either the input and output files are not regular files, the checks for reduction in size and file overwriting are not performed, the input file is not removed, and the attributes of the input file are not retained in the output file.

Examples

```
compress -v bigfile.exe
```

Compress **bigfile.exe** and rename that file to **bigfile.exe.Z**.

16.4 About uncompress

Uncompresses files that have been compressed using the **compress** command.

Syntax

```
uncompress [-cfv] [file...]
```

Description

The **uncompress** utility will restore files to their original state after they have been compressed using the **compress** utility. If no files are specified, the standard input will be uncompressed to the standard output.

Files compressed with **compress** typically have the extension **.Z**, and **uncompress** looks for and recognizes files with that extension as compressed files.

This utility supports the uncompressing of any files produced by **compress**. For files produced by **compress** on other systems, **uncompress** supports 9- to 16-bit compression (see **-b**, below).

Examples

```
uncompress myfile.txt.Z
```

Uncompress the file **myfile.txt.Z**.

Note that the **.Z** extension is assumed by default. If intead this command were given as

uncompress myfile.txt

...**uncompress** would still search for **myfile.txt.Z**, and uncompress it.

17. Process/Job Control Commands

17.1 ps command:

What does 'ps'mean?

ps is the shortage for Process Status. The command should be used to display the currently running processes on Unix/Linux systems. If you know the 'Task-Manager' which pops up under Windows NT/2000/XP when you press CTRL+ALT+DEL then you have a clue what ps does under Unix/Linux. Ps can show you the running processes on your system in different ways. I will describe the basic ones you should know.

Why is it good to know how the ps command works?

If you have a process which seems to hang (e.g. netscape navigator on some buggy websites) and you want to stop the process, then you can determine the process id of the process. Why do you need the process id? You can stop the process with the help of the 'kill' command. The kill command needs a process number otherwise it won't know to which process it should send the 'kill' signal. Other example. You have started a process and now your machines becomes slower and slower. You stop the process but your machine still gets slower and slower. Now it will be helpful if you can stop the process. In this case you also need its pid.
Next example: You want to know what processes your friend just uses (you know he has an remote session open), then you can also use the ps command to find out which processes belong to him.

When you execute a program on your UNIX system, the system creates a special environment for that program. This environment contains everything needed for the system to run the program as if no other program were running on the system.
Whenever you issue a command in UNIX, it creates, or starts, a new process. When you tried out the **ls** command to list directory contents, you started a process. A process, in simple terms, is an instance of a running program.
The operating system tracks processes through a five digit ID number known as the **pid** or process ID . Each process in the system has a unique pid.
Pids eventually repeat because all the possible numbers are used up and the next pid rolls or starts over. At any one time, no two processes with the same pid exist in the system because it is the pid that UNIX uses to track each process.
Starting a Process:
When you start a process (run a command), there are two ways you can run it:

- Foreground Processes
- Background Processes

Foreground Processes:
By default, every process that you start runs in the foreground. It gets its input from the keyboard and sends its output to the screen.
You can see this happen with the ls command. If I want to list all the files in my current directory, I can use the following command:

```
$ls ch*.doc
This would display all the files whose name start with ch and ends with .doc:
ch01-1.doc   ch010.doc  ch02.doc    ch03-2.doc
ch04-1.doc   ch040.doc  ch05.doc    ch06-2.doc
ch01-2.doc   ch02-1.doc
```

The process runs in the foreground, the output is directed to my screen, and if the ls command wants any input (which it does not), it waits for it from the keyboard.
While a program is running in foreground and taking much time, we cannot run any other

commands (start any other processes) because prompt would not be available until program finishes its processing and comes out.

Background Processes:

A background process runs without being connected to your keyboard. If the background process requires any keyboard input, it waits.
The advantage of running a process in the background is that you can run other commands; you do not have to wait until it completes to start another!
The simplest way to start a background process is to add an ampersand (&) at the end of the command.

```
$ls ch*.doc &
This would also display all the files whose name start with ch and ends with
.doc:
ch01-1.doc   ch010.doc  ch02.doc    ch03-2.doc
ch04-1.doc   ch040.doc  ch05.doc    ch06-2.doc
ch01-2.doc   ch02-1.doc
```

Here if the **ls** command wants any input (which it does not), it goes into a stop state until I move it into the foreground and give it the data from the keyboard.
That first line contains information about the background process - the job number and process ID. You need to know the job number to manipulate it between background and foreground.
If you press the Enter key now, you see the following:

```
[1]   +   Done                    ls ch*.doc &
$
```

The first line tells you that the **ls** command background process finishes successfully. The second is a prompt for another command.

Listing Running Processes:

It is easy to see your own processes by running the **ps** (process status) command as follows:

```
$ps
PID       TTY       TIME         CMD
18358     ttyp3     00:00:00     sh
18361     ttyp3     00:01:31     abiword
18789     ttyp3     00:00:00     ps
```

One of the most commonly used flags for ps is the **-f** (f for full) option, which provides more information as shown in the following example:

```
$ps -f
UID      PID  PPID C STIME    TTY   TIME CMD
amrood   6738 3662 0 10:23:03 pts/6 0:00 first_one
amrood   6739 3662 0 10:22:54 pts/6 0:00 second_one
amrood   3662 3657 0 08:10:53 pts/6 0:00 -ksh
amrood   6892 3662 4 10:51:50 pts/6 0:00 ps -f
```

Here is the description of all the fileds displayed by ps -f command:

Column	Description
UID	User ID that this process belongs to (the person running it).
PID	Process ID.
PPID	Parent process ID (the ID of the process that started it).

C	CPU utilization of process.
STIME	Process start time.
TTY	Terminal type associated with the process
TIME	CPU time taken by the process.
CMD	The command that started this process.

There are other options which can be used along with **ps** command:

Option	Description
-a	Shows information about all users
-x	Shows information about processes without terminals.
-u	Shows additional information like -f option.
-e	Display extended information.

17.2 Stopping Processes:

Ending a process can be done in several different ways. Often, from a console-based command, sending a CTRL + C keystroke (the default interrupt character) will exit the command. This works when process is running in foreground mode.
If a process is running in background mode then first you would need to get its Job ID using **ps** command and after that you can use **kill** command to kill the process as follows:

```
$ps -f
UID      PID  PPID C STIME    TTY   TIME CMD
amrood   6738 3662 0 10:23:03 pts/6 0:00 first_one
amrood   6739 3662 0 10:22:54 pts/6 0:00 second_one
amrood   3662 3657 0 08:10:53 pts/6 0:00 -ksh
amrood   6892 3662 4 10:51:50 pts/6 0:00 ps -f
$kill 6738
```

Terminated
Here **kill** command would terminate first_one process. If a process ignores a regular kill command, you can use **kill -9** followed by the process ID as follows:
$kill -9 6738
Terminated

17.3 top Command:

The top program provides a dynamic real-time view of a running system. It can display system summary information as well as a list of tasks currently being managed by the Linux kernel.
The top command monitors CPU utilization, process statistics, and memory utilization. The top section contains information related to overall system status - uptime, load average, process counts, CPU status, and utilization statistics for both memory and swap space.

The top command produces a frequently-updated list of processes. By default, the processes are ordered by percentage of CPU usage, with only the "top" CPU consumers shown. The top command shows how much processing power and memory are being used, as well as other information about the running processes.

```
PID   USER         PR    NI    VIRT  RES   SHR  S %CPU %MEM  TIME+        COMMAND
3166 apache        15    0 29444 6112 1524 S   6.6       2.4 0:00.79  httpd
```

PID – process ID of the process

USER – User who is running the process

PR – The priority of the process

NI – Nice value of the process (higher value indicates lower priority)

VIRT – The total amount of virtual memory used

RES – Resident task size

SHR – Amount of shared memory used

S – State of the task. Values are **S** (sleeping), **D** (uninterruptible sleep), **R** (running), **Z** (zombies), or **T** (stopped or traced)

%CPU – Percentage of CPU used

%MEM – Percentage of Memory used

TIME+ – Total CPU time used

COMMAND – Command issued

17.4 NICE

Every running process in Unix has a priority assigned to it.

You can change the process priority using nice and renice utility. Nice command will launch a process with an user defined scheduling priority. Renice command will modify the scheduling priority of a running process.

Linux Kernel schedules the process and allocates CPU time accordingly for each of them. But, when one of your process requires higher priority to get more CPU time, you can use nice and renice command as explained in this tutorial.

The process scheduling priority range is from -20 to 19. We call this as nice value.

A nice value of -20 represents highest priority, and a nice value of 19 represent least priority for a process.

By default when a process starts, it gets the default priority of 0.

1. Display Nice Value of a Process

The current priority of a process can be displayed using ps command.

The “NI” column in the ps command output indicates the current nice value (i.e priority) of a process.

We’ll launch a test program called test.pl which will be used to demonstrate nice and renice command. This test program will do certain tasks, and will be running for a while.

```
$ perl test.pl
```

If you execute ps command as shown below, you can notice that this test.pl program has the default nice value of 0 (look at the NI column in the following output).

```
$ ps -fl -C "perl test.pl"
F S UID    PID   PPID  C PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
0 R bala 6884  6424 99  80   0 -  1556 -      13:45 pts/3    00:05:54 perltest.pl
```

2. Launch a Program with Less Priority

Instead of launching the program with the default priority, you can use nice command to launch the process with a specific priority.

In this example, test.pl is launched with a nice value of 10.

```
$ nice -10 perl test.pl
```

17.5 Renice

NAME

renice - alter priority of running processes

SYNOPSIS

```
renice priority [[-p ] pid ... ] [[-g ] pgrp ... ] [[-u ] user ... ]
```

DESCRIPTION

Renice alters the scheduling priority of one or more running processes. The following *who* parameters are interpreted as process ID's, process group ID's, or user names. **Renice 'ing** a process group causes all processes in the process group to have their scheduling priority altered. **Renice 'ing** a user causes all processes owned by the user to have their scheduling priority altered. By default, the processes to be affected are specified by their process ID's.

18. Text Manipulation/Searching Commands

18.1 The grep Command:

The grep program searches a file or files for lines that have a certain pattern. The syntax is:

$grep pattern file(s)

The name "grep" derives from the ed (a UNIX line editor) command g/re/p which means "globally search for a regular expression and print all lines containing it."

A regular expression is either some plain text (a word, for example) and/or special characters used for pattern matching.

The simplest use of grep is to look for a pattern consisting of a single word. It can be used in a pipe so that only those lines of the input files containing a given string are sent to the standard output. If you don't give grep a filename to read, it reads its standard input; that's the way all filter programs work:

```
$ls -l | grep "Aug"
-rw-rw-rw-   1 john  doc     11008 Aug  6 14:10 ch02
-rw-rw-rw-   1 john  doc      8515 Aug  6 15:30 ch07
-rw-rw-r--   1 john  doc      2488 Aug 15 10:51 intro
-rw-rw-r--   1 carol doc      1605 Aug 23 07:35 macros
$
```

There are various options which you can use along with grep command:

Option	**Description**
-v	Print all lines that do not match pattern.
-n	Print the matched line and its line number.
-l	Print only the names of files with matching lines (letter "l")

-c Print only the count of matching lines.

-i Match either upper- or lowercase.

Next, let's use a regular expression that tells grep to find lines with "carol", followed by zero or more other characters abbreviated in a regular expression as ".*"), then followed by "Aug".
Here we are using *-i* option to have case insensitive search:

```
$ls -l | grep -i "carol.*aug"
-rw-rw-r--   1 carol doc      1605 Aug 23 07:35 macros
$
```

1. Search for the given string in a single file

The basic usage of grep command is to search for a specific string in the specified file as shown below.

Syntax:

```
grep "literal_string" filename
$ grep "this" demo_file
this line is the 1st lower case line in this file.
Two lines above this line is empty.
```

2. Checking for the given string in multiple files.

```
Syntax:
grep "string" FILE_PATTERN
```

This is also a basic usage of grep command. For this example, let us copy the demo_file to demo_file1. The grep output will also include the file name in front of the line that matched the specific pattern as shown below. When the Linux shell sees the meta character, it does the expansion and gives all the files as input to grep.

```
$ cp demo_file demo_file1

$ grep "this" demo_*
demo_file:this line is the 1st lower case line in this file.
demo_file:Two lines above this line is empty.
demo_file:And this is the last line.
demo_file1:this line is the 1st lower case line in this file.
demo_file1:Two lines above this line is empty.
demo_file1:And this is the last line.
```

3. Case insensitive search using grep -i

Syntax:

```
grep -i "string" FILE


grep
```

This is also a basic usage of the grep. This searches for the given string/pattern case insensitively. So it matches all the words such as “the”, “THE” and “The” case insensitively as shown below.

```
$ grep -i "the" demo_file
THIS LINE IS THE 1ST UPPER CASE LINE IN THIS FILE.
```

```
this line is the 1st lower case line in this file.
This Line Has All Its First Character Of The Word With Upper Case.
And this is the last line.
```

First create the following demo_file that will be used in the examples below to demonstrate grep command.

```
$ cat demo_file
THIS LINE IS THE 1ST UPPER CASE LINE IN THIS FILE.
this line is the 1st lower case line in this file.
This Line Has All Its First Character Of The Word With Upper Case.

Two lines above this line is empty.
And this is the last line.
```

1. Search for the given string in a single file

The basic usage of grep command is to search for a specific string in the specified file as shown below.

Syntax:

```
grep "literal_string" filename
$ grep "this" demo_file
this line is the 1st lower case line in this file.
Two lines above this line is empty.
```

2. Checking for the given string in multiple files.

```
Syntax:
grep "string" FILE_PATTERN
```

This is also a basic usage of grep command. For this example, let us copy the demo_file to demo_file1. The grep output will also include the file name in front of the line that matched the specific pattern as shown below. When the Linux shell sees the meta character, it does the expansion and gives all the files as input to grep.

```
$ cp demo_file demo_file1

$ grep "this" demo_*
demo_file:this line is the 1st lower case line in this file.
demo_file:Two lines above this line is empty.
demo_file:And this is the last line.
demo_file1:this line is the 1st lower case line in this file.
demo_file1:Two lines above this line is empty.
demo_file1:And this is the last line.
```

3. Case insensitive search using grep -i

```
Syntax:
grep -i "string" FILE
```

This is also a basic usage of the grep. This searches for the given string/pattern case insensitively. So it matches all the words such as “the”, “THE” and “The” case insensitively as shown below.

```
$ grep -i "the" demo_file
THIS LINE IS THE 1ST UPPER CASE LINE IN THIS FILE.
this line is the 1st lower case line in this file.
This Line Has All Its First Character Of The Word With Upper Case.
```

```
And this is the last line.
```

18.2 Awk Introduction and Printing Operations

Awk is a programming language which allows easy manipulation of structured data and the generation of formatted reports. Awk stands for the names of its authors "**A**ho, **W**einberger, and **K**ernighan"

The Awk is mostly used for pattern scanning and processing. It searches one or more files to see if they contain lines that matches with the specified patterns and then perform associated actions.

Some of the key features of Awk are:

- Awk views a text file as records and fields.
- Like common programming language, Awk has variables, conditionals and loops
- Awk has arithmetic and string operators.
- Awk can generate formatted reports

Awk reads from a file or from its standard input, and outputs to its standard output. Awk does not get along with non-text files.

Syntax:

```

awk '/search pattern1/ {Actions}
     /search pattern2/ {Actions}' file
```

In the above awk syntax:

- search pattern is a regular expression.
- Actions – statement(s) to be performed.
- several patterns and actions are possible in Awk.
- file – Input file.
- Single quotes around program is to avoid shell not to interpret any of its special characters.

Awk Working Methodology

- Awk reads the input files one line at a time.
- For each line, it matches with given pattern in the given order, if matches performs the corresponding action.
- If no pattern matches, no action will be performed.
- In the above syntax, either search pattern or action are optional, But not both.
- If the search pattern is not given, then Awk performs the given actions for each line of the input.
- If the action is not given, print all that lines that matches with the given patterns which is the default action.
- Empty braces with out any action does nothing. It wont perform default printing operation.
- Each statement in Actions should be delimited by semicolon.

Let us create employee.txt file which has the following content, which will be used in the examples mentioned below.

```
$cat employee.txt
100  Thomas  Manager    Sales       $5,000
```

```
200   Jason    Developer   Technology   $5,500
300   Sanjay   Sysadmin    Technology   $7,000
400   Nisha    Manager     Marketing    $9,500
500   Randy    DBA         Technology   $6,000
```

Awk Example 1. Default behavior of Awk

By default Awk prints every line from the file.

```
$ awk '{print;}' employee.txt
100   Thomas   Manager     Sales        $5,000
200   Jason    Developer   Technology   $5,500
300   Sanjay   Sysadmin    Technology   $7,000
400   Nisha    Manager     Marketing    $9,500
500   Randy    DBA         Technology   $6,000
```

In the above example pattern is not given. So the actions are applicable to all the lines. Action print with out any argument prints the whole line by default. So it prints all the lines of the file with out fail. Actions has to be enclosed with in the braces.

Awk Example 2. Print the lines which matches with the pattern.

```
$ awk '/Thomas/
> /Nisha/' employee.txt
100   Thomas   Manager     Sales        $5,000
400   Nisha    Manager     Marketing    $9,500
```

In the above example it prints all the line which matches with the 'Thomas' or 'Nisha'. It has two patterns. Awk accepts any number of patterns, but each set (patterns and its corresponding actions) has to be separated by newline.

18.3 Sed Command in Unix and Linux Examples

Sed is a Stream Editor used for modifying the files in unix (or linux). Whenever you want to make changes to the file automatically, sed comes in handy to do this. Most people never learn its power; they just simply use sed to replace text. You can do many things apart from replacing text with sed. Here I will describe the features of sed with examples.

Consider the below text file as an input.

```
>cat file.txt
unix is great os. unix is opensource. unix is free os.
learn operating system.
unixlinux which one you choose.
```

Sed Command Examples
1. Replacing or substituting string

Sed command is mostly used to replace the text in a file. The below simple sed command replaces the word "unix" with "linux" in the file.

```
>sed 's/unix/linux/' file.txt
linux is great os. unix is opensource. unix is free os.
learn operating system.
linuxlinux which one you choose.
```

Here the "s" specifies the substitution operation. The "/" are delimiters. The "unix" is the search pattern and the "linux" is the replacement string.

By default, the sed command replaces the first occurrence of the pattern in each line and it won't replace the second, third...occurrence in the line.

2. Replacing the nth occurrence of a pattern in a line.

Use the /1, /2 etc flags to replace the first, second occurrence of a pattern in a line. The below command replaces the second occurrence of the word "unix" with "linux" in a line.

```
>sed 's/unix/linux/2' file.txt
unix is great os. linux is opensource. unix is free os.
learn operating system.
unixlinux which one you choose.
```

3. Replacing all the occurrence of the pattern in a line.

The substitute flag /g (global replacement) specifies the sed command to replace all the occurrences of the string in the line.

```
>sed 's/unix/linux/g' file.txt
linux is great os. linux is opensource. linux is free os.
learn operating system.
linuxlinux which one you choose.
```

4. Replacing from nth occurrence to all occurrences in a line.

Use the combination of /1, /2 etc and /g to replace all the patterns from the nth occurrence of a pattern in a line. The following sed command replaces the third, fourth, fifth... "unix" word with "linux" word in a line.

```
>sed 's/unix/linux/3g' file.txt
unix is great os. unix is opensource. linux is free os.
learn operating system.
unixlinux which one you choose.
```

18.4 The cut Command

For most of the example, we'll be using the following test file.

```
$ cat test.txt
cat command for file oriented operations.
cp command for copy files or directories.
ls command to list out files and directories with its attributes.
```

1. Select Column of Characters

To extract only a desired column from a file use -c option. The following example displays 2nd character from each line of a file test.txt

```
$ cut -c2 test.txt
a
p
s
```

As seen above, the characters a, p, s are the second character from each line of the test.txt file.

2. Select Column of Characters using Range

Range of characters can also be extracted from a file by specifying start and end position delimited with -. The following example extracts first 3 characters of each line from a file called test.txt

```
$ cut -c1-3 test.txt
cat
```

```
cp
ls
```

3. Select Column of Characters using either Start or End Position

Either start position or end position can be passed to cut command with -c option.

The following specifies only the start position before the '-'. This example extracts from 3rd character to end of each line from test.txt file.

```
$ cut -c3- test.txt
t command for file oriented operations.
 command for copy files or directories.
 command to list out files and directories with its attributes.
```

The following specifies only the end position after the '-'. This example extracts 8 characters from the beginning of each line from test.txt file.

```
$ cut -c-8 test.txt
cat comm
cp comma
ls comma
```

The entire line would get printed when you don't specify a number before or after the '-' as shown below.

```
$ cut -c- test.txt
cat command for file oriented operations.
cp command for copy files or directories.
ls command to list out files and directories with its attributes.
```

4. Select a Specific Field from a File

Instead of selecting x number of characters, if you like to extract a whole field, you can combine option -f and -d. The option -f specifies which field you want to extract, and the option -d specifies what is the field delimiter that is used in the input file.

The following example displays only first field of each lines from /etc/passwd file using the field delimiter : (colon). In this case, the 1st field is the username. The file

```

$ cut -d':' -f1 /etc/passwd
root
daemon
bin
sys
sync
games
bala
```

5. Select Multiple Fields from a File

You can also extract more than one fields from a file or stdout. Below example displays username and home directory of users who has the login shell as "/bin/bash".

```
$ grep "/bin/bash" /etc/passwd | cut -d':' -f1,6
root:/root
bala:/home/bala
```

18.5 What is `vi`?

The default editor that comes with the UNIX operating system is called vi (**vi**sual editor). [Alternate editors for UNIX environments include pico and emacs, a product of GNU.]

The UNIX vi editor is a full screen editor and has two modes of operation:

8. *Command mode* commands which cause action to be taken on the file, and
9. *Insert mode* in which entered text is inserted into the file.

In the command mode, every character typed is a command that does something to the text file being edited; a character typed in the command mode may even cause the vi editor to enter the insert mode. In the insert mode, every character typed is added to the text in the file; pressing the <Esc> (*Escape*) key turns off the Insert mode.

While there are a number of vi commands, just a handful of these is usually sufficient for beginning vi users. To assist such users, this Web page contains a sampling of basic vi commands. The most basic and useful commands are marked with an asterisk (* or star) in the tables below. With practice, these commands should become automatic.

NOTE: Both UNIX and vi are **case-sensitive**. Be sure not to use a capital letter in place of a lowercase letter; the results will not be what you expect. To Get Into and Out Of vi

To Start vi

To use vi on a file, type in vi filename. If the file named filename exists, then the first page (or screen) of the file will be displayed; if the file does not exist, then an empty file and screen are created into which you may enter text.

*	**`vi filename`**	*edit `filename` starting at line 1*
	`vi -r filename`	*recover `filename` that was being edited when system crashed*

To Exit vi

Usually the new or modified file is saved when you leave vi. However, it is also possible to quit vi without saving the file.

Note: The cursor moves to bottom of screen whenever a colon (:) is typed. This type of command is completed by hitting the <Return> (or <Enter>) key.

*	**`:x<Return>`**	*quit `vi`, writing out modified file to file named in original invocation*
	`:wq<Return>`	*quit `vi`, writing out modified file to file named in original invocation*
	`:q<Return>`	*quit (or exit) `vi`*
*	**`:q!<Return>`**	*quit `vi` even though latest changes have not been saved for this `vi` call*

Moving the Cursor

Unlike many of the PC and MacIntosh editors, **the mouse does not move the cursor** within the vi editor screen (or window). You must use the the key commands listed below. On some UNIX platforms, the arrow keys may be used as well; however, since vi was designed with the Qwerty keyboard (containing no arrow keys) in mind, the arrow keys sometimes produce strange effects in vi and should be avoided.

If you go back and forth between a PC environment and a UNIX environment, you may find that this dissimilarity in methods for cursor movement is the most frustrating difference between the two.

In the table below, the symbol ^ before a letter means that the <Ctrl> key should be held down while the letter key is pressed.

*	`j` ***or*** `<Return>` **[*or* down-arrow]**	*move cursor down one line*
*	`k` **[*or* up-arrow]**	*move cursor up one line*
*	`h` ***or*** `<Backspace>` **[*or* left-arrow]**	*move cursor left one character*
*	`l` ***or*** `<Space>` **[*or* right-arrow]**	*move cursor right one character*
*	`0` **(zero)**	*move cursor to start of current line (the one with the cursor)*
*	`$`	*move cursor to end of current line*
	`w`	*move cursor to beginning of next word*
	`b`	*move cursor back to beginning of preceding word*
	`:0<Return>` *or* `1G`	*move cursor to first line in file*
	`:n<Return>` *or* `nG`	*move cursor to line n*
	`:$<Return>` *or* `G`	*move cursor to last line in file*

Screen Manipulation

The following commands allow the `vi` editor screen (or window) to move up or down several lines and to be refreshed.

	`^f`	*move forward one screen*
	`^b`	*move backward one screen*
	`^d`	*move down (forward) one half screen*
	`^u`	*move up (back) one half screen*
	`^l`	*redraws the screen*
	`^r`	*redraws the screen, removing deleted lines*

Adding, Changing, and Deleting Text

Unlike PC editors, you cannot replace or delete text by highlighting it with the mouse. Instead use the commands in the following tables.

Perhaps the most important command is the one that allows you to back up and *undo* your last action. Unfortunately, this command acts like a toggle, undoing and redoing your most recent action. You cannot go back more than one step.

*	`u`	*UNDO WHATEVER YOU JUST DID; a simple toggle*

The main purpose of an editor is to create, add, or modify text for a file.

Inserting or Adding Text

The following commands allow you to insert and add text. Each of these commands puts the vi editor into insert mode; thus, the <Esc> key must be pressed to terminate the entry of text and to put the vi editor back into command mode.

*	`i`	*insert text before cursor, until* `<Esc>` *hit*
	`I`	*insert text at beginning of current line, until* `<Esc>` *hit*
*	`a`	*append text after cursor, until* `<Esc>` *hit*
	`A`	*append text to end of current line, until* `<Esc>` *hit*
*	`o`	*open and put text in a new line below current line, until* `<Esc>` *hit*
*	`O`	*open and put text in a new line above current line, until* `<Esc>` *hit*

18.6 The find Command

The find command is used to locate files on a Unix or Linux system. find will search any set of directories you specify for files that match the supplied *search criteria*. You can search for files by name, owner, group, type, permissions, date, and other criteria. The search is recursive in that it will search all subdirectories too. The syntax looks like this:

```
find where-to-look criteria what-to-do
```

All arguments to find are optional, and there are defaults for all parts. (This may depend on which version of find is used. Here we discuss the freely available Gnu version of find, which is the version available on YborStudent.) For example, *where-to-look* defaults to . (that is, the current working directory), *criteria* defaults to none (that is, select all files), and *what-to-do* (known as the find *action*) defaults to -print (that is, display the names of found files to standard output). Technically, the criteria and actions are all known as find *primaries*.

For example:

```
Find
```

will display the pathnames of all files in the current directory and all subdirectories. The commands

```
Find . -print
find -print
find .
```

do the exact same thing. Here's an example find command using a search criterion and the default action:

```
find / -name foo
```

This will search the whole system for any files named foo and display their pathnames. Here we are using the criterion -name with the argument foo to tell find to perform a name search for the filename foo. The output might look like this:

```
/home/wpollock/foo
/home/ua02/foo
/tmp/foo
```

If find doesn't locate any matching files, it produces no output.

The above example said to search the whole system, by specifying the root directory (“/”) to search. If you don't run this command as root, find will display a error message for each directory on which you don't have read permission. This can be a lot of messages, and the matching files that are found may scroll right off your screen. A good way to deal with this problem is to redirect the error messages so you don't have to see them at all:

```
find / -name foo 2>/dev/nul
```

19. Information Commands & Miscellaneous Commands

19.1 About cal

Calendar for the month and the year.

Syntax

```
Cal [options][[[day]month]year]
```

Description

Cal originally appeared in version 6 of AT&T Unix. Since then there have been versions released for BSD, Linux, and other Unix variants. You should check your particular installation's manual for version-specific options. Listed below are the traditional syntax and options for Unix **cal**.

In general, if no options are given, **cal** displays the current month at the command line. It's a quick and convenient way to glance at the dates of the month, and can be useful as part of a login script.

-1	Display a single month. This is the default.
-3	Display three months: last month, this month, and next month.
-s	Display the calendar using Sunday as the first day of the week.

Examples:

```
cal
```

Displays the calendar for this month.

```
Cal 12 2000
```

Displays the calendar for December of the year 2000.

19.2 Date

The date command under UNIX displays date and time. You can use the same command set date and time. You must be the super-user (root) to change the date and time on Unix like operating systems. The date command shows the date and time read from the kernel clock.

UNIX Date Command Syntax

The syntax is:

```
Date
date "+format"
```

Task: Display Current Date and Time

Type the following command:

```
date
```

Sample outputs:

```
Tue Oct 27 15:35:08 CDT 2009
```

When executed without arguments, the date command shows the current date and time.
Task: Set The Current Time

To set the current time to 05:30:30, enter:

```
Date 0530.30
```

Task: Set Date

Set the date to Oct 25, 12:45 a.m., enter:

```
Date 10250045
```

19.3 exit

About exit

Exit the shell.

Syntax

```
exit
```

Issuing the **exit** command at the shell prompt will cause the shell to exit.

In some cases, if you have jobs running in the background, the shell will remind you that they are running and simply return you to the command prompt. In this case, issuing **exit** again will terminate those jobs and exit the shell.

Common aliases for **exit** include "**bye**", "**logout**", and "**lo**".

19.4 About clear

Clears the screen.

Syntax:

```
CLEAR
```

Examples:

```
CLEAR
```

19.5 WHO

As a Linux user, sometimes it is required to know some basic information like :

- Time of last system boot
- List of users logged-in
- Current run level etc

Though this type of information can be obtained from various files in the Linux system but there is a command line utility 'who' that does exactly the same for you. In this article, we will discuss the capabilities and features provided by the 'who' command.

The basic syntax of the who command is :

```
Who [OPTION]...[ FILE | ARG1 ARG2 ]
```

Examples of 'who' command

1. Get the information on currently logged in users

This is done by simply running the 'who' command (without any options). Consider the following example:

```
$ who
himanshu  tty7                2012-08-07  05:33 (:0)
himanshu  pts/0               2012-08-07  06:47(:0.0)
himanshu  pts/1              2012-08-07   07:58(:0.0)
```

2. Get the time of last system boot

The is done using the -b option. Consider the following example:

```
$Who -b
System boot 2012-08-07 05:32
```

So we see that the above output gives the exact date and time of last system boot.

3. Get information on system login processes

This is done using the -l option. Consider the following example:

```
$ who -l
LOGIN    tty4        2012-08-07 05:32              1309 id=4
LOGIN    tty5        2012-08-07 05:32              1313 id=5
LOGIN    tty2        2012-08-07 05:32              1322 id=2
LOGIN    tty3        2012-08-07 05:32              1324 id=3
LOGIN    tty6        2012-08-07 05:32              1327 id=6
LOGIN    tty1        2012-08-07 05:32              1492 id=1
```

So we see that information related to system login processes was displayed in the output.

4. Get the hostname and user associated with stdin

This is done using the -m option. Consider the following example:

```
$who -m
himanshu pts1 2012-08-07 07:58 (:0.0)
```

So we see that the relevant information was produced in the output.

5. Get the current run level

This is done using the -r option. Consider the following example:

```

$ who -r
run-level 2 2012-08-07 05:32
```

So we see that the information related to current run level (which is 2) was produced in the output.

19.6 The w Command

w - show who is logged on and what they are doing

SYNOPSIS

w - [**-husfV**] [*user*]

DESCRIPTION

w displays information about the users currently on the machine, and their processes. The header shows, in this order, the current time, how long the system has been running, how many users are currently logged on, and the system load averages for the past 1, 5, and 15 minutes.

COMMAND-LINE OPTIONS

-h

Don't print the header.

-u

Ignores the username while figuring out the current process and cpu times. To demonstrate this, do a "su" and do a "w" and a "w -u".

-s

Use the short format. Don't print the login time, JCPU or PCPU times.

19.7 whoami

About whoami

whoami prints the effective user ID.

Syntax

```
Whoami [OPTION]
```

This command prints the username associated with the current effective user ID.

Running **whoami** is the same as running the **id** command with the options **-un**.

Options

--help	Display a help message, and exit.
--version	Display version information, and exit.

Examples

```
Whoami
```

Displays the name of the user who runs the command.

19.8 Man

About man

On Linux and other Unix-like operating systems, **man** is the interface used to view the system's reference manuals.

Description

man is the system's manual viewer; it can be used to display manual pages, scroll up and down, search for occurrences of specific text, and other useful functions.

Each argument given to **man** is normally the name of a program, utility or function. The manual page associated with each of these arguments is then found and displayed. A section number, if provided, will direct **man** to look only in that section of the manual. The default action is to search in all of the available sections, following a pre-defined order and to show only the first page found, even if page exists in several sections.

General Options

-h, **--help**	Print a help message and exit.
-V, **--version**	Display version information and exit.
-C *file*, **--config-file=***file*	Use configuration file *file* rather than the default of **~/.manpath**.
-d, **--debug**	Print debugging information.

Examples

```
man man
```

View the manual page for the **man** command.

```
man --nh --nj man
```

View the manual page for **man**, with no hyphenated words or justified lines.

19.9 About whatis

whatis displays short manual page descriptions.

Each manual page has a short description available within it. **whatis** searches the manual page names and displays the manual page descriptions of any name matched.

name may contain wildcards (**-w**) or be a regular expression (**-r**). Using these options, it may be necessary to quote the name or escape (\) the special characters to stop the shell from interpreting them.

index databases are used during the search, and are updated by the mandb program. Depending on your installation, this may be run by a periodic **cron** job, or may need to be run manually after new manual pages have been installed. To produce an old style text whatis database from the relative index database, issue the command:

Examples

```
whatis whatis
```

Display a description of what **whatis** is.

19.10 About finger

finger looks up and displays information about system users.

If no options are specified, **finger** defaults to the **-l** style output if operands are provided, otherwise to the **-s** style. Note that some fields may be missing, in either format, if information is not available for them.

If no arguments are specified, **finger** will print an entry for each user currently logged into the system.

Examples

```
finger -p ch
```

Display information about the user **ch**. Output will appear similar to the following:

```
Login name: admin
In real life: Computer Hope
On since Feb 11 23:37:16 on pts/7 from domain.computerhope.com
28 seconds Idle Time
Unread mail since Mon Feb 12 00:22:52 2001
```

19. 11 About fc and history

The **fc** command lists, edits, or re-executes commands previously entered to a shell. The **history** command allows you to use words from previous command lines in the command line you are typing. This simplifies spelling corrections and the repetition of complicated commands or arguments.

Syntax: history

history [-c] [-d *offset*] [*n*]

history -anrw [*filename*]

history -ps *arg* [*arg*...]

Description

Each shell (the Bourne shell, the Bourne Again Shell, the C Shell, the Korn Shell, etc.) has its own slight differences in how it handles, and allows access to, the command history. In general, the following commands will help you navigate and use your command history within your Linux/Unix shell.

history displays or manipulate the history list with line numbers, prefixing each modified entry with a '*****'. An argument of *n* lists only the last *b* entries.

history

Typing **history** alone would give results similar to the following:

```
2 grep --help
3 bg
4 fg
5 pine
6 cd public_html
7 rm index.html
8 sz index.html
9 ls -laxo
10 chmod 755 index.htm
```

!ls

Executes the most recently executed command that begins with the letters **ls**.

!!

Would re-execute the most recently executed command.

19.12 About id

Prints real and effective user and group IDs.

Syntax

```
id [OPTION]... [USERNAME]
```

Description

Print user and group information for the specified *USERNAME*, or (when *USERNAME* omitted) for the current user.

Examples

```
 Id
```

Reports user and group IDs similar to the example below:

```
 uid=12345(comphope) gid=12(users)
```

19.13 About logname

Prints the login name of the current user.

Syntax

```
logname [OPTION]
```

Options

--help	Display a help message, and exit.
--version	Display version information and exit.

Examples

```
Logname
```

Returns the name of the currently logged in user.

19. 14 About tty

NAME

tty - print the file name of the terminal connected to standard input

SYNOPSIS

tty [*OPTION*]...

DESCRIPTION

Print the file name of the terminal connected to standard input.

-s, **--silent**, **--quiet**
print nothing, only return an exit status
--help
display this help and exit
--version
output version information and exit

20. Environment

An important Unix concept is the **environment**, which is defined by environment variables. Some are set by the system, others by you, yet others by the shell, or any program that loads another program.

A variable is a character string to which we assign a value. The value assigned could be a number, text, filename, device, or any other type of data.
For example, first we set a variables TEST and then we access its value using **echo** command:

```
$TEST="Unix Programming"
$echo $TEST
```

Unix Programming

Note that environment variables are set without using $ sign but while accessing them we use $sign as prefix. These variables retain their values until we come out shell.

When you login to the system, the shell undergoes a phase called initialization to set up various environment. This is usually a two step process that involves the shell reading the following files:

- /etc/profile
- profile

The process is as follows:

1. The shell checks to see whether the file **/etc/profile** exists.
2. If it exists, the shell reads it. Otherwise, this file is skipped. No error message is displayed.
3. The shell checks to see whether the file **.profile** exists in your home directory. Your home directory is the directory that you start out in after you log in.
4. If it exists, the shell reads it; otherwise, the shell skips it. No error message is displayed.

As soon as both of these files have been read, the shell displays a prompt:

$

This is the prompt where you can enter commands in order to have them execute.

Note - The shell initialization process detailed here applies to all **Bourne** type shells, but some additional files are used by **bash** and **ksh**.

The .profile File:

The file **/etc/profile** is maintained by the system administrator of your UNIX machine and contains shell initialization information required by all users on a system.

The file **.profile** is under your control. You can add as much shell customization information as you want to this file. The minimum set of information that you need to configure includes

- The type of terminal you are using
- A list of directories in which to locate commands
- A list of variables effecting look and feel of your terminal.

You can check your **.profile** available in your home directory. Open it using **vi** editor and check all the variables set for your environment.

Setting the Terminal Type:

Usually the type of terminal you are using is automatically configured by either the **login** or **getty** programs. Sometimes, the autoconfiguration process guesses your terminal incorrectly.

If your terminal is set incorrectly, the output of commands might look strange, or you might not be able to interact with the shell properly.

To make sure that this is not the case, most users set their terminal to the lowest common denominator as follows:

```
$TERM=vt100
$
```

Setting the PATH:

When you type any command on command prompt, the shell has to locate the command before it can be executed.

The PATH variable specifies the locations in which the shell should look for commands. Usually it is set as follows:

```
$PATH=/bin:/usr/bin
$
```

Here each of the individual entries separated by the colon character, :, are directories. If you request the shell to execute a command and it cannot find it in any of the directories given in the PATH variable, a message similar to the following appears:

```
$hello
hello: not found
$
```

Environment Variables:

Following is the partial list of important environment variables. These variables would be set and accessed as mentioned above:

Variable	Description
DISPLAY	Contains the identifier for the display that X11 programs should use by default.
HOME	Indicates the home directory of the current user: the default argument for the cd built-in command.
IFS	Indicates the Internal Field Separator that is used by the parser for word splitting after expansion.
LANG	LANG expands to the default system locale; LC_ALL can be used to override this. For example, if its value is pt_BR, then the language is set to (Brazilian) Portuguese and the locale to Brazil.
LD_LIBRARY_PATH	On many Unix systems with a dynamic linker, contains a colon-separated list of directories that the dynamic linker should search for shared objects when building a process image after exec, before searching in any other directories.
PATH	Indicates search path for commands. It is a colon-separated list of directories in which the shell looks for commands.
PWD	Indicates the current working directory as set by the cd command.
RANDOM	Generates a random integer between 0 and 32,767 each time it is referenced.
SHLVL	Increments by one each time an instance of bash is started. This variable is useful for determining whether the built-in exit command ends the current

	session.
TERM	Refers to the display type
TZ	Refers to Time zone. It can take values like GMT, AST, etc.
UID	Expands to the numeric user ID of the current user, initialized at shell startup.

Following is the sample example showing few environment variables:

```
$ echo $HOME
/root
]$ echo $DISPLAY

$ echo $TERM
xterm
$ echo $PATH
/usr/local/bin:/bin:/usr/bin:/home/amrood/bin:/usr/local/bin
```

Environment Variables :

An environment variable is a variable that is available to any child process of the shell. Some programs need environment variables in order to function correctly.

EXPORTING :

Syntax :

```
name=value ; export name
export name=value
```

Example :

```
PATH=/sbin:/bin ; export PATH
export PATH=/sbin:/bin
```

21. Filename Substitution

It is sometimes referred to as globbing. Filename substitution is the process by which the shell expands a string containing wildcards into a list of filenames.

Wildcards Used in Filename Substitution

Wildcard	Description
*	Matches zero or more occurrences of any character
?	Matches one occurrence of any character
[*characters*]	Matches one occurrence of any of the given *characters*

21.1 Using the * Wildcard :

The simplest form of filename substitution is the * character. The * tells the shell to match zero or more occurrences of any character. If given by itself, it matches all filenames. For example, the command

Examples :

```
$ ls *
$ ls ch1*
$ ls *.doc
$ ls Backup*doc
```

21.2 Using the ? Wildcard :

One limitation of the * wildcard is that it matches one or more characters each time.

As an example, consider a situation where you need to list all files that have names of the form ch0*X*.doc, where *X* is a single number or letter. It seems like the command

```
$ ls ch0*.doc
```

would produce the appropriate match, but the actual output might look like:

```
ch01-1.doc ch010.doc ch02.doc ch03-2.doc ch04-1.doc ch040.doc
ch05.doc  ch06-2.doc  ch01-2.doc  ch02-1.doc  ch020.doc  ch03.doc  ch04-2.doc  ch05-
1.doc ch050.doc ch06.doc ch01.doc ch02-2.doc ch03-1.doc ch030.doc ch04.doc ch05-
2.doc ch06-1.doc ch060.doc
```

In order to match only one character, the shell provides you with the ? wildcard. You can rewrite the command using this wildcard:

```
$ ls ch0?.doc
```

Now you see that the output matches only those files you are interested in:

```
ch01.doc ch02.doc ch03.doc ch04.doc ch05.doc ch06.doc
```

Say that you now want to look for all files that have names of the form ch*XY*, where *X* and *Y* are any number or character. You can use the command

```
$ ls ch??.doc
```

21.3 Matching Sets of Characters :

Two potential problems with the ? and * wildcards are

- They match any character, including special characters such as hyphens (-) or underlines (_).
- You have no way to indicate that you want to match only letters or only numbers to these operators.

Sometimes you need more control over the exact characters that you match. Consider the situation where you want to match filenames of the form ch0*X*, where *X* is a number between 0 and 9. Neither the * or the ?
operator is cut out for this job.

Fortunately, the shell provides you with the capability to match sets of characters using the

[wildcard. The syntax for using this wildcard is

```
command [characters]
```

Here *command* is the name of a command, such as ls, and *characters* represents the characters you want to match. For example, the following command fulfills the previous requirements:

```
$ ls ch0[0123456789].doc
ch01.doc ch02.doc ch03.doc ch04.doc ch05.doc ch06.doc
```

One thing that you might have noticed is that you had to list all the characters that you wanted matched. The shell provides a mechanism to shorten the list. For example, the command

```
$ ls ch0[0-9].doc
```

produces the same list of files. As you can probably guess, this is most useful when you're trying to match sets of letters. For example,

```
$ ls [a-z]*
```

lists all the files starting with a lowercase letter. To match all the files starting with uppercase letters use the following:

```
$ ls [A-Z]*
```

The [wildcard also enables you to combine sets by putting the sets together. For example,

```
$ ls [a-zA-Z]*
```

matches all files that start with a letter, whereas the command

```
$ ls *[a-zA-Z0-9]
```

matches all files ending with a letter or a number.

As you can see from the previous examples, the maximum amount of flexibility in filename substitution occurs when you couple the [wildcard with the other wildcards.