CHAPTER 2

Simple Commands

In this chapter, we will introduce the reader to basic UNIX commands.

2.1 THE PASSWD COMMAND

The syntax of this command is :

passwd [account name]

This command allows users to change their passwords. This passwd command will ask you for your current password and then ask you for a new password. New passwords must be...

- At least 8 characters in length
- Must contain at least two letters
- Must contain at least two numbers and / or symbols like $ - + [ ] ^
- The password cannot have names, dates, or words that can be found in the dictionary.
- All passwords are case sensitive, meaning that it matters if you use a capital letter "A" or a lowercase letter "a" so make sure your caps lock key isn't on when you start typing in your password.

A sample session with password command.

```
$ passwd
Enter the new password (minimum of 5, maximum of 8 characters).
Please use a combination of upper and lower case letters and numbers.
New password:
Re-enter new password:
Password changed.
$
```

Important : People often opt to create basic, easily crackable passwords that use names, dates, or words that can be found in the dictionary. Doing so not only puts your account at risk, but endangers everyone. If you make a simple password, hackers could access your email, web page, assume your identity and possibly even get your financial information from email messages that you may have sent or received. Therefore, do not create silly, basic passwords. It WILL come back to haunt you, guaranteed!

Bad Passwords :

abc123
me
larry99
mycat
bobbyjoe
102868

Good Passwords :

2^jh[w3*
3EfW*a91
3-kou&A5
l@rrY99
m3ath3AD

Of course, you are expected remember the password. Windows and Mac's allow you to save your password so you don't even have to remember it if you don't want to. Do you know your social security number? How long is that compared to your 8 character password. How often do you use your social security number compared to your password? If you could remember some long strange sequence of numbers in SSN that you only use occasionally, you can easily remember your password also. Also, if you do opt to make a crackable password, your account may be shut off in some machines.

2.2 THE CD COMMAND

With the help of this command we can move from one directory to another directory. The syntax of this command is :

cd directoryname

Example:

cd /tmp	(It takes you to /tmp directory)
cd /usr/local	(It takes you to /usr/local directory)
cd ~	(In C shell it takes you to your home directory)
cd ~/rao	(In C shell, it takes you to rao directory of your home directory).
cd	(This command takes you to your home directory in bash shell)

We can specify the either absolute or relative path while specifying the directory name with this command.

2.3 THE MKDIR COMMAND

This command is used to create a new directory.

mkdir rao

This creates rao directory in the current directory.

mkdir /tmp/rao

This creates rao directory in /tmp directory.

mkdir /bin/rao

This command fails for normal user because of permissions (/bin belongs to super user). That is, normal user will not be having permissions to create files in /bin directory. Thus, we may get an error message "permission denied". Of course, if we work as a super user (administrator) then it works.

2.4 THE RMDIR COMMAND

This is used to remove empty directories.

rmdir rao

This removes rao directory of current working directory.

rmdir /tmp/rao

This removes rao directory in /tmp directory.

Do remember that we can not remember a directory having files. We can remove empty directories only. Of course, we can not remove other's directories also. You can remove your directories only. For example, as a normal user if you execute the following command, we will get an error message such as "permissions denied".

rmdir /mnt

Even if a directory belongs to you, you can not remove the same if you don't have permissions to do so. We shall discuss about the permissions later.

2.5 THE PWD COMMAND

This command displays where currently we are located. Usage of this command is :

pwd

2.6 COMMAND HISTORY

Most of the shells including bash support command history for convenience; *i.e.,* most recently executed commands are stored in history buffer which users can browse through any time without retyping the same. For example, the list of previous commands may be obtained by executing the following command.

history

!n

(n is an integer) will re-execute the nth command.

!!

This executes the most recent command.

!cp

This executes the most recent command which starts with cp.

history -2 lists the last two commands

history 3 5 Displays command numbers 3 to 5.

In some shells, up arrow, down arrow keys can be used to browse the commands in the history buffer.

In a nutshell the following can be used to manipulate commands in the history buffer

!*n*	Refer to command line *n*.
!-*n*	Refer to the command *n* lines back.
!!	Refer to the previous command. This is a synonym for '!-1'.
!*string*	Refer to the most recent command starting with *string*.
!?*string*[?]	Refer to the most recent command containing *string*. The trailing '?' may be omitted if the *string* is followed immediately by a newline.
!#	The entire command line typed so far.

^*string1*^*string2*^

Quick Substitution. Repeat the last command, replacing *string1* with *string2*. Equivalent to !!:s/*string1*/*string2*/.

Word designators are used to select desired words from the event. A ':' separates the event specification from the word designator. It may be omitted if the word designator begins with a '^', '$', '*', '-', or '%'. Words are numbered from the beginning of the line, with the first word being denoted by 0 (zero). Words are inserted into the current line separated by single spaces.

!!:$

designates the last argument of the preceding command. This may be shortened to !$.

!fi:2

designates the second argument of the most recent command starting with the letters fi.

Here are the word designators :

0 (zero)

The 0th word. For many applications, this is the command word.

n

The *n*th word.

^

The first argument; that is, word 1.

$

The last argument.

%

The word matched by the most recent '?*string*?' search.

x*-*y

A range of words; '-*y*' abbreviates '0-*y*'.

*****

All of the words, except the 0th. This is a synonym for '1-$'. It is not an error to use '*' if there is just one word in the event; the empty string is returned in that case.

x*

Abbreviates '*x*-$'

***x*-**

Abbreviates '*x*-$' like '*x**', but omits the last word.

If a word designator is supplied without an event specification, the previous command is used as the event.

After the optional word designator, you can add a sequence of one or more of the following modifiers, each preceded by a ':'.

h

Remove a trailing pathname component, leaving only the head.

t

Remove all leading pathname components, leaving the tail.

r

Remove a trailing suffix of the form '.*suffix*', leaving the basename.

e

Remove all but the trailing suffix.

p

Print the new command but do not execute it.

q

Quote the substituted words, escaping further substitutions.

x

Quote the substituted words as with 'q', but break into words at spaces, tabs, and newlines.

s/*old*/*new*/

Substitute *new* for the first occurrence of *old* in the event line. Any delimiter may be used in place of '/'. The delimiter may be quoted in *old* and *new* with a single backslash. If '&' appears in *new*, it is replaced by *old*. A single backslash will quote the '&'. The final delimiter is optional if it is the last character on the input line.

&

Repeat the previous substitution.

g

a

Cause changes to be applied over the entire event line. Used in conjunction with 's', as in gs/*old*/*new*/, or with '&'.

G

Apply the following 's' modifier once to each word in the event.

The value of the HISTSIZE shell variable is used as the number of commands to save in a history list. The text of the last $HISTSIZE commands (default 500) is saved. The shell stores each command in the history list prior to parameter and variable expansion but after history expansion is performed, subject to the values of the shell variables HISTIGNORE and HISTCONTROL. When the shell starts up, the history is initialized from the file named

by the HISTFILE variable (default **~/.bash_history**). The file named by the value of HISTFILE is truncated, if necessary, to contain no more than the number of lines specified by the value of the HISTFILESIZE variable. When an interactive shell exits, the last $HISTSIZE lines are copied from the history list to the file named by $HISTFILE. If the histappend shell option is set, the lines are appended to the history file, otherwise the history file is overwritten. If HISTFILE is unset, or if the history file is unwritable, the history is not saved. After saving the history, the history file is truncated to contain no more than $HISTFILESIZE lines. If HISTFILESIZE is not set, no truncation is performed.

If the HISTTIMEFORMAT is set, the time stamp information associated with each history entry is written to the history file, marked with the history comment character. When the history file is read, lines beginning with the history comment character followed immediately by a digit are interpreted as timestamps for the previous history line.

The built-in command fc may be used to list or edit and re-execute a portion of the history list. The history built-in may be used to display or modify the history list and manipulate the history file. When using command-line editing, search commands are available in each editing mode that provide access to the history list The shell allows control over which commands are saved on the history list. The HISTCONTROL and HISTIGNORE variables may be set to cause the shell to save only a subset of the commands entered. The cmdhist shell option, if enabled, causes the shell to attempt to save each line of a multi-line command in the same history entry, adding semicolons where necessary to preserve syntactic correctness. The lithist shell option causes the shell to save the command with embedded newlines instead of semicolons. The shopt builtin is used to set these options.

The following options can be used with history command.

-c

Clear the history list. This may be combined with the other options to replace the history list completely.

-d *offset*

Delete the history entry at position *offset*. *offset* should be specified as it appears when the history is displayed.

-a

Append the new history lines (history lines entered since the beginning of the current Bash session) to the history file.

-n

Append the history lines not already read from the history file to the current history list. These are lines appended to the history file since the beginning of the current Bash session.

-r

Read the current history file and append its contents to the history list.

-w

Write out the current history to the history file.

-p

Perform history substitution on the *arg*s and display the result on the standard output, without storing the results in the history list.

-s

The *args* are added to the end of the history list as a single entry.

When any of the -w, -r, -a, or -n options is used, if *filename* is given, then it is used as the history file. If not, then the value of the HISTFILE variable is used.

2.7 RECOMMENDED FILENAME CONVENTIONS

We should note here that a directory is merely a special type of file. So the rules and conventions for naming files apply also to directories.

In naming files, characters with special meanings such as **/ * & %** , should be avoided. Also, avoid using spaces within names. The safest way to name a file is to use only alphanumeric characters, that is, letters and numbers, together with _ (underscore) and . (dot).

Good filenames	***Bad filenames***
project.txt	project
my_big_program.c	my big program.c
venkat_naga.doc	venkat & naga.doc

File names conventionally start with a lower-case letter, and may end with a dot followed by a group of letters indicating the contents of the file. For example, all files consisting of C code may be named with the ending **.c**, for example, **prog1.c**. Then in order to list all files containing C code in your home directory, you need only type **ls *.c** in that directory.

2.8 THE CP COMMAND

cp command is used to duplicate a file(s).

Syntax

cp source destination

Example usages with explanation.

cp a1.c /tmp	*creates a1.c file in /tmp directory which contains same content as that of file a1.c of current working directory.*
cp /bin/ls /tmp/AA	*Creates a new file AA in /tmp directory with the content of /bin/ls*
cp /tmp/a1.c .	*Creates a1.c file in current working directory with the content of file /tmp/a1.c*
cp a1.c a2.c	*Creates a2.c in current working direct with the content of a1.c*
cp *.c /tmp	*Copies all files with extension c in current directory to /tmp directory*
cp /tmp/*.c .	*Copies all files with extension c in /tmp directory to current working directory*

cp /bin/* /tmp	*Copies all files of /bin directory to / tmp*
cp -r sourcedirectory destinationdirectory	*Copies all files, subdirectories and files in them of the source directory to destination directory.*
cp *.c /bin	*This command will fail if you are a normal user as we do not have permissions usually on /bin directory. However, it will work for super (root) user. It copies all C language files of C.W.D to /bin directory.*
cp a1 a2 a3 /tmp	*Copies files a1 a2 a3 of current working directory to the directory / tmp*
cp -v –i –u a1 s1	*This command works in verbose mode as we have given –v option. That is, it displays what cp command is doing. If s1 file is older than a1 or it is not existing then it will ask you whether to overwrite the same with a1 content or not as we have given –I option also. If your answer is yes, then it overwrites. Else it does not do any thing.*

2.9 THE MV COMMAND

Command mv is used to move file(s) from one directory to another directory or to rename the file.

The options include

-i interactive confirmation overwrites

-f force a copy

-R recursively copy to a directory

Syntax

mv source destination

mv a1.c /tmp	creates a1.c file in /tmp directory while file a1.c of current working directory is removed.
mv a1.c a2.c	Creates a2.c in current working direct with the content of a1.c while a1.c is disappeared
mv *.c /tmp	Moves all files with extension c in current directory to /tmp directory
mv /tmp/*.c .	Moves all files with extension c in /tmp directory to current working directory
mv /bin/* /tmp	*Moves all files of /bin directory to /tmp*

2.10 THE WC COMMAND

wc filename or wc<filename

These command displays number of lines, words and characters in the given file.

wc -l filename

This displays number lines in the given file.

wc -w filename

This displays number words in the given file.

wc -c filename

This displays number characters in the given file.
The following commands works same as "wc filename".

wc –l –w –c filename

wc –lwc filename

wc –w –l –c filename

wc –wlc filename

wc –l –c –w filename

wc –lcw filename

This flexibility in command line options became possible because of the intelligent command line parser, shell. That is, with most of the UNIX commands we can change the order of the options and also group the options.

2.11 THE LINK FILES

Links (shortcuts) can be used to define aliases to files or directories. We may define links to reduce typing burden while referring to a file which in a farthest (deepest) directory. UNIX supports to types of links (shortcuts) for files and directories known as hard links and symbolic links.

Example:

ln a1 a6

Here, a6 becomes hard link to the file a1. Whatever operations we do on a6 is really seen from a1 also. The reverse also is true. In fact, a6 will not take extra disk space. If we delete a1 (or a6) yet the file content is accessible through other name.

Hard links can not be created to directories. Also, they can not be created to the files of other partitions.

ls -l a1 a6 **gave the following result**

```
-rw-r—r—    2 root    root    20 Feb 14 00:13 a1
-rw-r—r—    2 root    root    20 Feb 14 00:13 a6
```

ln a1 a7 **(another hardlink is created for a1).**

ls -al a1 a6 a7 **gave the following result**

-rw-r—r— 3 root root 20 Feb 14 00:13 a1
-rw-r—r— 3 root root 20 Feb 14 00:13 a6
-rw-r—r— 3 root root 20 Feb 14 00:13 a7

We can observe that link count is increasing whenever a new hard link is created for a file. Similarly, whenever we remove a hard link file (or original file) link count will be reduced.

rm a6

ls -l a1 a7 **gives results**

-rw-r—r— 2 root root 20 Feb 14 00:13 a1
-rw-r—r— 2 root root 20 Feb 14 00:13 a7

I-node numbers of hard link and original files are same.

ls -li a1 a7

264826 -rw-r—r— 2 root root 20 Feb 14 00:13 a1
264826 -rw-r—r— 2 root root 20 Feb 14 00:13 a7

The following commands displays same thing.

cat a1

cat a7

Assume, we add some text to a1 through appending operator like :

cat>>a1

This is additional matter added through a1 to a1.

^d

Still, we can same content either through a1 or a7.

cat a1

cat a7

Now, let us assume that will add some text to a7 by executing the following command.

cat>>a7

This is additional matter added through a7 to a1.

^d

Still, we can same content either through a1 or a7.

cat a1

cat a7

Now, we remove a1 (original file). However, the content is accessible through a7 (hard link file).

rm a1

cat a7

Symbolic Links

Symbolic links can be created by using –s option with ln command. In the following example, we are creating symbolic link file (a8) to the file a1.

ln -s a1 a8

ls -l a1 a8

```
-rw-r—r—    1 root    root    20 Feb 14 00:13 a1
lrwxrwxrwx  1 root    root    2 Feb 14 00:20 a8 -> a1
```

We can see the difference. Though, whatever operations we do on symbolic link file really takes place on the original file. However, if we delete original file, the information of the file can not be accessible through symbolic link unlike hard link files. Of course, if we delete symbolic link yet the information is accessible through original name. Moreover, i-node numbers or original file and symbolic link files are different. In fact, symbolic link file will take separate disk block in which path of the original file is saved.

Is -li a1 a8

```
264826      -rw-r—r—     3 root     root    20 Feb 14 00:13 a1
264831      lrwxrwxrwx   1 root     root    2 Feb 14 00:20 a8 -> a1
```

The following commands displays same thing.

cat a1

cat a8

Assume, we add some text to a1 through appending operator like :

cat>>a1
This is additional matter added through a1 to a1.
^d

Still, we can same content either through a1 or a7.

cat a1

cat a8

Now, let us assume that will add some text to a7 by executing the following command.

cat>>a8
This is additional matter added through a8 to a1.
^d

Still, we can same content either through a1 or a8.

cat a1

cat a8

Now, we remove a1 (original file). However, the content is not accessible through a8 (symbolic link file).

rm a1

cat a8 (We will get error).

A main advantage of symbolic link files is that they can be used to create links for directories and also to the files of other partitions. In fact, symbolic links are used for SW fine tuning. For example, check for file 'X' in Linux system, which is normally symbolic link to the appropriate X server (Check in /usr/X11R6/bin).

ls -l /usr/X11R6/bin/X **gave me the following results**

lrwxrwxrwx 1 root root 7 Feb 7 06:31 /usr/X11R6/bin/X -> XFree86

If we want to change (switch) to some other X server, simply we change X to point to that server and start the X server.

Also, consider one practical application related to system administration. Assume, that total system is currently on a single disk and is almost full. Thus, administrator is decided to use another disk (or partition) to allocate space for new users. However, there are many automatic system administration scripts which assume all the user's directories are under / home directory. Which means new users also should be under /home in principle. However, that drive in which /home is physically available is full. Thus, we physically allocate space for new users in another partition (new drive) and then create symbolic links to them under /home directory such that the new users will be logically under /home.

2.12 THE MORE COMMAND

This command is used to see the content of the files page by page or screen by screen fashion. This is very useful if the file contains more number of lines. The syntax of this command usage is :

more **[-**dlfpcsu **] [-**num **] [+/ pattern] [+ linenum] [*file ...*]**

OPTIONS

Command line options are described below. Options are also taken from the environment variable **MORE** (make sure to precede them with a dash ("-")) but command line options will override them.

-num

This option specifies an integer which is the screen size (in lines).

-d

more will prompt the user with the message "[Press space to continue, 'q' to quit.]" and will display "[Press 'h' for instructions.]" instead of ringing the bell when an illegal key is pressed.

-l

more usually treats **^L** (form feed) as a special character, and will pause after any line that contains a form feed. The -**l** option will prevent this behavior.

-f

Causes **more** to count logical, rather than screen lines (i.e., long lines are not folded).

-p

Do not scroll. Instead, clear the whole screen and then display the text.

-c

Do not scroll. Instead, paint each screen from the top, clearing the remainder of each line as it is displayed.

-s

Squeeze multiple blank lines into one.

-u

Suppress underlining.

+/

The **+/** option specifies a string that will be searched for before each file is displayed.

+num

Start at line number **num**

While we are viewing the file contents, the following COMMANDS can be also used.

h or ?

Help: display a summary of these commands. If you forget all the other commands, remember this one.

SPACE

Display next k lines of text. Defaults to current screen size.

z

Display next k lines of text. Defaults to current screen size. Argument becomes new default.

RETURN

Display next k lines of text. Defaults to 1. Argument becomes new default.

d or ^D

Scroll k lines. Default is current scroll size, initially 11. Argument becomes new default.

q or Q or INTERRUPT

Exit.

s

Skip forward k lines of text. Defaults to 1.

f

Skip forward k screen full of text. Defaults to 1.

b or ^B

Skip backwards k screen full of text. Defaults to 1. Only works with files, not pipes.

Go to place where previous search started.

=

Display current line number.

/ pattern

Search for kth occurrence of regular expression. Defaults to 1.

n

Search for kth occurrence of last r.e. Defaults to 1.

Example usages :

more filenames(s)

more file1 file2

This displays content of the files file1 and file2 one after another.

more <file1

This also displays the content of the file1 in screen by screen fashion.

more file1 file2 ... filen > XXX

This command creates file XXX such that it contains the content of all the given files in the strictly same order.

more +/rao filename

This command displays the content of the given file starting from the line which contains the string "rao".

more +10 filename

This command displays the content of the file from 10'th line.

The following command creates duplicate file of file x1 with name x2.

more <x1 >x2

2.13 THE LESS COMMAND

This is also used for viewing the file contents in page by page fashion. Unlike, more command this will be having extra freedom to move in backward director also. The general syntax of this command is :

less options filename(s)

Some of the available options are :

- -g: Highlights just the current match of any searched string.
- -I: Case-insensitive searches.
- -M: Shows more detailed prompt, including file position.
- -N: Shows line numbers (useful for source code viewing).
- -S: Disables line wrap ("chop long lines"). Long lines can be seen by side scrolling.
- -?: Shows help.

While viewing the file contents, the following commands can be used.

- [Arrows]/[Page Up]/[Page Down]/[Home]/[End]: Navigation.
- [Space bar]: Next page.
- **b**: Previous page.
- *n***g**: Jump to line number *n*. Default is the start of the file.
- *n***G**: Jump to line number *n*. Default is the end of the file.
- **/***pattern*: Search for *pattern*. Regular expressions can be used.
- **n**: Go to next match (after a successful search).
- **N**: Go to previous match.
- **m***letter*: Markthe current position with *letter*.

- **`***letter*: Return to position *letter*. [**`** = single quote]
- **`^** or **g**: Go to start of file.
- **`$** or **G**: Go to end of file.
- **s**: Save current content in a file.
- **=**: File information.
- **F**: continually read information from file and follow its end. Useful for logs watching. Use **Ctrl**+**C** to exit this mode.
- **-***option*: Toggle command-line option -*option*.
- **h**: Help.
- **q**: Quit.

2.14 THE PG COMMAND

This command is also used to see the content of the files in page by page fashion. However, this is not available in recent versions. Rather more command is in wide use and is more flexible.

2.15 THE NL COMMAND

This command is used to display the content of the file along with line numbers.

Example :

nl filename

2.16 THE TAIL COMMAND

The usage syntax of this command is given as :

tail filename(s)

This command displays last 10 lines of each of the given file(s).

tail -1 filename(s)

This command displays last line of the given file(s).

tail -3 filename(s)

This command displays last three lines of each of the given file(s).

tail +2 filename(s)

This command displays second line to last line of the given file(s).

2.17 THE HEAD COMMAND

The usage syntax of this command is :

head filename(s)

The above command displays first 10 lines of each of the given file(s).

head -2 filename(s)

This command displays first 2 lines of the given file(s)

What happens if we execute the command: dead +2 filename

2.18 THE LS COMMAND

This command displays names of the files and directories of current directory. For example, we have got the following output.

```
a1
a2
a3
a4
a5
```

The following command displays names of files and directories of current directory in long fashion. That is, file permissions, owner name, group, links, time stamps, size and names.

ls -l

```
total 4
-rw-r—r— 1  root      root     0 Feb 13 23:55 a1
-rw-r—r— 1  root      root     0 Feb 13 23:55 a2
-rw-r—r— 1  root      root     0 Feb 13 23:56 a3
-rw-r—r— 1  root      root     0 Feb 13 23:55 a4
-rw-r—r— 1  root      root     290 Feb 13 23:59 a5
```

In UNIX, files whose names starts with . are called as hidden files. If we want to see their details also then we have to use -an option (Of course either alone or with other options).

For example, the following command displays other files also whose names starts with `.' .

ls -al

```
total 12
drwxr-xr-x   2 root    root     4096 Feb 14 00:01 .
drwxr-x—     29 root   root     4096 Feb 14 00:01 ..
-rw-r—r—     1 root    root     0 Feb 13 23:55 a1
-rw-r—r—     1 root    root     0 Feb 13 23:55 a2
-rw-r—r—     1 root    root     0 Feb 13 23:56 a3
-rw-r—r—     1 root    root     0 Feb 13 23:55 a4
-rw-r—r—     1 root    root     882 Feb 14 00:01 a5
-rw-r—r—     1 root    root     0 Feb 14 00:01 .aa1
```

A Note on File types

UNIX supports a small number of different file types. The following Table 2.1 summarizes these different file types. What the different file types are and what their purpose is will be explained as we progress. File types are signified by a single character.

Table 2.1 UNIX file types

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

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

- **"normal" files,**

Files under UNIX are just a collection of bytes of information. These bytes might form a text file or a binary file.

When we run ls –l command we will see some lines starts with – indicating they are normal files.

- **directories or directory files,**

Remember, for UNIX a directory is just another file which happens to contain the names of files and their I-node. An i-node is an operating system data structure which is used to store information about the file.

When we run ls –l command we will see some lines starts with d indicating they are normal files.

- **special or device files.**

Explained in more detail later on in the text these special files provide access to devices which are connected to the computer. Why these exist and what they are used for will be explained.

Run the following commands.

ls –l /dev/ttyS*

We will see that every line to start with 'c' indicating they are character special files; it is acceptable to us as they refer to terminals which are character devices.

ls –l /dev/hda*

We will see that every line to start with 'b' indicating they are block special files; it is acceptable to us as they refer to disk partitions which are block devices.

The following command displays details of the files in chronological order.

ls -alt

```
total 12
drwxr-xr-x   2 root   root    4096 Feb 14 00:03 .
drwxr-x—    29 root   root    4096 Feb 14 00:03 ..
-rw-r—r—    1 root   root    1451 Feb 14 00:03 a5
```

```
-rw-r—r—    1 root    root    0 Feb 14 00:01 .aa1
-rw-r—r—    1 root    root    0 Feb 13 23:56 a3
-rw-r—r—    1 root    root    0 Feb 13 23:55 a2
-rw-r—r—    1 root    root    0 Feb 13 23:55 a1
-rw-r—r—    1 root    root    0 Feb 13 23:55 a4
```

ls -l filename

It displays only that file details if it exists.

ls -l directoryname

It displays the files and directory details in the given directory.

All the options -a, -t etc can be also used. Moreover, UNIX commands will be having excellent command line interface. Thus, all the following commands are equivalent.

ls -a -l -t

ls -alt

ls -a -t -l

ls -atl

ls -l -a -t

ls -lat

ls -l -t -a

ls -lta

ls -t -l -a

ls -tla

ls -t -a -l

ls -tal

-R option with ls command displays details of files and subdirectories recursively.

Example :

ls -alR / (Of course you can go for a cup of coffee and come back before you see the prompt again!!).

This command displays all the files in UNIX system.

2.19 THE RM COMMAND

This command can be used to remove file(s)

Example :

rm xyz

This command removes file xyz (If it is not write protected).

Only legal owner of the file can remove file (Exception for super user).

rm f1 f2 f3 .... fn

Removes all files f1, f2, ... fn.

rm a*.c

Removes all files with extension c and primary name starts with a.

rm a?.c

Removes all files with extension c and primary name is two characters length with first character as a.

rm a[0-9]*.c

Removes all files with extension c and primary name starting with a and second character as digit.

rm a[!a-zA-Z0-9]*.c

Removes all files with extension c and primary name starting with a and second character is other than alphanumeric.

rm -R directoryname

Removes all files, sub-directories of the given directory recursively.

rm -i file(s)

This allows interactive deletion. That is, it prompts before deleting the file(s).

rm -F file(s)

File(s) are deleted forcibly (ignoring permissions).

2.20 THE UNLINK FILENAME

This command also removes the given file. It is usage is:

unlink filename

2.21 THE FUSER COMMAND

With this command we can know who are using a given file. It is advisable to use this command, especially before removing any file. The syntax of this command is :

fuser –u filename

This command displays users who are currently using the given file.

2.22 THE SCRIPT COMMAND

This is very useful to capture the activity on a terminal.

2.23 DOS2UNIX [FILENAME]

This command changes a file from dos formatting to UNIX formatting. Some applications such as CGI scripts must be UNIX formatted in order to run.

Example :

dos2UNIX filename

UNIX2dos [filename] This command changes a file from UNIX formatting to DOS formatting. It’s usage syntax is :

UNIX2dos filename

Both the above commands give converted output on the screen. If we want the same in a file, we can use redirection operator.

The d2u command This command converts the line endings of text files from DOS style (0x0d 0x0a) to UNIX style (0x0a).

Usage: d2u [OPTION...] [input file list...]

Main options (not all may apply)

-A, —auto	Output format will be the opposite of the auto detected sour format
-D, —u2d	Output will be in DOS format
—UNIX2dos	Output will be in DOS format
-U, —d2u	Output will be in UNIX format
—dos2UNIX	Output will be in UNIX format
—force	Ignore binary file detection
—safe	Do not modify binary files

mac2UNIX [filename]

This command changes a file from Mac formatting to UNIX formatting.

Example :

mac2UNIX filename

On some versions of UNIX, we are having commands such as todos, toUNIX, to_dos, or to_UNIX to convert DOS files to UNIX and vice versa.

2.24 THE BASENAME COMMAND

This command can be used to separate name of the file from its full path. It is also possible to remove extension from the file's name. This commands usage will be :

basename filenamewithpath

basename filenamewithpath [extension]

Examples :

basename /usr/bin/sort	Output **"sort".**
basename include/stdio.h .h	Output **"stdio".**

2.25 THE DIRNAME COMMAND:

This command can be used to extract directory of a file from its full path.

dirname filenamewithpath

Examples :

dirname /usr/bin/sort	Output **"/usr/bin".**
dirname stdio.h	Output **".".**

2.26 PRINTING

The general syntax of the command is :

lpr [options] files...

lpr -#2 filename	prints two copies of the given file
lpq	prints the printer queue status along with printer process job id.
lprm jobid	removes specified printer job id from printer queue (only legal owner can do this. Exception for super user).

2.27 MTOOLS

Mtools are used to copy files from/to floppy's.

mcopy rao a:	copies file rao of PWD to floppy.
mcopy a:\rao .	copies file rao from floppy to C.W.D
mdel a:\rao	removes file rao from floppy
mdir	displays content of floppy
mcd	changes directory in floppy

2.28 CONCLUSIONS

This chapter gives overview of most commonly used Linux commands such as cp, mv, cd, pwd, passwd, etc.,. It also explains about hard link and symbolic link files. It describes command history and its use to increase the productivity of the user.