❖ File/Directory Related command

1) ls:

This command is used **see the list of files and directory at current location or specified path**.

Syntax:

$ls [-Option] [Path]

Options:

-1	Shows Your **Full information** (Like a permissions, link, owners, size etc) of file and directory
-ld	List **only directory**
–x	**Multicolumn** output
-F	Marks **directories** with **slash (/),** and excitable **file** with **asterisk (*).**
-a	**Show all filenames**, even **file that are hidden** (beginning with a dot)
-R	**Recursive list** (content of directories and subdirectories file)
-r	**Sorts filenames** in **reverse order**
-t	Sorts filenames display **column wise** by **last modification time**
-lt	Sorts filenames listing **each line** by **last modification time**
-u	Sorts filenames by **last access time**
-lu	Sort by listing each line shows last access time
-i	Display inode number
ls ~	Display List the content(file & directories) of your home directory.
ls /	List the Content of your root directory.
ls ../	List the Content of your parent directory.
ls */	List the Contents of al sub directory.
ls -d */	List the directories in the current directory.

(2) cat

This command is **use to displays files contents**, **creates new file**, or **appends data to an existing file**.

Syntax 1:- To display the content of file

$cat [option] File Name

Example 1

$cat Example1.txt **OR** $cat abc Example1.txt pro4

Syntax 2 :- To create a new file (if this file is exist then override the data)

$cat > File Name
Enter the file content
Ctrl + d

Example 2

$ca > Example2.txt
Hello, this my second command.
Ctrl + d

Syntax 3:- To append existing file

$cat >> File Name
Enter the file content want to append
Ctrl+d

Example 3

$cat >> Example1.txt
Enter the file content want to append
Ctrl+d

Options:

-b Display **number with each line**, but **omit** the line number with blank lines.

-n Display number with each line (also **include with blank** lines).

-s To compress **more than 2** blank line into one blank line.

-e A **$** display at the **end of each line**.

-t Tab will be displayed as **^I**.

(3) cd (Change Directory)

This command is used to **change one directory forward**, **backward** and **supporting to go to root directory** directly.

Syntax

cd dir_nm	Would go into the given **particular directory**.
cd ..	Change working directory to **parent directory**.
cd ../../	Would go back TWO directories.

Example

If current directory /home/abc and user home directory is /home/guest then

$cd abc	now current directory is **abc**
$cd test	now current directory is / abc/test
$cd ../../	Now you **go back two directories**

(4) pwd: (Print / Present working directory)

This command is used to see path for working directory in unix you can't see the prompt with path at that time if we want to know the what is our current path.

Syntax

$pwd

Example:- If current directory is /home abc so, will show you /home/abc

$pwd

(5) mv command (Renaming Files and Moving a file from one place to another place)

The mv command renames files. It has **two distinct** functions:

i) It **renames** a file or directory.
ii) It **moves** a group of files to a different directory.

i) **To rename a file or directory ::** mv **doesn't create a copy** of the file; it only renames it. No additional space is consumed on disk during renaming. If the destination file **doesn't exist, it will be created**.

Syntax 1

$mv Old_File_Name New_File_Name

Example [To rename the file **abc.txt to xyz.txt(**destination file**)]**

$mv abc.txt xyz.txt

ii) To move a file from one place to another

Syntax

$mv File_Name Destination_folder or path of Destination folder

Example

To move a.txt file from root to home/abc directory then following command is used from root.

$mv a.txt /home/abc OR $mv a.txt ../

Above command will **move file** named **a.txt from root to /home/abc**

(6) cp (Copying a file)

This command is used to copy a file.

Syntax

$cp [option] Source file path or file Target path

Option

-i ask before overwriting a destination file
-r it copies directory structure.

Example

To copy a a.txt file from /home/abc to /home/guest following command is used

$cp /home/abc/a.txt /home/guest

Above command will copy a.txt file from /home/abc to /home/guest directory

(7) ln (Link a file)

This command is used to link a file. Links are used for sharing of files from two different locations in directory structure.

There are two types of link ::

1) soft link
2) hard link.

Soft Link: - It links file name **if source is deleted** then link **will not be found**

Hard Link: - It links physical storage area of the file if **source file is deleted** still it will target the physical part of the file and **data can be found.**

Syntax:

$ln [Option] Source File Name Link File Name

Option

-s Creates **soft link,** if this option is **not specified with ln command then hard link** will be created.

Example

$ ln –s abc.txt xyz.txt

Soft link will be created if ***abc.txt*** file will be deleted then user will not be able to see the content of xyz.txt because it is related to abc.txt file name.

$ ln abc.txt xyz.txt

Hard link will be created if ***abc.txt*** file will be deleted still user will be able to work will the content of abc.txt file.

(8) rm (Remove/delete a File)

Syntax

$rm [Option] [Path] **OR** File_Name

Options

-i	Ask before removing a file
-r	Performs recursive deletion in directory
-f	Removes write protected file also

Example

$rm /home/guest/abc.txt

Above command will remove abc.txt file.

(9) rmdir: (Remove directory)

Syntax:

$rmdir [option] Directory_Name

Option

-p, --parents

Remove DIRECTORY and its ancestors; e.g., 'rmdir -p a/b/c' is similar to 'rmdir a/b/c a/b a'

-v, --verbose

Output a diagnostic for every directory processed.

-r --recursive

Would remove a directory, even if not empty.

This command is useful to remove **empty directory the prime condition** of this command is this that whatever the directory is, you are going to remove it **that must be empty** else message will display on the screen that directory is not empty.

(10) mkdir: (Make Directory)

This command is used to **create a directory** on current path or specified path.

Syntax:

$mkdir [options] Directory_Name

Options:

-m, --mode=MODE
set file mode (as in chmod), not a=rwx - umask

-p, --parents
no error if existing, make parent directories as needed

-v, --verbose
print a message for each created directory

(11) Umask (User mask)

- Whatever the file is created by the owner at that time which permission is **default given to user**, **group** and **other** that default permission will be set using Umask command.
- The umask value tells UNIX which of the **three permission are to be denied** rather than granted. The current value of umask can be easily determined by just typing umask.
- Default on directory 755 [777 – 022] out 777 and file 644 [666 – 022] out 666.

Syntax

$umask

It's display **0022**, Here **first** 0 indicates that for owner no permission is denied, **Second** 0 indicates that for user no permission is denied, **third** number 2 indicates that write permission is denied to group and **fourth** number two is also indicating that write permission denied to others.\

Example1:- Changing default permission using umask

$umask 111

This would see to it that here on wards any new file that you create would have the permissions 555 and any directory that you create would have the permission 666.

Example2:- Changing Particular File/directory

$umask 0044 abc

12) Chmod (Change Permissions of File and Directory)

Generally, the default setting write-protects a file from all but the user may have read access. However, this may not be so on your system. To know your system's default, **create a file.**

```
$cat > xstart
$ ls –l
-rw-r--r--    1    kumar    metal  1906  Sep   5    23:38  xstart
```

Syntax

$chmod <<User Category>> <<Operation>> <<Permission>> <<File Name>>

User Category : User, Group, Other
Operation : Assign or remove a permission using +, – and =.
Permission : Read, Write, Execute

Example

To assign execute permission to the user of the file **xstart**, we need to write following command

```
$chmod  u+x   xstart
$ls –l
-rwxr--r--  1  root  root  1906 May 10  20:30  xstart
```

To use multiple characters to represent the user category (ugo).

```
$chmod ugo+x  xstart        or    $chmod  a+x   xstart   or    $chmod  +x    xstart
$ls –l
-rwxr-xr-x  1  root  root  1906 May 10  20:30  xstart
```

To use more than one file at a time

$chmod u+x note note1 note2 note 3

To grant and revoke a permission at a time

$chmod a-x,go+r xstart

Abbreviations Used By chmod

Category	Operation	Permission
u user	+ Assigns Permission	r Read Permission
g group	- Remove Permission	w Write Permission
o Other	= Assigns absolute permission	x Execute permission
a All(ugo)		

Syntax Absolute Permissions

$chmod Combination of Decimal Code

Combination of Decimal Code

- Read Permission - 4 (Octal 100)
- Write Permission - 2 (Octal 010)
- Execute Permission - 1 (Octal 001)

Binary	Octal	Permissions	Significance
000	0	---	No permission
001	1	--x	Executable Only
010	2	-w-	Writable Only
011	3	-wx	Writable and Executable
100	4	r--	Readable Only
101	5	r-x	Readable and Executable
110	6	rw-	Readable and Writable
111	7	rwx	Readable, Writable and Executable

Example 1

$chmod 666 xstart;
$ls –l xstart;
-rw-rw-rw- 1 kumar metal 1906 May 10 20:30 xstart

The 6 indicates read and write permission (4 + 2). To restore the original permissions to the file, you need to remove the write permission (2) from group and others:

Example 2

$chmod 644 xstart
$ls –l xstart
-rw-r--r-- 1 kumar metal 1906 May 10 20:30 xstart

To assign all permissions to the owner, read and write permissions to the group, and only execute permission to the other, use following command.

Example 3

$chmod 761 xstart
-rwxrw---x 1 kumar metal 1906 May 10 20:30 xstart

(13) Chown: (Change Owner)

- This command is used to **change the owner of file**.
- It **transfers ownership** of a file to a user, and it seems that it can optionally **change the group as well.**
- The command **requires the user-id** (UID) of the recipient, followed by one or more filenames.

- Changing ownership requires super user permission, so let's first change our status to that of super user with the su command with sudo.

$sudo su
Password:
root@sbics:/home/sejal#

After the password is successfully entered, **sudo su returns** a # prompt, the same prompt used by root. su lets us acquire super user status if we know the root password. To now reject the ownership of the file note to **root**, use chown in the following way:

Syntax:

Chown < New_Owner_Name> <filename(s)>

Example

1. **Create admin:-**

sudo useradd bca_04_admin

2. **Make directory**:-

mkdir bca_04_doc

3. **check the admin:-**

ls –ld bca_04_doc

drwxr-xr-x 2 **root root** 4096 Jan 5 11:44 bca_04_doc

4. **change owner permission:-**

chown bca_04_admin bca_04_doc

5. **Once again check the owner**:-

ls –ld bca_04_doc

drwxr-xr-x 2 **bca_04_admin root** 4096 Jan 5 11:44 bca_04_doc

- Once ownership of the file has been given away to **root**, the permissions that previously applied to **root** now apply to **bca_04_admin**.
- Thus, **root** can no longer edit note since there's **no write privilege for Owner** and others.
- He can't get back the ownership either. But he can copy the file to his own directory, in which case he becomes the owner of the copy.

(14) chgrp: (Changing Group Owner):

By default, the group owner of a file is the group to **which the owner belongs**. The chgrp (change group) command changes a file's group owner. A user can change the group owner of a file, but only to a group to which she also belongs. Yes, a user can belong to more than one group. chgrp shares a similar syntax with chown.

Syntax:

chgrp <<New Group Name>> <<File Name>>

1. **Create group**:-
   sudo groupadd bca_04

6. **check the group:-**
   ls –ld bca_04_doc
   drwxr-xr-x 2 bca_04_admin **root** 4096 Jan 5 11:44 bca_04_doc

7. **change group permission:-**
   chgrp bca_04 **bca_04_doc**

8. **Once again check the group**:-
   ls –ld bca_04_doc
   drwxr-xr-x 2 bca_04_admin **bca_04** 4096 Jan 5 11:44 bca_04_doc

If bca_04_admin is also a member of the bca_04 group then above command will work, if he is not a member of the bca_04 group then super user can make the command work. Note that bca_04_admin can reverse this action and restore the previous group ownership (to metal) because he is still owner of the file and consequently retains all right related to it.

(15) Find (Locating Files)

This command is used to **find a particular file.**

Syntax

$find [Path]/File Name

Example 1

$find abc.txt **OR** $find /home/guest/abc.txt

User can use wild card character with file **name * means all character** and **? means 1 character**

Example 2

To find the file **which having extension name .txt** following command is used

$find *.txt

Example 3

To find the file which **2nd character is d** then following command is used

$find ?d*.*

(16) Pg commands

This command is used to see the information of the document page wise.

Syntax

$pg Starting_Line_Number No_of_Lines_Per_Page [Option] File Name

Options:

-*number*	The number of lines per page. Usually, default window size is 23.
-c	**Clear the screen** before a page is displayed, if the **terminfo** entry for the terminal provides this capability.
-e	Do not **pause** and **display** "(EOF)" at the end of a file.
-f	Do **not split long lines**.
-n	Without this option, commands must be terminated by a newline character (n). With this option, **pg** advances once a command letter is entered.
-p *string*	Instead of the **normal prompt ":"**, *string* is displayed. If string contains "**%d**", its first occurrence is replaced by the number of the current page.
+*number*	Start at the given line.

Example

$pg +10 –15 –p “Page No. %d” abc.txt

Explanation

- Above command starts displaying the contents of **abc.txt** file, 15 lines at a time from 10th line onwards.
- At the end of each displayed page a prompt comes which **displays the page number** on view.
- This prompt **overrides the default ‘:’** prompt of the pg command.

Command:

1. **H** help
2. **q or Q** Quit
3. **<blank>** next page
4. **L** Next Line
5. **N** **next file**
6. **P** **Previous file**

(17) More Command (Paging out)

This command is used to **see the information** of the document page wise.

Syntax

$more [option] File Name

Option	Description
-c	Page through the file by clearing the window. (not scrolling).
-f	Count logical lines rather than screen lines (wrapping text)
-r	Display all control characters.
-s	Displays multiple blank lines as one blank line.
-u	Does not display underline characters and backspace (^H).
-w	Waits for a user to press a key before exiting.
-n	Displays n lines per window.
+*num*	Displays the file starting at line number *num*.

Example

$more -3 abc.txt

Explanation

You will see the contents of **abc.txt** on the screen, one page at a time. At the bottom of the screen, you'll also see the **percentage** of the file that has been viewed: ----More----(20%)

Navigation of More Command

Irrespective of version, more uses the spacebar to **scroll forward a page at a time**. You can also scroll by small and large increment of lines or screens. To move forward one page, use

f or spacebar	To move page forward
b	To move back one page
Enter	One line forward
q	Quit

(18) less (Paging Out)

This command is used to see the information of the document page wise. This command work just like more command and it is older version.
It is similar to more, but has the extended capability of allowing both forward and backward navigation through the file.

Syntax

$less File_Name

Example

$less abc.txt

Explanation

You will see the contents of abc.txt on the screen, one page at a time. At the bottom of the screen, you'll also see the filename and : that is ready to accept the navigation command.

(19) head (First Default 10)

- When user retrieves the file data using 'cat' command, the default numbers of lines that will be displayed to the screen is 24.

- If file contains more than 24 lines, then the head of file will scroll away & user cannot see the head. So, Unix provides a special command as 'head'.

- The command is used to display the top most information of the files. The general syntax is like below:

Syntax:

$head [options] File_Name

Option:

-n, --lines=[-]NUM
Print the first NUM lines instead of the first 10; with the leading '-', print all but the last NUM lines of each file

-q, --quiet, --silent
never print headers giving file names

-v, --verbose
Always print headers giving file names

Example:

$head -15 myfile

(20) tail

- Like head command, Unix offers another command as 'tail'. It will display specified number of lines from the bottom of a file. The general syntax is like below:

Syntax:

$tail [option] File Name

-n, --lines=[+]NUM
Output the last NUM lines, instead of the last 10; or use -n +NUM to output starting with line NUM

-q, --quiet, --silent
never print headers giving file names

-v, --verbose
always print headers giving file names

Example:- Should you decide to view **last fifteen** lines you simply have to say.
$tail -15 myfile

(21) wc (Word Count)

This command is used to **count number of characters**, **words** and **lines** in a given input file. If file is not given, it takes input from standard input.

Syntax

$wc [Option] File Name

Options

-l	Counts number of lines
-w	Counts number of words
-c	Counts number of characters

Example

$ wc abc.txt

Output

2 20 42 abc.txt

Note: Spacebar and Enter also known as on character

(22) touch

The *touch* command is the easiest way to **create new, empty files**. It is also used to change the *timestamps* (i.e., dates and times of the most recent access and modification) on existing files and directories.

Syntax:- `$touch [option] file_name(s)`

When used without any options, touch creates new files for any file names that are provided as *arguments* (i.e., input data) if files with such names do not already exist. Touch can create any number of files simultaneously.

Touch Command Options

1. **-a**, change the access time only
2. **-c**, if the file does not exist, do not create it
3. **-d**, update the access and modification times
4. **-m**, change the modification time only
5. **-r**, use the access and modification times of file
6. **-t**, creates a file using a specified time

Example:

1. How to Create an Empty File

touch file1

2. How to Create Multiple Files

touch file1 file2 file3

3. Explicitly Set the Access and Modification times

touch -c -t YYDDHHMM file1

-rw-rw-r-- 1 sejal sejal 0 Feb 15 12:12 file1

touch -c -t 01101130 file1

-rw-rw-r-- 1 sejal sejal 0 Jan 10 11:30 file1

4. Create a File using a specified time

touch -c -t 01101130 file9

-rw-rw-r-- 1 sejal sejal **0 Jan 10 11:30 file9**

Concept of Redirection and Piping:

Standard Input, Output, Error

- **Many Unix commands get input from** what is called **"standard input"** and **send their output to standard output** (often abbreviated as standard input and standard output).
- Your shell sets things up that **standard input is your keyboard**, and **standard output is the screen**.
- Here's an example using the cat command. Normally, cat reads data from all of the files specified by the command line, and sends this data directly to standard output. Therefore, using the command:

(1) Output Redirection: (> / >>)

- Now, let's say that **you want to send the output of cat to a file**, to save our shopping list on disk. The shell lets you redirect standard output to a file name, by using the ">" symbol.
- Here's how it works: **for example:- $cat amg.txt > output.txt**
- The **>** operator **causes a new file to be created**.
- If you had already created a file named **output.txt**, it would be deleted and replaced with the new data.
- If you wanted to add the new data to the old output, you could use the >> operator.
  For example: $cat amg.txt >> output.txt
- This will add content of amg.txt to output.txt file both file must be exists at specified path.

(2) Input redirection: (<)

- There are two possible sources of input for UNIX commands.
- Programs such as **ls** and **find** get their input from the command line in the form of options and filenames. Other programs, such as **cat**, can **get their data from the standard input** as well as from the command line. Try the cat command with no options on the command line:
  $cat
- There is no response. Because n files are specified with the command, **cat** waits to get its input from your keyboard, the standard input file.
- The program will accept input lines from the keyboard until it sees a line which begins with **Ctrl+D, which is the end of file signal for standard input.**
- To redirect the standard input, you use the < operator. For example, if you wanted cat to get its input from output.txt you could use the command.

  $cat < output.txt
  The difference between this command and
  $cat output.txt

Is a subtle one. In filenames provided as options to a command, you can use filename substitution. When redirecting input, you must use the name of an existing file or device.

(3) Standard Error:

- Whenever the command is given wrongly at prompt there must be an error occurs to store the error message in file because we can see in windows base application that if any system error occurs that generates the log file for that particular error. In that way in unix such a file can be generated as standard error.

Example:

1. **vi test__redirect.sh**
   echo "hii"
   echo"hello"

2. **sh test_redirect.sh**
   hii
   test_redirect.sh: line 2: echohello: command not found

3. **sh test_redirect.sh 2>errorfile**
   hii

4. **cat errorfile**
   test_redirect.sh: line 2: echohello: command not found

File	File Descriptor
Standard Input STDIN	0
Standard Output STDOUT	1
Standard Error STDERR	2

(4) Concept of Piping:

- Suppose that you wanted a **directory listing that was sorted by the mode file type plus permissions**. To accomplish this, you might redirect the output from ls to a data file and then sort that data file.

For example.

$ls –l > tempfile

$sort < tempfile
-rw-rw-r- 1 marsha adept 1024 Jan 20 14:14 Lines.dat
-rw-rw-r- 1 marsha adept 3072 Jan 20 14:14 Lines.idx
-rw-rw-r- 1 marsha adept 256 Jan 20 14:14 Pages.dat

Although you get the result that you wanted, there are **three drawback to this method**:

1. You might end up with **a lot of temporary file** in your directory.
2. You would have to **go back and remove them**.
3. The sort program **does not begins its work until the first command is complete**.

- This is not too significant with the small amount of data used in this example, but it can make a considerable difference with larger files.

- Fortunately, **there is a better way**.
- The pipe symbol (|) causes the standard output of **the program on the left side of the pipe to be passed directly to the standard input of the program on the right side** of the pipe symbol.
- Therefore, to get the same results as before, you can use the pipe symbol.

For example

$ls–l | sort

```
-rw-rw-r- 1 marsha adept 1024 Jan 20 14:14 Lines.dat
-rw-rw-r- 1 marsha adept 3072 Jan 20 14:14 Lines.idx
-rw-rw-r- 1 marsha adept 256 Jan 20 14:14 Pages.dat
```

To connect two or more operation within the same stream at that time pipe sign is used to perform such a operation in unix.