**UNIX/LINUX COMMANDS**

**Clear**—-it is used to clear all the data from the console.

**FILE -COMMANDS**

**Cat**—-create a new file

Ex: cat > testing.txt

…. ctrl+d….to exit from the cat file.

To check whether the file is created or not by using cat command, then we have to use

The **ls** command.

\*\* To display the content of the created cat file (or) cat files then we have to use

Cat filename

Ex: cat testing.txt

Cat > (creation of a file)

Cat fileName(display the content of a file)

\*\*we can also use cat command to concatenate one or more files to display the content.

Cat file1 file2

Cat file1 file2 file3

Ex: cat testing.txt test.txt

\*\*to append(adding the data to the existing file)  data/content to the file by using cat command then we have to use: cat >> File\_Name

Ex: cat testing.txt

Then we have to add the data and then ctrl+d

**CP**

Copy command is used to copy the content from one file to another file

Syntax: cp source-file  Targetted\_flle (new\_file/anotherfile)

Ex: cp testing.txt testing\_new.txt

    Ls

Cat testing\_new.txt(displays the content which was copied from testing.txt)

**MV**

Used for renaming a file(changing the name of the file)

Ex: mv testing.txt auto.txt

Ls

Cat auto.txt

Note: testing.txt will be renamed to auto.txt and content of testing.txt will be not deleted it will be same and it will be displaying in auto.txt.

\*\*Used for renaming a directory

Ex: mv mydir yourdir

ls

\*\*Used for moving files from one directory to another directory.

Ex: mv auto.txt yourdir

auto.txt(file in bigdata directory) yourdir(directory)

Cd ..going back one step to another directory.

**RM**

Remove the file

Ex: rm testing\_new.txt

To remove the directory we have to use **rm -r** command

Ex: rm -r yourdir

To remove the directory with empty files then we have to use rmdir command.

Ex: rmdir nishanth

Nishanth is a directory which contains no files or empty directory.

**DIRECTORY-COMMANDS**

Directory is noting but it is a folder or windows.

**Mkdir:** it is used to create a new directory/subdirectories.

Ex: mkdir yourdir

* We can create multiple directories at a same time in the same folder/directories.

Ex: mkdir testdir1 testdir2 testdir3

Use ls command to display the directories.

* We cn multiple subdirectories also at the same time.

  Ex: mkdir -p world/countries/city

Here world is a parent directory, inside world directory country is another directory, and inside country city is another directory.

**CD**

Cd .. – it will goes one step back to the other directory

Cd~----it will directly goes to the home/root directory.

Cd filename/directory– If we want to get particular file or directory then we are going to use

                                    this command.

**COMMANDS:**

**Touch:** touch is a command by which we can create an empty file.

 Ex: touch myfile.txt

\*\* By using touch command we can also create an hidden file also.

Ex: touch .myhiddenfile.txt (by using **touch .fileName**)

To display the hidden file,use **ls -a** command.

**Pwd:** it is present working directory.

This command is used to display the location of the current working directory.

**LS**

This command is used to display the all the files and directories which are present inside the current directory.

**ls -l:** long list.it will display all the detailed information about all the files and directories.

I.e: It is used to display the files and directories also it will be display the privileges(rwxr),users,and size of the file,date of the file.

**ls -a:** it will display all the files along with hidden files present inside the current directory.

**ls -l -a:** it will display all the files detailed information along with hidden files present inside the current directory.

**ls -F:** by using this command will be able to see directories and files separately.

(\_F will be adding a slash character at the end of the directory.)

Ex: ls -F

abc.txt/  abc.txt1/  abc.txt3  def.txt  employees.csv  myfile.txt  test1.txt  test.txt  world/

abc.txt/ abc.txt/ —-> here / represents a directories.

**ls -r:** By using this command all the files and directories will be displayed in a reverse order.

**ls -R:** it will display the directories and their subdirectories also.

**ls -lS**(s should be uppercase):

It will display lengthy list within a proper order with a higher size order.

i.e. (higher size data/file will be displayed first)

**ls -l fileName:**  syntax: ls -l file/Name (or) directory

If we want to display the file in a specific directory then we are going to use this command.

**WILDCARD CHARACTERS IN UNIX/LINUX:**

**?--> single character:**

If we want to display the files contains with only single character then we will use this command.

Ex: ls ? . \*

o/p:abc.txt  abc.txt1  abc.txt3  def.txt  employees.csv  myfile.txt  test1.txt  test.txt  world

Ex 2: ls ? . doc

O/P: it will displays the txt files only.

Here ? represents single character ,. Represents extension(like doc,txt etc). \* represents multiple characters.

\*---> multiple character: ex:ls t\*.txt

test1.txt  test.txt

T is the first character present in the files,\* is the multiple character,. Is the extension like doc,txt etc.

Ex 2: ls \*.txt

def.txt  myfile.txt  test1.txt  test.txt

**[ ]----> Range:** it will display the files which will be in the specified range.

 Ex: ls [a-z]\*.\*

abc.txt3  def.txt  employees.csv  myfile.txt  test1.txt  test.txt

Ex 2: ls [a-z]\*.txt

def.txt  myfile.txt  test1.txt  test.txt

**UNIX/LINUX COMMANDS:**

**Head:**

This command displays the starting content of a file.

By default, head value is 10.

Ex: **head state.txt**

\*\*Head file1 file2: it will displays the starting content at a time for multiple files.

Ex: **head state.txt state1.txt**

\*\* head -n num:The 'head -n' option displays specified number of lines.

Ex: **head -n 5 state.txt**

\*\*head -c num:The 'head -c' command counts the number of bytes of a file.

Ex: **head -c 6 state.txt**

**TAIL**

tail command is used to display the last ten lines of one or more files.

Ex:  **tail state.txt**

\*\*tail file1 file2:

Ex: **tail state.txt state1.txt**

\*\*tail -n num:The 'tail -n' option displays specified number of lines.

Ex:   **tail -n 5 state.txt**

\*\*tail -c num:The 'tail -c' command counts the number of bytes of a file.

Ex: **tail -c 6 state.txt**

\*\*

Print the states between 10 to 20 out of 30 records

Ex: **$ head -n 20 state.txt | tail -10**

Firstly filter out first 20 records by using head and then last records by using tail.

We can combine head and tail by using **pipes(|)** .

**More :** forward navigation and limited backward navigation.

Ex: more test.txt (using space for next page, enter for next line)

**Less:**

Ex: less test.txt(using space for next page, enter for next line, upperarrow for forward line)

**CHMOD**

chmod command is used to change the access permissions of files and directories.

3 types of roles.

owners/users(u)

groups(g)

others(o)

3 types of permissions

read(r), write(w), execute(x)

Symbolic/Text method:

+--- add permissions

—  → remove permissions

\*\* write a command to add execute permission to owner of the file.

Ex: **chmod u+x sample.txt**

\*\*write a command to add execute permission to owner and add read,write permissions to group and others

Ex: .**chmod u+x,g+rw,o+rw sample.txt**

       (or)

**Chmod u+x,go+rw sample.txt**

\*\*write a command to remove read permissions from group and others.

Ex:  **chmod g-r,o-r sample.txt**

                 (or)

**Chmod go-r sample.txt**

EX:4: **chmod u-w,g-w,o-r sample.txt**

         Write permission cancelled from owner.

         Write permission cancelled from group.

         Read permission cancelled from others.

Ex 5: **chmod u+rwx, g+rwx, o+r sample.txt**

Numeric Method:

read–(r)---4

write–(w)---2

execute–(x)---1

Total of rwx—7

Ex 1: **chmod 000 sample.txt**

No permissions to owners, groups and others.

Ex 2: **chmod 777 sample.txt**

All permissions given to owners, group and others.

Ex 3: **chmod 444 sample.txt**

Read permissions given to owner, group and others.

Ex 4: **chmod 600 sample.txt**

Read(4), write(2) permissions given to owners,

No permissions will be given to groups and others.

Ex 5: **chmod 664 sample.txt**

  Read , write permissions given to owners, groups and read permission given to others.

UNIX COMMANDS

**Who:** display information about all sessions on the system along with data

**Whoami:** displays the username of the current user.

**Hostname:** displays the name of the current host system.

\*\* **hostname -i**:

**Uptime:**

**Cal:** this command is used to get the calendar.

Cal 20002021/2022/2023—> to display the whole calender of a year use this command

**cal yearName**.

Specific moth in specific year—> cal month(in numerical) year

Ex: **cal 4 2024** it will display the 4th month of 2024 year.

**Cal -3** : previous 3 months

**Cal -y** : current year

**Cal -m10** : to display specific month in a current year

**Date**

* Date format should be include in double quotes “ “.
* Date “+%Y” :to display current year.
* Date “+%m” :to display current month(m).
* Date “+%d” :to display current date(d).
* Date “+%d-%m-%Y” :to display current date, month and year..

**WC**

wc command helps in counting the lines, words, and characters in a file. It displays the number of lines, number of characters, and the number of words.

**Syntax:** wc filename

\*\*we can multiple files with wc command

Ex: **wc file1 file2**

We can use combination of wordcount like (wc -lw fileName)

I.e. we can get no of lines and no of words in a file.

* Wc–wordcount
* Wc -l length of the file
* Cat vishnu.txt | Wc -l —-->(pipe)  no of lines (or) wc–lines
* Cat vishnu.txt | Wc wl —-->(pipe)  no of words
* Cat vishnu.txt | Wc -m —-->(pipe)  no of characters
* Cat vishnu.txt | Wc -c —-->(pipe)  no of bytes
* Cat vishnu.txt | Wc -L —-->(pipe)  find the longest word

Help command

:q!

**Based on your cursor**

* SPACE or l or right arrow--- character right
* h or left arrow -- space to the left
* j or down arrow -- down one line -- 4j--10j
* k or up arrow -- Up one line -- 9k -- 9 lines UP ---
* w --- word to the right.. 5w---10w--12w
* b --- word to the left.. --5b
* $ --- end of the line..
* 0 --- begining of the line..
* e --- end of the word to the right..
* -(minus) --- beginning of previous line..
* ) --- end of the sentence..
* ( --- begining of the sentence..
* } --- end of the paragraph..
* { --- b

rm -.txt to remove all the text files

Rm  -R\* - \* ,rm -R  h\*e , rm -R r\*s(remove all characters before r and s) it will remove all the directories those starts or having - will be deleted

D—15w

Less file.txt   to display the content(less) content

More file.txt

**SORT**

**Sort fileName:** to display the sorted data in a file.

The sorted data is only for the  display purpose ,after sorting the data the original data will not be changed.

\*\*to store a sorted data into a new file we are using this command

Ex**:  sort fileName.txt >fileName.txt**

\*\*we can sort the data in multiple files at a ti,e

Ex: **sort file1 file2**

\*\*To sort the data in a reverse(desc) order

Ex: **sort -r fileName.txt**

**UNIQ**

It is used to get unique(or) duplicate values/data from the file.

\*\*the main use of this command is to get the unique values from the file

\*\*before using uniq command, first the data should be sorted and stored in a new file

Ex: sort fileName.txt >file1Name.txt

**Uniq file1Name.txt** (2nd step)

\*\*to get the duplicate or repeated values from the file

EX:  **uniq -d file1Name.txt**

**uniq -u file1Name.txt** (we get not repeated values)

**uniq -u file1Name.txt** ( we can get how many times the value/line is repeating in the file)

**CMP**

This command will compare file byte to byte.

Ex: **cmp file1 file2**

**DIFF**

Diff command is used for comparing two files ,also displays mismatches information.

It is additionally provides special characters .

Ex: **diff file1 file2**

<-first file ,> -second file, c-change , d-delete, a-add

**COMM**

Used to compare two sorted files

It provides output in 3 columns.

In first column displays unique lines of first line.

In second column displays unique lines of second line.

In third column displays common lines in 2 files.

Ex**: sort file1.txt > file1\_sorted.txt**

**Sort file2.txt > file2\_sorted.txt**

**Comm file1\_sorted  file2\_sorted**

**PIPING:**

The pipe is used to combine two or more commands, and in this, the output of one command acts as input to another command, and this command’s output may act as input to the next command, and so on.

You can make it do so by using the pipe character ‘|’.

**SYNTAX:** command\_1 | command\_2 | command\_3 | .... | command\_N

Ex1: 1. List all files and directories and give them as input to `grep` command using piping in Linux

**ls | grep file.txt**

Ex2: Counting Lines in a Text File:

**cat file.txt | wc -l**

**Grep(Global Regular Expression Pattern)**

This command is used for searching a required pattern in a file.

EX: **$ cat cities**

Hyderabad

Mumbai

Kolkata

Chennai

Hyderabad

Delhi

**$ grep “Chennai” cities** (Chennai should be either in single or double quotes)

o/p: chennai

**$ grep “delhi” cities**

o/p: delhi

Delhi

Delhi

-i → ignores case sensitivity in the searching pattern.

Ex1: **$ grep  -i “delhi” cities**

Delhi delhi delhi

-n →Displays line numbers along with the matched patterns

Ex**: grep -n “Delhi” cities.txt**

 o/p: 2: Delhi

        7: Delhi

       10: Delhi

-c —>counts the number of times the searching pattern repeats

Ex: **grep -c “Delhi” cities.txt**

o/p: 3

-v →5display the lines that does not matched with the pattern

Ex: **grep -c “Delhi” cities.txt**

o/p: Hyderabad

Mumbai

Chennai

Kolkata

Hyderabad

Mumbai

Chennai

Kolkata

-l →display the file names that matches with the pattern.(i.e we can just display the files that matches with the given string/pattern)

Ex: **grep “delhi” cities cities**

o/p: cities: Delhi

       cities1:Delhi

**nishu$: grep -l  “delhi” cities cities**

o/p: Cities

       Cities1

**Grep with regular expressions**

Ex1:  Display all lines that start with ‘D’

^ represents starting of the line.

**Grep ‘^D’ cities.txt**

Ex2: Display all lines that ends with ‘i’

$ represents end of the line

**Grep ‘i$’ cities.txt**

Ex3: Display all lines that end with “bad”.

**Grep ‘bad$’ cities.txt**

EX4: Display all lines

that contain any of the letters A,B,C,D.

**Grep ‘[A-D]’ cities.txt**

  (or)

**Grep ‘[ABCD]’ cities.txt**

Ex 5: search for vowels in the file a,e,i,o,u

**Grep ‘[aeiou]’ cities.txt**

EX6: search for consonants in the file (other than a,e,i,o,u)

**Grep “[^aeiou]” cities.txt**

Ex7: search for multiple patterns

 -e search for multiple patterns

**grep -e “Delhi” -e “chennai” cities.txt**

Instead of ‘e’ we can use egrep command (i.e both the commands will gives the same result).

**egrep”(Delhi | chennai)” cities.txt**

**Egrep:**

egrep is a pattern searching command which belongs to the family of grep functions. It works the same way as grep -E does.

Ex: **egrep”(Delhi | chennai)” cities.txt**

\*\*differences between grep and egrep

**egrep “(Delhi|Chennai)” cities.txt**

**grep “(Delhi|Chennai)” cities.txt**

Grep command understands patterns but not al

Egrep command understands all the patterns

-F  search for fixed strings(No pattern)

**grep -F “delhi”**

**> Hyderabad**

**> Mumbai” cities.txt**

**Instead of F ,we can use fgrep command.(Fixed String Global Regular Expression Pattern)**

**fgrep “Delhi**

**>Hyderabad**

**>Mumbai” cities.txt**

Fgrep command cannot understand patterns.understand fixed strings.

**Grep commands with piping:**

Ex: **cat cities.txt | grep “Delhi”**

o/p: Delhi

       Delhi

       Delhi

Ex2**: cat cities.txt | grep “^D”**

o/p: Delhi

Delhi

Delhi

Ex3:  **ls | grep ‘^emp’**

o/p:  emplist1

        Emplist2

        Emplist\_sorted

Ex4:  **ls -l | grep ‘cities’.**

o/p: cities

       Cities1

Ex4:  **ls -l | grep ‘cities’ | wc -l.**

o/p: 4

**PS:**

Types of process:

1) foreground process 2) background process

The ps command is used to view currently running processes on the system. It helps us to determine which process is doing what in our system, how much memory it is using, how much CPU space it occupies, user ID, command name, etc.

Ex: **nishu$ ps**

o/p: it will display the result

EX2: **nishu$ pwd**(foreground process)

**nishu$ pwd &**     (background process)

* **PID** is the process ID of running command
* **TTY** is the type of terminal where current command is running
* **TIME** tells how much time is used by CPU to run the process
* **CMD** is current command

**KILL:**

It is used for manually terminating the processes

Ex: kill 2552(2552 is a argument/process number.)