**#1) cal**: Displays the calendar.

* **Syntax**: cal [[month] year]
* **Example**: display the calendar for April 2018
  + $ cal 4 2018

**#2) date:** Displays the system date and time.

* **Syntax**: date [+format]
* **Example**: Display the date in dd/mm/yy format
  + $ date +%d/%m/%y

**#3) banner**: Prints a large banner on the standard output.

* **Syntax**: banner message
* **Example**: Print “Unix” as the banner
  + $ banner Unix

**#4) who**: Displays the list of users currently logged in

* **Syntax**: who [option] … [file][arg1]
* **Example**: List all currently logged in users
  + $ who

**#5) whoami**: Displays the user id of the currently logged-in user.

* **Syntax**: whoami [option]
* **Example**: List currently logged in user
  + $ whoami

**Unix File System Commands Touch, Cat, Cp, Mv, Rm, Mkdir (Part B)**

**#1) touch**: Create a new file or update its timestamp.

* **Syntax**: touch [OPTION]…[FILE]
* **Example**: Create empty files called ‘file1’ and ‘file2’
  + $ touch file1 file2

**#2) cat**: Concatenate files and print to stdout.

* **Syntax**: cat [OPTION]…[FILE]
* **Example**: Create file1 with entered cotent
  + $ cat > file1
  + Hello
  + ^D

**#3) cp**: Copy files

* **Syntax**: cp [OPTION]source destination
* **Example**: Copies the contents from file1 to file2 and contents of file1 is retained
  + $ cp file1 file2

**#4) mv**: Move files or rename files

* **Syntax**: mv [OPTION]source destination
* **Example**: Create empty files called ‘file1’ and ‘file2’
  + $ mv file1 file2

**#5) rm**: Remove files and directories

* **Syntax**: rm [OPTION]…[FILE]
* **Example**: Delete file1
  + $ rm file1

**#6) mkdir**: Make directory

* **Syntax**: mkdir [OPTION] directory
* **Example**: Create directory called dir1
  + $ mkdir dir1

**#7) rmdir**: Remove a directory

* **Syntax**: rmdir [OPTION] directory
* **Example**: Create empty files called ‘file1’ and ‘file2’
  + $ rmdir dir1

**#8) cd**: Change directory

* **Syntax**: cd [OPTION] directory
* **Example**: Change working directory to dir1
  + $ cd dir1

**#9) pwd**: Print the present working directory

* **Syntax**: pwd [OPTION]
* **Example**: Print ‘dir1’ if a current working directory is dir1
  + $ pwd

# Unix Processes Control Commands Like Ps And Top (Part C)

| **Command** | **ps - displays a snapshot of all current processes** |
| --- | --- |
| Common Syntax | $ ps [options] |
| Example | $ ps -ef |
|  | Show every process running, formatted as a table |
| **Command** | **top - displays a live status of current processes** |
| Common Syntax | $ top [options] |
| Example | $ top |
|  | Show a live view of all current processes |

| **Command** | **bg - resume a background suspended a job** |
| --- | --- |
| Common Syntax | $ bg [job\_spec …] |
| Example | $ xterm Ctrl-Z $ bg |
|  | Continue running a job that was previously suspended (using Ctrl-Z) in the background |
| **Command** | **fg - bring a background job to the foreground** |
| Common Syntax | $ fg [job\_spec] |
| Example | $ xterm Ctrl-Z $ bg $ fg |
|  | Bring a previous background job to the foreground |

| **Command** | **clear – clear a terminal screen** |
| --- | --- |
| Common Syntax | $ clear |
| Example | $ clear |
|  | Clear all prior text from the terminal screen |
| **Command** | **history – print history of commands in the current session** |
| Common Syntax | $ history [options] |
| Example | $ history |

* **Control-C:** This command terminates the currently running foreground process.
* **Control-D:** This command terminates the currently running login or terminal session.
* **Control-Z:** This command suspends the currently running foreground process to the background.

# Unix Utilities Programs Commands: Which, Man, Find Su, Sudo (Part D)

**#1) ls**: List directory contents

* **Syntax**: ls [OPTION] [FILE]
* **Example**: list all (including hidden files) directory contents, in long format, sorted by time,
  + $ ls -alt

**#2) which:**Locate a command

* **Syntax**: which [-a] filename
* **Example**: List all paths from where ‘cat’ can run
  + $ which -a cat

**#3) man**: Interface for working with the online reference manuals.

* **Syntax**: man [-s section] item
* **Example**: Show manual page for the ‘cat’ command
  + $ man cat

**#4) su**: Change user-id or become super-user.

* **Syntax**: su [options] [username]
* **Example**: Change user-id to ‘user1’ (if it exists)
  + $ su user1

**#5) sudo**: Execute a command as some other user or super-user

* **Syntax**: sudo [options] [command]
* **Example**: Get a file listing of an unlisted directory
  + $ sudo ls /usr/local/protected

**#6) find**: Used to search for files and directories as mentioned in the ‘expression’

* **Syntax**: find [starting-point] [expression]
* **Example**: In ‘/usr’ folder, find character device files, of name ‘backup’
  + $ find /usr -type c -name backup

**#7) du**: Estimate disk usage is blocks

* **Syntax**: du [options] [file]
* **Example**: Show number of blocks occupied by files in the current directory
  + $ du

**#8) df**: Show number of free blocks for mounted file system

* **Syntax**: df [options] [file]
* **Example**: Show number of free blocks in local file systems
  + $ df -l

### **Grep Command in Unix with Examples**

**Syntax:** grep [options] [pattern] [file]

The pattern is specified as a regular expression. A regular expression is a string of characters that is used to specify a pattern matching rule. Special characters are used to define the matching rules and positions.

**#1) Anchor Characters:** ‘^’ and ‘$’ at the beginning and end of the pattern are used to anchor the pattern to the start of the line, and to the end of the line respectively.

**Example:** “^Name” matches all lines that start with the string “Name”. The strings “\<” and “\>” are used to anchor the pattern to the start and end of a word respectively.

**#2) Wildcard Character:** ‘.’ Is used to match any character.

**Example:**“^.$” will match all lines with any single character.

**#3) Escaped Characters:** Any of the special characters can be matched as a regular character by escaping them with a ‘\’.

**Example:** “\$\\*” will match the lines that contain the string “$\*”

**#4) Character Range:** A set of characters enclosed in a ‘[‘ and ‘]’ pair specify a range of characters to be matched.

**Example:** “[aeiou]” will match all lines that contain a vowel. A hyphen can be used while specifying a range to shorten a set of consecutive characters. **E.g.** “[0-9]” will match all lines that contain a digit. A carat can be used at the beginning of the range to specify a negative range. **E.g.** “[^xyz]” will match all lines that do not contain x, y or z.

**#5) Repetition Modifier:** A ‘\*’ after a character or group of characters is used to allow matching zero or more instances of the preceding pattern.

**The grep command supports a number of options for additional controls on the matching:**

* -i: performs a case-insensitive search.
* -n: displays the lines containing the pattern along with the line numbers.
* -v: displays the lines not containing the specified pattern.
* -c: displays the count of the matching patterns.

**Examples:**

* Match all lines that start with ‘hello’. **E.g:** “hello there”

$ grep “^hello” file1

* Match all lines that end with ‘done’. **E.g:** “well done”

$ grep “done$” file1

* Match all lines that contain any of the letters ‘a’, ‘b’, ‘c’, ‘d’ or ‘e’.

$ grep “[a-e]” file1

* Match all lines that do not contain a vowel

$ grep “[^aeiou]” file1

* Match all lines that start with a digit following zero or more spaces. **E.g:** “ 1.” or “2.”

$ grep “ \*[0-9]” file1

* Match all lines that contain the word hello in upper-case or lower-case

$ grep -i “hello”

### **Cut Command in Unix with Examples**

The cut command extracts a given number of characters or columns from a file. For cutting a certain number of columns it is important to specify the delimiter. A delimiter specifies how the columns are separated in a text file

**Example:** Number of spaces, tabs or other special characters.

**Syntax:**

cut [options] [file]

The cut command supports a number of options for processing different record formats. For fixed width fields, the -c option is used.

$ cut -c 5-10 file1

This command will extract characters 5 to 10 from each line.

For delimiter separated fields, the -d option is used. The default delimiter is the tab character.

$ cut -d “,” -f 2,6 file1

This command will extract the second and sixth field from each line, using the ‘,’ character as the delimiter.

**Example:**

**Assume the contents of the data.txt file is:**

Employee\_id;Employee\_name;Department\_name;Salary  
10001;Employee1;Electrical;20000  
10002; Employee2; Mechanical;30000  
10003;Employee3;Electrical;25000  
10004; Employee4; Civil;40000

**And the following command is run on this file:**

$ cut -c 5 data.txt

**The output will be:**

o

1

2

3

4

**If the following command is run on the original file:**

$ cut -c 7-15 data.txt

**The output will be:**

ee\_id; Emp

Employee1

Employee2

Employee3

Employee4

**If the following command is run on the original file:**

$ cut -d “,” -f 1-3 data.txt

**The output will be:**

Employee\_id;Employee\_name;Department\_name

10001;Employee1;Electrical

10002; Employee2; Mechanical

10003;Employee3;Electrical

10004; Employee4; Civil

### **ls Command in Unix with Examples**

**ls Syntax:**

ls [options] [paths]

**The ls command supports the following options:**

* ls -a: list all files including hidden files. These are files that start with “.”.
* ls -A: list all files including hidden files except for “.” and “..” – these refer to the entries for the current directory, and for the parent directory.
* ls -R: list all files recursively, descending down the directory tree from the given path.
* ls -l: list the files in long format i.e. with an index number, owner name, group name, size, and permissions.
* ls – o: list the files in long format but without the group name.
* ls -g: list the files in long format but without the owner name.
* ls -i: list the files along with their index number.
* ls -s: list the files along with their size.
* ls -t: sort the list by time of modification, with the newest at the top.
* ls -S: sort the list by size, with the largest at the top.
* ls -r: reverse the sorting order.

### **Tar Command in Unix with Examples**

The archive format preserves the directory structure, and the file system attributes such as permissions and dates.

**Tar Syntax:**

tar [function] [options] [paths]

**Tar options:**

**The tar command supports the following functions:**

* tar -c: Create a new archive.
* tar -A: Append a tar file to another archive.
* tar -r: Append a file to an archive.
* tar -u: Update files in an archive if the one in the filesystem is newer.
* tar -d: Find the diff between an archive and the filesystem.
* tar -t: List the contents of an archive.
* tar -x: Extract the contents of an archive.

While specifying the function, the ‘-‘ prefix is not required, and the function can be followed by other single letter options.

**Some of the supported options include:**

* -j: Read or write archives using the bzip2 compression algorithm.
* -J: Read or write archives using the xz compression algorithm.
* -z: Read or write archives using the gzip compression algorithm.
* -a: Read or write archives using the compression algorithm determined by the archive file name.
* -v: Perform the operations verbosely.
* -f: Specify the file name for the archive.

### **Unix Sort Command with Examples**

**Sort Syntax:**

sort [options] [files]

**Sort Options:**

**Some of the options supported are:**

* sort -b: Ignore blanks at the start of the line.
* sort -r: Reverse the sorting order.
* sort -o: Specify the output file.
* sort -n: Use the numerical value to sort.
* sort -M: Sort as per the calendar month specified.
* sort -u: Suppress lines that repeat an earlier key.
* sort -k POS1, POS2: Specify a key to do the sorting. POS1 and POS2 are optional parameters and are used to indicate the starting field and the ending field indices. Without POS2, only the field specified by POS1 is used. Each POS is specified as “F.C” where F represents the field index, and C represents the character index from the start of the field.
* sort -t SEP: Use the provided separator to identify the fields.

With the “-k” option, the sort command can be used to sort flat file databases. Without the “-k” option, the sorting is performed using the entire line. The default separator for fields is the space character. The -t option can be used to change the separator.

### **Unix Cat command with Examples**

**Cat Command Syntax:**

cat [options] [files]

The cat command becomes a very powerful tool when combined with Unix shell’s input and output redirection symbols:

* cmd > file.txt: the “>” symbol redirects the stdout stream from the command to replace the contents of the specified file. The file will be created if it doesn’t exist, or its contents will be replaced if it does.
* cmd >> file.txt: the “>>” symbol redirects the stdout stream from the command to append to the contents of the specified file. The file will be created if it doesn’t exist, or the new contents will be appended to the same file if it does.

**The cat command is a filter that can be used for multiple purposes:**

1. Display the contents of text files.
2. Copy text files into a new file.
3. Append a text file to the end of another text file.

**##Shell Scripting**

**Vi Editor :**

### **Modes of Unix Vi Editor**

The vi editor has three modes of operation viz. the command mode, the insert mode, and the ex-command mode.

#### **#1) Command mode:**

In this mode, all the keys work as commands. These keys are used for inserting, appending, deleting, opening new lines, moving the cursor over the paragraphs and sentences, etc. In this mode, the keys are not displayed but each key performs an operation.

By default the vi editor is in command mode, hence we cannot type text in command mode. In order to write programs or text in vi editor, we need to switch to the insert mode which can be done by pressing the escape button.

#### **#2) Insert mode:**

In this mode, we can insert, append, edit or replace texts. We can switch from the command mode to Insert mode by pressing the escape button and then press I or A to enter into insert mode.

#### **#3) Ex command mode:**

This mode is used for entering commands at the bottom line of the vi editor called as a command line. To switch to Ex command mode press escape key then type: (colon). In order to save the contents and quit from the vi editor press wq after the : (colon). i.e :wq.

**Vi editor saving  and quitting commands:**

**:w** -Save the contents of the file.

**:q** – Quit from vi editor.

**:q!** -quit from vi editor by discarding any changes.

**:wq** -Save the file and quit from the vi editor.

### **Operators in Unix**

#### **#1) Shell Arithmetic Operators Example**

**These consist of basic mathematical operations:**

* Addition: +
* Subtraction: –
* Multiplication: \*
* Division: /
* Modulus: %

Each of these operators performs the operation on two integer variables or constants.

**For Example, the below program illustrates each of these operations:**

$ c=`expr $a + $b`

$ echo “the value of addition=$c”

$ d=`expr $a - $b`

$ echo “the value of subtraction=$d”

$ e= expr $a \\* $b`

$ echo “the value of multiplication=$e”

$ f=`expr $a / $b`

$ echo “the value of division=$f”

$ g= echo `expr $a % $b`

$ echo “the value of modulus=$c”

The Unix shell does not natively support floating point operations.  A separate command line tool must be used for this.  The ‘bc’ co0mmand is the most standard tool for this.

**Example:**

$ c = `echo “$a + $b” | bc`

$ d = `echo “$a + $b” | bc`

Note that each of the operators needs to be surrounded by a space on both sides, and the ‘\*’ operators need to be escaped with a backslash ‘\’.

#### **#2) Shell Logical Boolean Operators Example**

**The logical operators in Unix are as follows:**

* Not:!
* And: -a
* Or: -o

### **Unix Conditional Statements The if-elif-fi**

Unix provides a number of relational operators in addition to the logical operators mentioned earlier. These can be used to compare numeric values.

* -lt less than
* -le less than or equal to
* -gt greater than
* -ge greater than or equal to
* -eq equal to
* -ne not equal to

**Unix provides a number of ways for conditionally executing the other commands.**

**These are covered below:**

**#1) The if statements**

**Example:**

if <control command>

then

<statements>

fi

**#2) The if…else statements**

**Example:**

if <control command>

then

<statements>

else

<statements>

fi

**#3) The if…elif…else…fi statement**

**Example:**

if <control command>

then

<statements>

elif

then

<statements>

else

<statements

fi

**Given below are some example programs that illustrate these conditional statements:**

**#1) Check if an input number is positive:**

$ echo “Enter a number”

$ read num

$ if [ $num -gt 0 ]

$ then

$ echo “It is a positive number”

$ fi

**#2) Check if an input number is positive or not:**

$ echo “Enter a number”

$ read num

$ if [ $num -gt 0 ]

$ then

$ echo “It is a positive number”

$ else

$ echo “It is not a positive integer”

$ fi

**#3) Check if an input number is positive, zero or negative:**

$ echo “Enter a number”

$ read num

$ if [ $num -gt 0 ]

$ then

$ echo “It is a positive number”

$ elif [ $num -eq 0 ]

$ then

$ echo “num is equal to zero”

$ else

$ echo “It is not a positive integer”

$ Fi

**The Shell Switch Case Syntax and Examples:**

case <word> in

<first pattern>)

<statements>

;;

<second pattern>)

<statements>

;;

\*)

<default statements>

;;

esac

Here, the value of the word expression is matched against each of the choice patterns.  If a match is found then the corresponding statements are executed until the ‘;;’ statement is encountered.  If there is no match, the default statements under ‘\*)’ are executed.

**The following is an Example of a switch case program:**

echo “Enter a number”

read num

case $num in

[0-9])

echo “you have entered a single digit number”

;;

[1-9][1-9])

echo “you have entered a two-digit number”

;;

[1-9][1-9][1-9])

echo “you have entered a three-digit number”

;;

\*)

echo “your entry does not match any of the conditions”

;;

Esac

### **Loops in Unix**

**You may use different loops based on the situation.**

**They are:**

**#1) Unix For loop statement**

**Example:** This program will add 1+2+3+4+5 and result will be 15

for i in 1 2 3 4 5

do

sum=`expr $sum + $i`

done

echo $sum

**#2) Unix While loop statement**

**Example:** This program will print the value of ‘a’ five times, from 1 to 5.

a=1

while [ $a -le 5 ]

do

echo “value of a=” $a

a=`expr $a + 1`

done

**#3) Unix Until loop statement**

This program will print the value of ‘a’ two times from 1 to 2.

a=1

until [ $a -ge 3 ]

do

echo “value of a=” $a

a=`expr $a + 1`

done

While running these loops, there may be a need to break out of the loop in some condition before completing all the iterations or to restart the loop before completing the remaining statements. This can be achieved with the ‘break’ and ‘continue’ statements.

**The following program illustrates the ‘break’ operation:**

num=1

while [ $num -le 5 ]

do

read var

if [ $var -lt 0 ]

then

break

fi

num=`expr $num + 1`

done

echo “The loop breaks for negative numbers”

### **Working with Functions in Unix**

Shell functions typically do not return the result to the calling code. Instead, global variables or output streams are used to communicate the result. The variable ‘errno’ is often used to communicate whether a command ran successfully or not.

A number of commands also print out their result into the ‘stdout’ stream so that the calling function can read into a variable.

**In this tutorial we will cover:**

* How to create functions
* Passing parameters to a function
* Returning a value from a function

**Syntax for defining functions:**

function\_name()

{

…

<statements>

…

}

To invoke a function, simply use the function name as a command.

**Example:**

$ function\_name

To pass parameters to the function, add space separated arguments like other commands.

**Example:**

$ function\_name $arg1 $arg2 $arg3

The passed parameters can be accessed inside the function using the standard positional variables i.e. $0, $1, $2, $3 etc.

**Example:**

function\_name()

{

…

c = $1 + $2

…

}

**Functions can return values using any one of the three methods:**

**#1)** Change the state of a variable or variables.

**#2)** Use the return command to end the function and return the supplied value to the calling section of the shell script.

**Example:**

function\_name()

{

echo “hello $1”

return 1

}

Running the function with a single parameter will echo the value.

$ function\_name ram

hello ram

Capturing the return value (stored in $?) as follows:

$ echo $?

1

**#3)** Capture the output echoed to the stdout.

**Example:**

$ var = `function\_nameram`

$ echo $var

hello ram

### **Unix Filter Commands**

* **grep:** Find lines in stdin that match a pattern and print them to stdout.
* **sort:** Sort the lines in stdin, and print the result to stdout.
* **uniq:** Read from stdin and print unique (that are different from the adjacent line) to stdout.
* **cat:** Read lines from stdin (and more files), and concatenate them to stdout.
* **more:** Read lines from stdin, and provide a paginated view to stdout.
* **cut:** Cut specified byte, character or field from each line of stdin and print to stdout.
* **paste:** Read lines from stdin (and more files), and paste them together line-by-line to stdout.
* **head:** Read the first few lines from stdin (and more files) and print them to stdout.
* **tail:** Read the last few lines from stdin (and more files) and print them to stdout.
* **wc:** Read from stdin, and print the number of newlines, words, and bytes to stdout.
* **tr:** Translate or delete characters read from stdin and print to stdout.

**Next, let’s work through some of these commands in detail:**

| **Command** | **grep - It is a command for pattern searching in a file and prints those lines containing that specified pattern. If the file name is not mentioned, grep searches in stdin.** |
| --- | --- |
| Common Syntax | $ grep [option] pattern [filename …] |
| Example | $ grep ‘[A-M]’ file1 Prints those lines which contains capital letters in the range of A to M |
| **Command** | **wc - It is a command to count the number of lines, words and characters in a file** |
| Common Syntax | $ wc [OPTION] ….[FILE] |
| Example | $ cat file1 Hello How do you do $ wc file1 2 5 20 file1  No of lines-2 No of words-5 No of characters(bytes)-20 |

| **Command** | **more - This command is used to display the page one screen at a time** |
| --- | --- |
| Common Syntax | more [options] file… |
| Example | $ls -l | more Will display long listing of files and directories one screen at a time |
| **Command** | **paste: this command is used to paste the contents of two files.** |
| Common Syntax: | paste [OPTION] ….[FILE]…. |
| Example: | paste file1 file2 This command will combine the contents of file1 and file2 |

### **Pipes in Unix**

A series of filter commands can be piped together using the pipe symbol: ‘|’. When two commands are piped together, the stdin of the second program is read from the stdout of the first program. This creates a powerful mechanism for running complex commands quickly.

| **Command** | **sort: this command is used to sort the contents of the file. This command is also useful to merge the sorted files and store the result in some file. The contents of the original file remain unaltered.** |
| --- | --- |
| Common Syntax: | sort[OPTION]…[FILE] |
| Example1: | sort file1 This command will sort the contents of file1 |
| Example2: | sort -o output\_file file1 file2 This will sort the contents of file1 and file2 and save the result in output\_file file. |
| **Command** | **cut – this command is used to cut a given number of characters or columns from a file. For cutting a certain number of columns it is important to specify the delimiter. A delimiter specifies how the columns are separated in a text file e.g. number of spaces, tabs or other special characters.** |
| Common Syntax: | cut OPTION …[FILE] |
| Example 1 | cut -c 5-10 file1 It will cut 5 to 10 characters from each line of file1 |
| Example 2 | cut -d “,“ -f2,6 file1 This will cut 2nd and 6th fields from file1, where the fields are separated by delimiter “,” |

This will cut 2nd and 6th  fields from file1, where the fields are separated by the delimiter “,”.

Let us now see an **Example** of using pipes to print out a sorted list of unique words. If file1 has a list of words in a random order with random repetitions, then the following piping can be used to achieve this.

$ sort file1 | uniq > file2

Here, the sort command reads input from the file ‘file1’ and sends the output to stdout. The pipe symbol causes the output of the sort command to be redirected to the input of the uniq command. The uniq commands reads the sorted list from its stdin and prints the unique words from there to its stdout.

Finally, the output redirection symbol ‘>’ redirects the stdout of the uniq command to the file ‘file2’.

### **More Unix Filter Commands**

| **Command** | **tr – this command is used to translate the characters in a file in some other forms like squeezing the repetitive characters and replacing it with the single occurrence of that character or deleting a character. This can be achieved by using different options available with the tee command.** |
| --- | --- |
| Common Syntax: | tr [OPTION] ……SET1[SET2] |
| Example1: | tr -d ‘,’ < file1 This command will delete all the occurrence of comma(“,”) from the file file1 |
| Example 2 | $ tr -d “hello” < file1 This command will delete all the occurrence of any of the characters h or e or l or o from the file fle1 |
| **Command** | **tee – this command is used to read the input from stdin and redirect the output to the stdout i.e. display screen as well as to the specified files at the same time. It is primarily used in conjunction with pipes and filters.** |
| Common Syntax: | tee[OPTION]….[FILE] |
| Example | $ cat file1|tee filenew This command will display the contents of file1 on the display screen and it will also be saved in filenew which can be viewed by the command cat filenew |

### **Unix sed and awk Text Processing Utilities**

Unix provides sed and awk as two text processing utilities that work on a line-by-line basis. The sed program (stream editor) works well with character-based processing, and the awk program (Aho, Weinberger, Kernighan) works well with delimited field processing.

Both use regular expressions to find patterns and support commands to process the matches.

| **Command** | **awk – this command is a useful and powerful command used for pattern matching as well as for text processing.** |
| --- | --- |
| Common Syntax | awk [options] ‘program text’ file |
| Example | $ls -l | awk ‘{print $3}’ This command will display only the third column from the long listing of files and directories. |
| **Command** | **sed – this is a powerful command for editing a ‘stream’ of text. It can read input from a text file or from piped input, and process the input in one pass..** |
| Common Syntax | sed[OPTION]…..[-f][file] |
| Example1 | sed -n ‘/hello/p’ file1 This command will display all the lines which contains hello |
| Example2 | sed ‘s/hello/HELLO/’ file1 This command will substitute hello with HELLO everywhere in the file. |
| Example3 | sed ‘/hello/,+2d’ file1 This command will delete the two lines starting with the first match of ‘hello’ |