**UNIX COMMANDS​**

***Introduction to vi editor​***

* vi editor is generally used to edit existing files, create new files from scratch or to just read a text file.​
* ***Starting the vi Editor :-​***

The following commands are used to open a file-​

**vi filename** : Creates new file or opens an existing file.**​**

**vi -R filename** : Opens an existing file in read only mode.​

**view filename** : Opens an existing file in read only mode.​

* ***Saving file:-​***

The following commands can be used in command mode for saving the file-​

:q – To quit the vi editor​

:q!- To quit the editor without saving changes​

:w - To save the contents of the editor​

:wq – To Save and quit​

:x – To save and quit​

* ***Moving within a file in vi editor:-​***

k : Moves cursor up one line​

j : Moves cursor down one line​

h : Moves cursor left by a character position​

l : Moves cursor right by a character position​

0 or | : Positions the cursor at the beginning of a line​

$ : Positions the cursor at the end of a line​

* ***Copying lines in vi editor:-​***

yy : Copies the current line.​

yw : Copies the current word from the character the lowercase w cursor is on, until the end of the word.​

P : Puts the yanked text before the cursor. ​

p : Puts the copied text after the cursor.​

* ***Deleting lines in vi editor:-​***

x : Deletes character under the cursor​

X : Deletes the character before the cursor ​

Dd : Deletes the entire line the cursor is on​

* ***Opening a file in insert/edit mode in vi editor:-​***

To edit the file, you need to be in the insert mode. There are many ways to enter the insert mode from the command mode −​

i : Inserts text before the current cursor location​

I : Inserts text at the beginning of the current line​

a : Inserts text after the current cursor location ​

A : Inserts text at the end of the current line​

o : Creates new line below current line​

O : Creates a new line above the current line​

* ***Searching a character or string:-​***

For a string search, the / and ? commands are used. The / command searches forwards (downwards) in the file. The ? command searches backwards (upwards) in the file.​

^ : Searches at the beginning of the line (Use at the beginning of a search expression).​

. : Matches a single character.​

\* : Matches zero or more of the previous character.​

$ : End of the line (Use at the end of the search expression).​

[ : Starts a set of matching or non-matching expressions.​

< : This is put in an expression escaped with the backslash to find the ending or the beginning of a word​

* ***Replacing a string in vi editor:-​***

***Syntax to replace a string****-​*

:s/search/replace/g​

Here, :s/ stands for substitution command​

g indicates globally, i.e to replace all the occurrences of search pattern.​

**Basic file editing commands:​**

***cat command: ​***

* It displays content of the file as output.​

1. To display content of a single file​

**Syntax:** cat filename​

2. To view multiple files​

**Syntax:** cat file1 file2​

3. To view contents of the file along with line numbers for each line​

**Syntax:** cat -n filename​

4. To create a new file​

**Syntax:** cat > newfile​

5. Copy contents of one file to another​

**Syntax:** cat filename-whose-content-to-copy > destination-filename

6. To suppress repeated empty lines in output

**Syntax:** cat -s filename

7. ​To append content of one file to end of another file

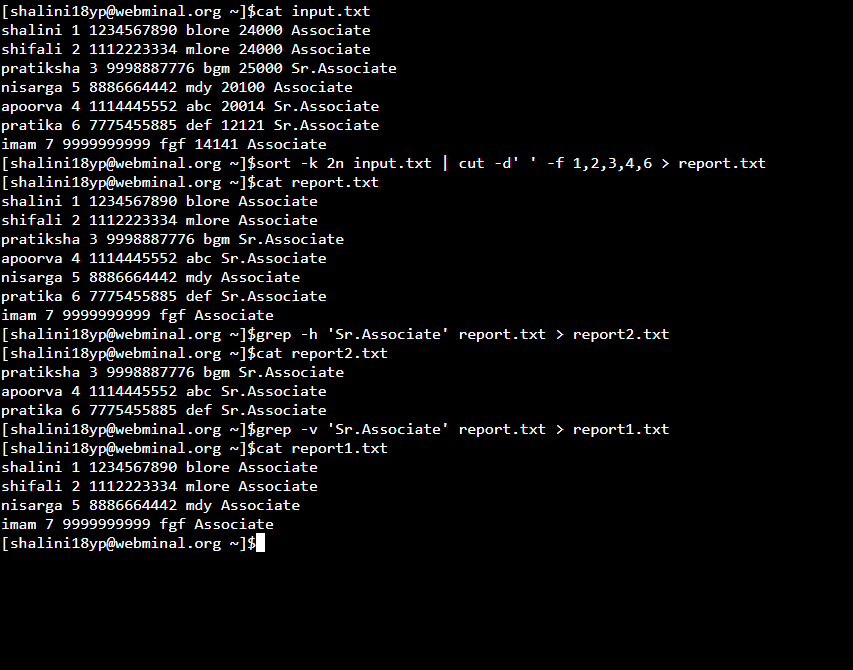
**Syntax:** cat file1 >> file2

8. To display content of file in reverse order

**Syntax:** tac filename

9. To highlight the end of line

**Syntax:** cat -E filename

****

***mv Command***

* mv stands for move.
* It can be used to move one or more files or directories from one place to another in a filesystem.
* mv command is mainly used to perform two functions:

a. To move group of files or directories

b. To rename files or folders.

* **Syntax:**

mv [options] source destination

* **Options:**

1. -i : If destination file already exists, then it asks for user confirmation before overwriting.

mv -i a.txt b.txt

mv: overwrite 'b.txt'? y

2. -f(Force) : mvprompts for confirmation while overwriting the destination file if the file is write protected. The **-**f option overrides this minor protection and overwrite the destination file forcefully and delete the source file.

mv -f a.txt b.txt

3. -n (no-clobber): With **-**n option, mv prevent an existing file from being overwritten.  
 mv -n geek.txt b.txt

4. -b(Backup) : This will create a backup of the existing file which will be overwritten as a result of mv command.

mv -b a.txt b.txt

ls : b.txt b.txt~ c.txt d.txt

***ls Command***

* ls command lists the directory content of files and directories.​
* The various options with ls are:​

1. –t : It will open the last edited file. Sorts the file by last modification time, showing last edited file first. head -1 picks up this file.​

**Example**: ls –t | head -1​

2. -1: To display one file per line.​

**Example**: ls -1​

**Output**: a.out​

sample.txt​

1.txt ​

3. –l : Displays all information about the files / directories​

**Example:** ls –l 1.txt​

**Sample Output:​**

-rw-rw-r– 1 maverick maverick 1047 Oct 31 13:15 1.txt​

This output contains 7 fields:​

1st Character – ***File Type***: First character specifies the type of the file.​  
 – normal file d : directory s : socket file l : link file​

* ***Field 1 – File Permissions:*** Next 9 character specifies the files permission. The every 3 characters specifies read, write, execute permissions for user(root), group and others respectively in order.​
* ***Field 2 – Number of links:*** Second field specifies the number of links for that file. In this example, 1 indicates only one link to this file.​
* ***Field 3 – Owner:*** Third field specifies owner of the file. In this example, this file is owned by username ‘maverick’.​
* ***Field 4 – Group:*** Fourth field specifies the group of the file. In this example, this file belongs to ”maverick’ group.​
* ***Field 5 – Size:*** Fifth field specifies the size of file in bytes. In this example, ‘1047’ indicates the file size in bytes.​
* ***Field 6 – Last modified date and time:*** Sixth field specifies the date and time of the last modification of the file. In this example, ‘Oct 31 13:15’ specifies the last modification time of the file.​
* ***Field 7 – File name:*** The last field is the name of the file. In this example, the file name is 1.txt.​

4. –ld: To display directory information​

**Example:** ls –ld /etc​

Displays drwxr-xr-x 132 root root 12288 May 28 04:14 /etc​

5. –lt: Order file based on last modified time​

**Example**: ls –lt ​

To print the files based on modification time in the reverse order we can use –r option​

**Example:** ls –ltr​

6. –a: Display hidden files. Hidden files in Unix starts with ‘.’ in its file name.It will show all the files including the ‘.’ (current directory) and ‘..’ (parent directory).​

**Example**: ls –a​

ls –A : To show the hidden files, but not the ‘.’ (current directory) and ‘..’ (parent directory).​

7. –R : Display files recursively using ls –R​

Shows all unhidden files in the system recursively.​

8. –i: ls –i displays file inode number​

9. –n: ls –n displays file userid and groupid instead of user and group names.​

***grep Command***

* grep stands for globally search for regular expression and print out.​
* It searches a file for a particular pattern of characters and displays all the lines that contain that pattern.​
* Pattern used in grep is referred to as regular expression.​
* **Syntax:​**

grep [options] pattern [files]​

* The various options used with grep are: ​

1. –c : This option prints only a count of lines that match a pattern.​

**Example-** grep –c “unix” file1.txt​

This command will only print the count of lines in which the pattern ​unix is matched ​

2.-i : ignores case while matching the pattern.​

**Example:** grep –I “UNix” file1.txt ​

It matches the words like “UNIX”, “Unix”, “unix”. ​

3. –n : Display matched lines and their line numbers . This option ​

enables us to show line numbers while displaying.​

**Example:** grep –n “unix” file1.txt​

Output for this will be like this​

1:unix is great os. ​

4:unix is easy to learn. ​

4.-v : This prints all the lines that do not match the pattern​

**Example:** grep –v “unix” file1.txt​

5.-w : matches only whole words. The grep command by default will match a word​ even if it is found as a substring.​

**Example:** grep –w “unix” file1.txt​

6. –o : Displays only the matched string.​

**Example:** grep –o “unix” file1.txt​

Displays – unix​

unix​

unix​  
 if there are three matches to the pattern unix.​

7. ^ : Matching lines that start with a string.​

**Example:** grep “^unix” file1.txt​  
 8. $ : Matching lines that end with a string.​

**Example:** grep “os$” file1.txt​

9. –e : Specifies expression with –e option. Can use multiple times.​

**Example:** grep –e “Agarwal” –e “Aggarwal” –e “Agrawal” file1.txt​

10. –f : Takes pattern from file one per line​

**Example:** cat pattern.txt​

Agarwal​

Aggarwal​

Agrawal​

grep –f pattern.txt file1.txt​

***sed Command***

* sed stands for stream editor.
* sed command can be used for modifying files.​

**Syntax:** sed OPTIONS... [SCRIPT] [INPUTFILE...] ​

1.Replacing or substituting a string​

sed ‘s/unix/linux/’ file.txt​

Here, “s” specifies the substitution operation. The “/” are delimiters. The “unix” is the search pattern and the “linux” is the replacement string. By default, the sed command replaces the first occurrence of the pattern in each line and it won’t replace the second, third…occurrence in the line. ​

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

sed 's/unix/linux/2' file.txt ​

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

sed 's/unix/linux/g' file.txt ​

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

sed 's/unix/linux/3g' file.txt ​

5. Replacing string on a specific line number ​

sed '3 s/unix/linux/' file.txt​

6. Duplicating the replaced line with /p flag ​

The /p print flag prints the replaced line twice on the terminal. If a line does not have the search pattern and is not replaced, then the /p prints that line only once.​

sed 's/unix/linux/p' file.txt​

7. Replacing string on a range of lines​

sed '1,3 s/unix/linux/' file.txt​

8. Deleting lines from a particular file ​

a. To Delete a particular line say n​

**Syntax:** sed 'nd' filename.txt ​

**Example:** sed '5d' filename.txt​

b. To Delete a last line​

**Syntax:**  sed '$d' filename.txt​

c. To Delete line from range x to y​

**Syntax:** sed 'x,yd' filename.txt ​

**Example:** sed '3,6d' filename.txt​

d. To Delete from nth to last line​

**Syntax:** sed 'nth,$d' filename.txt ​

**Example:**  sed '12,$d' filename.txt​

e. To Delete pattern matching line​

**Syntax:** sed '/pattern/d' filename.txt ​

**Example:** sed '/abc/d' filename.txt​

***awk Command***

* Scripting language used for manipulating data and generation of reports.​
* awk does not require compiling​.
* Tiny but effective programs​.
* Mainly used for pattern scanning and processing​.
* **Operation​**

(a) Scans a file line by line​  
 (b) Splits each input line into fields​  
 (c) Compares input line/fields to pattern​  
 (d) Performs action(s) on matched lines​

* **Syntax:** awk options ‘selection\_criteria {action}’ inputfile > outputfile​
* **Options:** ​

-f program-file : Reads the AWK program source from the file program-file, instead of from the first command line argument. ​

-F fs : Use fs for the input field separator ​

* Default behavior of awk is that it prints every line of data from specified file​

**Example:** awk ‘{print}’ employees.txt​

**Built In Variables In Awk​**

* Awk’s built-in variables include the field variables—$1, $2, $3, and so on ($0 is the entire line) — that break a line of text into individual words or pieces called fields.​
* **NR:** NR command keeps a current count of the number of input records. Remember that records are usually lines. Awk command performs the pattern/action statements once for each record in a file.​
* **NF:** NF command keeps a count of the number of fields within the current input record.​
* **FS:** FS command contains the field separator character which is used to divide fields on the input line. The default is “white space”, meaning space and tab characters. FS can be reassigned to another character (typically in BEGIN) to change the field separator.​
* **RS:** RS command stores the current record separator character. Since, by default, an input line is the input record, the default record separator character is a newline. ​
* **OFS:** OFS command stores the output field separator, which separates the fields when Awk prints them. The default is a blank space. Whenever print has several parameters separated with commas, it will print the value of OFS in between each parameter.​
* **ORS:** ORS command stores the output record separator, which separates the output lines when Awk prints them. The default is a newline character. print automatically outputs the contents of ORS at the end of whatever it is given to print.​
* Example that finds difference between two files using awk built-in variable NR

***chmod Command***

* chmod command is used to change the access mode of a file.
* chmod stands for change mode.
* **Syntax :** chmod <action> <-filename->​
* **Numeric mode :** the permissions are represented by number from 0 to 7.

1 (001) ,2(010) and 4(100) represents execute ,write and read respectively.​

**Example:** chmod 777 file.txt​

This gives read write and execute(1+2+4) to the user ,group and

others.

* **Symbolic mode :​** The characters r,w and x represents read write and execute respectively and u,g,o and a represents user group others and all group respectively.​

The operators +,- and = is used to add ,remove and equate the permissions specified.​

**Example:** chmod go+x filename.txt ​

***ln Command***

* Used for creating a link or pointer to a file.
* **Syntax:**

ln filename linkname

* There are two types of link:

1. Hard link:

- Contains the actual content of the file.

- Has same inode number as that of the file.

- Deleting original file will not affect the hard link.

- **Synatx:** ln filename linkname

2. Soft link:

- Contains path of the original file.

- Has different inode number.

- Similar to shortcuts created in windows.

- Deleting original file will create a dangling pointer

- **Syntax:** ln -s filename linkname

***touch Command***

* touch command is used to create, change and modify timestamps of files.
* It is used to create files without any content.
* The file created with this command is empty.
* This command is mainly used when a user does not have data to store at the time of file creation.
* **Syntax:** touch filename
* Touch command to create multiple files:​

**Syntax:​**

touch filename1 filename2 filename3​

* Options used:​

1. touch -a: This command is used to change access time only. To change or update the last access or modification times of a file touch -a command is used.​

**Syntax:​**

touch -a filename​

2. touch -c : This command is used to check whether a file is created or not. If not created then don’t create it. This command avoids creating files.​

**Syntax​:**

touch -c fileName​

3. touch -c-d : This is used to update access and modification time.​

**Syntax:​**

touch -c-d filename​

4. touch -m : This is used to change the modification time only. It only updates last modification time.​

**Syntax:​**

touch -m filename​

5. touch -r : This command is used to use the timestamp of another file.

**Syntax:​**

touch -r second\_file\_name first\_file\_name​

6. touch -t : This is used to create a file using a specified time.​

**Syntax:​**

touch -t YYMMDDHHMM fileName​

***date Command***

* datecommand is used to display the system date and time.
* It is also used to set the system date and time.
* **Syntax:**

date [OPTION]... [+FORMAT]

* **Options:**

1. date (no option): With no options, the date command displays the current date and time, including the abbreviated day name, abbreviated month name, day of the month, the time separated by colons, the time zone name, and the year.

**Syntax: $** date

**Output:** Wed Nov 20 05:51:01 CET 2019

2. -u : Displays the time in GMT(Greenwich Mean Time)/UTC(Coordinated Universal Time )time zone.

**Syntax:** date -u

**Output:** Wed Nov 20 04:53:10 UTC 2019

3. -s or –set Option: To set the system date and time -s or –set option is used.  
 **Syntax:** $date --set="date to be set"

4. -r option: This is used to display the last modified timestamp of a datefile .

**Syntax:** $date -r file.txt

5. **List of Format specifiers used with date command:**

%D: Display date as mm/dd/yy.

%d: Display the day of the month (01 to 31).

%a: Displays the abbreviated name for weekdays (Sun to Sat).

%A: Displays full weekdays (Sunday to Saturday).

%h: Displays abbreviated month name (Jan to Dec).

%b: Displays abbreviated month name (Jan to Dec).

%B: Displays full month name(January to December).

%m: Displays the month of year (01 to 12).

%y: Displays last two digits of the year(00 to 99).

%Y: Display four-digit year.

%T: Display the time in 24 hour format as HH:MM:SS.

%H: Display the hour.

%M: Display the minute.

%S: Display the seconds.

**Syntax:** $date +%[format-option]

***tr Command***

* The tr command in UNIX is a command line utility for translating or deleting characters.
* tr stands for translate.

**Syntax :**

$ tr [OPTION] SET1 [SET2]

* **Options:**

1. To convert uppercase to lowercase

cat filename | tr "[A-Z]" "[a-z]"

OR

cat filename | tr “[:upper:]” “[:lower:]”

2. To squeeze repitition of characters using -s

echo "Welcome To Mainframes" | tr -s [:space:] ' '

Output: Welcome To Mainframes

3. To delete specific characters using -d

echo "Welcome To Mainframes" | tr -d 'w'

Output: elcome To Mainframes

4. To complement sets using -c

For example, to remove all characters except digits, you can use the following.

echo "my ID is 73535" | tr -cd [:digit:]

Output: 73535

***cal Command***

* cal command is a calendar command in Linux which is used to see the calendar of a specific month or a whole year.
* By default, cal command shows current month calendar as output.
* **Syntax:**

cal [ [ month ] year]

* Rectangular bracket means it is optional, so if used without option, it will display a calendar of current month and year.
* **cal :** Shows current month calendar on the terminal.
* **cal 07 2019 :** Shows calendar of selected month and year.
* **cal 2019 :** Shows the whole calendar of the year.
* **cal -3 :** Shows calendar of previous, current and next month

***Data Redirection***

1. Output Redirection: The output from a command normally intended for standard output can be easily diverted to a file instead. This capability is known as output redirection.The **greater-than character >** is used for output redirection.  
    cat userfile > outputfile
2. Input Redirection: Just as the output of a command can be redirected to a file, so can the input of a command be redirected from a file. The **less-than-character** < is used for input redirection.  
    wc -1 < users
3. Here Document: A **here document** is used to redirect input into an interactive shell script or program.  
    command << delimiter  
    document  
    delimiter
4. Discard the output: you can discard the output by redirecting it to the file **/dev/null**

command > /dev/null

Following is a complete list of commands which you can use for redirection-

1. pgm > file

Output of pgm is redirected to file

2. pgm < file

Program pgm reads its input from file

3. pgm >> file

Output of pgm is appended to file

4. n > file

Output from stream with descriptor n redirected to file

5. n >> file

Output from stream with descriptor n appended to file

6. n >& m

Merges output from stream n with stream m

7. n <& m

Merges input from stream n with stream m

8. << tag

Standard input comes from here through next tag at the start of line

9. **|**

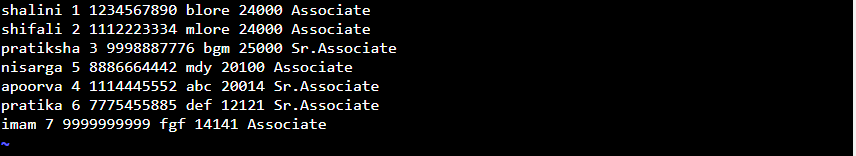
Takes output from one program and sends it to another

***File Reading Commands***

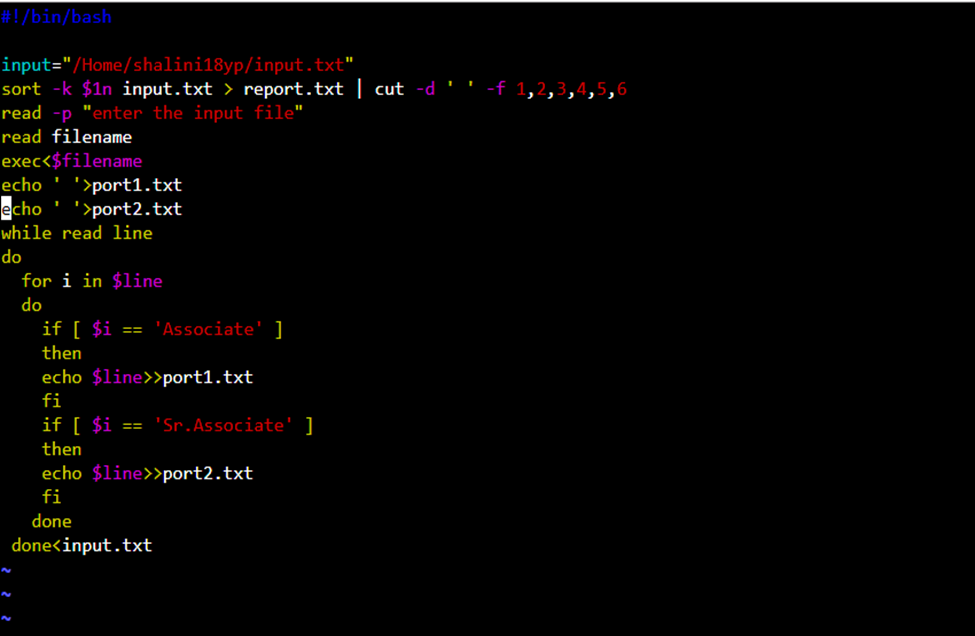
* To read files line by line in unix, we can use while loop and for loops.
* Example- reading a file using while and for loop

*ASSIGNMENT*

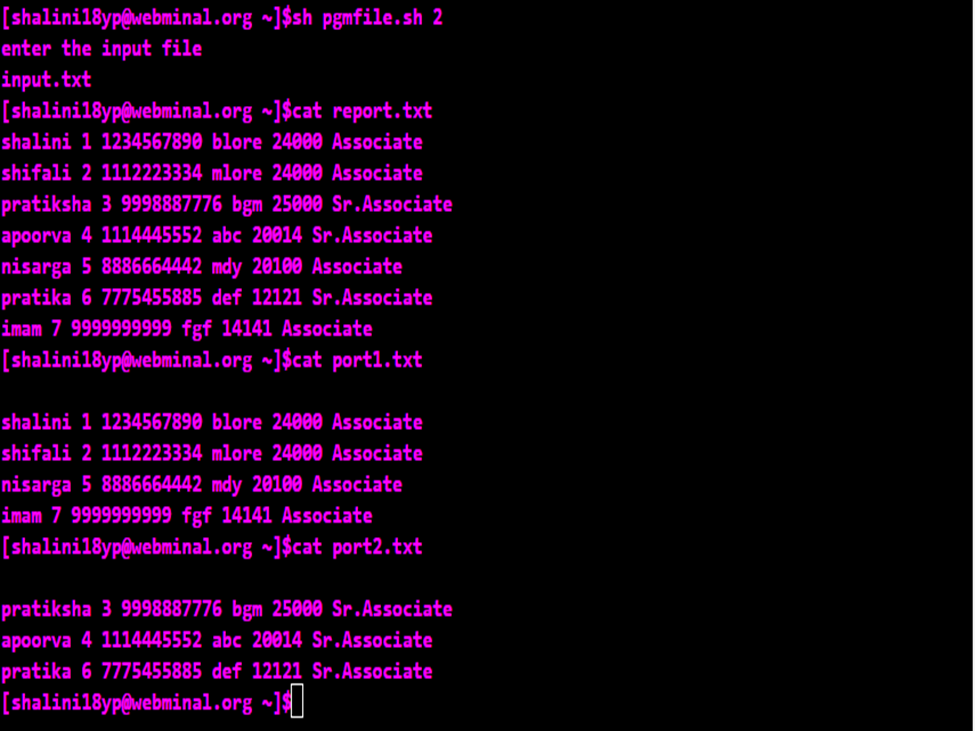
Input file



Shell Script



Output



***Shell Loops***

***if then else***This block will process if specified condition is true.  
**Syntax:**

if [ expression ]

then

statement

fi

The statement inside if block will get executed only if the expression given in if is true.

if [ expression ]

then

statement

else

statement

fi

If the expression given in if is true then the statement inside if will be executed, but if the expression in if is false then the statement given under else will be executed.

***if..elif..else..fi***

* To use multiple conditions in one if-else block, then elif keyword is used in shell.
* If expression1 is true then it executes statement 1 and 2, and this process continues. If none of the condition is true then it processes else part.
* Syntax:

if [ expression1]

then

statement1

statement2

elif [ expression2 ]

then

statement3

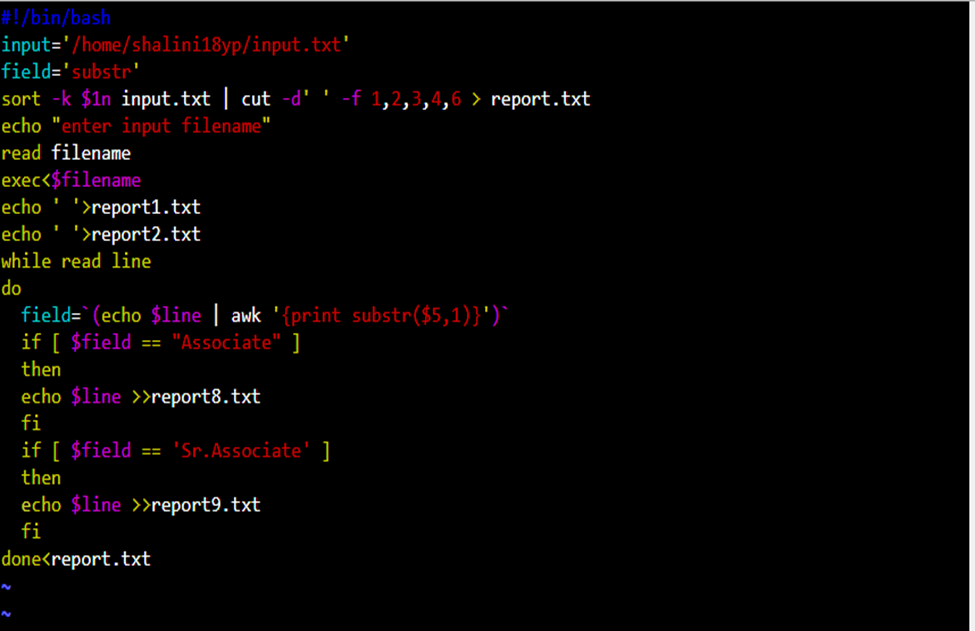
statement4

else

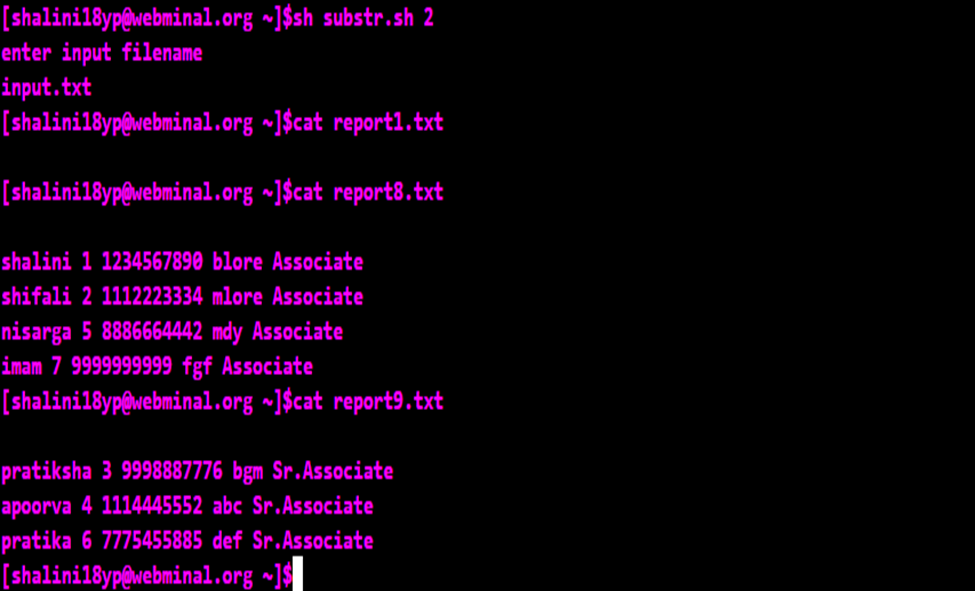
statement5

fi

* Example:



* Output:



***Shell for loop:***

* The for loop operate on lists of items.
* It repeats a set of commands for every item in a list.
* Syntax:

for var in word1 word2 ...wordn

do

Statement to be executed

done

* Example:

for a in 1 2 3 4 5 6 7 8 9 10

do

# if a is equal to 5 break the loop

if [ $a == 5 ]

then

break

fi

# Print the value

echo "Iteration no $a"

done

* Output:

Iteration no 1

Iteration no 2

Iteration no 3

Iteration no 4

***Shell while loop***

* A conditional statement is given with while and based on the condition while loop is executed.
* While the condition is true, the statements inside the while loop are executed.
* Once the condition is false, the loop is stopped and control comes out of the loop.
* Syntax:

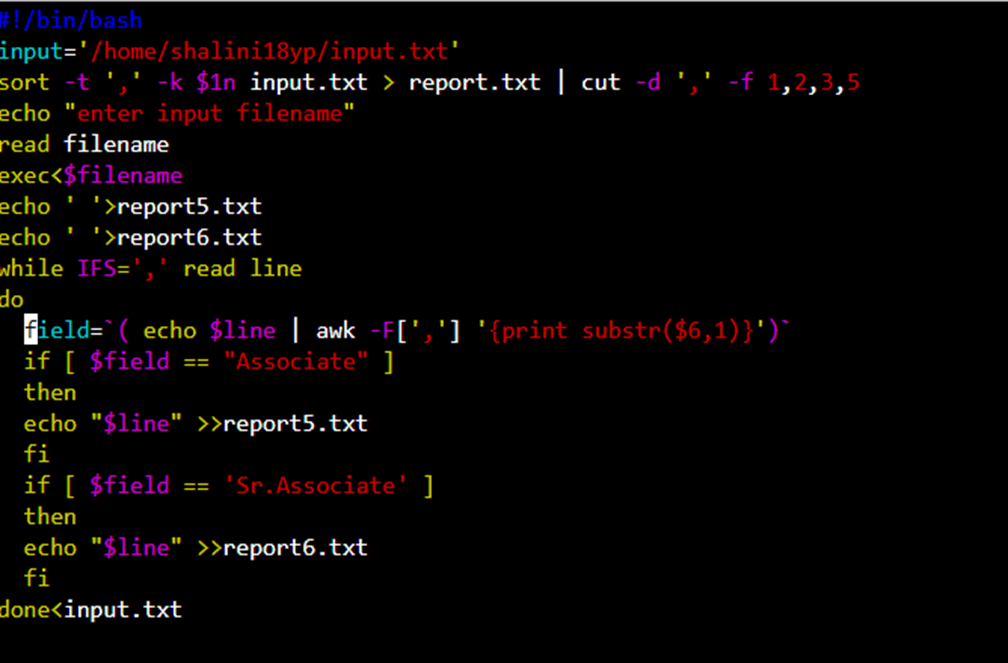
while command

do

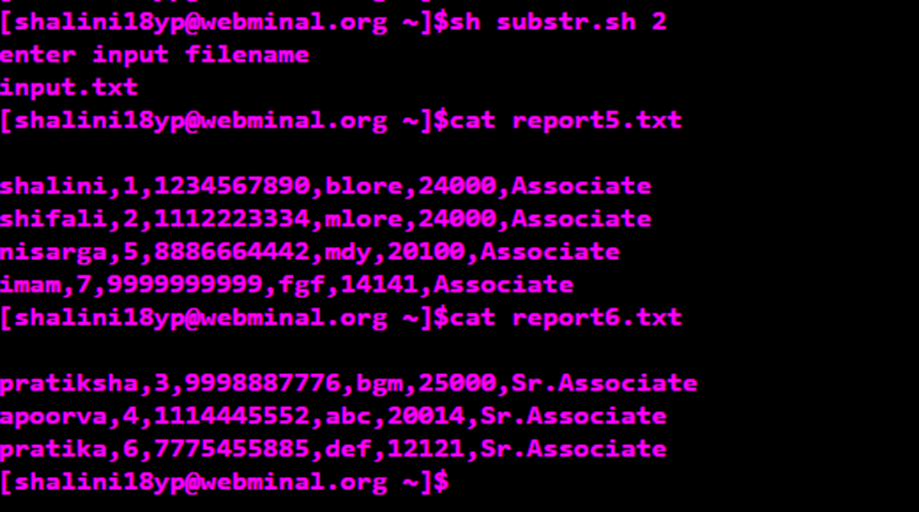
Statement to be executed

done

* Example:



* Output:



***Shell until loop***

* Until command in Linux used to execute a set of commands as long as the final command in the ‘until’ Commands has an exit status which is not zero. It is mostly used where the user needs to execute a set of commands until a condition is true.​
* Syntax:​

until COMMANDS; do COMMANDS; done​

​

* Here, if the COMMANDS get evaluated to false then the statements will be executed. If the COMMANDS get evaluated to true then the no statements will be executed and control will go after the done statement.​

***Shell Functions***

* Functions enable you to break down the overall functionality of a script into smaller, logical subsections, which can then be called upon to perform their individual tasks when needed.
* Shell functions are similar to subroutines, procedures, and functions in other programming languages.
* Creating Functions:

Syntax to declare a function-

function\_name ( ) {

list of commands

}

The name of function is **function\_name**, and we use the same name to call it from elsewhere in your scripts.

The function name must be followed by parentheses, followed by a list of commands enclosed within braces.

Example:

# Define your function here

Hello () {

echo "Hello World"

}

# Invoke your function

Hello

* Pass Parameters to a Function:

You can define a function that will accept parameters while calling the function. These parameters would be represented by **$1**, **$2** and so on.

For the same example above, we can pass the parameter while invoking the function as below-

Hello hi world

* Returning values from a function:

If you execute an **exit** command from inside a function, its effect is not only to terminate execution of the function but also of the shell program that called the function.

We can return any value from your function using the **return** command whose syntax is as follows −

return code

* Nested Functions:

A function can call another function, this feature is known as nesting of

functions.

If a function calls itself then it is known as recursive function.

Example-

#First function declaration

function1 () {

echo "Hi"

#invoking second function

function2

}

function2 () {

echo "Hello"

}

#invoking function1

function1

Output-

Hi Hello

***case Statement***

* case....esac statement is used to perform multiway branch in unix.
* Syntax:

case word in

pattern1 )

Statement to be executed if pattern1 matches

;;

pattern2 )

Statement to be executed if pattern2 matches

;;

pattern3 )

Statement to be executed if pattern3 matches

;;

\* )

Default condition to be executed

;;

esac

* Here the string word is compared against every pattern until a match is found.
* The statement following the matching pattern executes.
* If no matches are found, the case statement exits without performing any action.
* Example-

fruit="kiwi"

case "$fruit" in

"apple" )

echo " A for apple"

;;

"mango" )

echo "Mango is king of fruits"

;;

"kiwi" )

echo "New Zealand is famous for kiwi"

;;

esac

* Output-

New Zealand is famous for kiwi

***eval Statement***

* The eval command is used to execute the arguments as a shell command on unix or linux system.
* Eval command comes in handy when you have a unix or linux command stored in a variable and you want to execute that command stored in the string.
* The eval command first evaluates the argument and then runs the command stored in the argument.
* Example-

Command="ls -ltr"

eval $Command

Here, the command ls -ltr is evaluated and the output of that command is given.

***let Statement***

* The **let** command is an arithmetic operator
* It is almost same as (( )).
* Difference is that, let is an arithmetic command while (( )) is a compound command.
* It is a built-in command which instructs shell to perform an evaluation of arithmetic expressions.
* No spaces should be used around the arithmetic operant with **let** command.
* Example-

let y="234+58"

echo $y

* Output-

292

***Shell Reading and Handling User Input***

* User input can be taken in two ways:

1. By passing input as command line arguments.

2. By reading input from promt and using read command to fetch it.

* Command Line Arguments:

In general the command line arguments are passed to the bash or shell script to change the behavior of the script.

Example-

Script- cat shell.sh

echo "Script execution starts"

echo "$@"

echo "$0"

echo "$1"

echo "$2"

echo "$#"

echo "Script execution ends"

Running the Script- shell.sh hi hello

Here, hi hello are command line arguments

Output-

Script execution starts

hi hello

shell.sh

hi

hello

2

Script execution ends

Explanation of $ variables:

a. $@ : contains all the arguments

b. $0 : contains script name

c. $1 : First argument

d. $2 : Second argument

e. $n : Nth argument

f. $# : Count of arguments passed.

* Input from the Prompt:

read command is used to get input from the user.

Syntax:

read -p "Prompt" variable1 variable2 variableN

-p "Prompt" - Display prompt to user without a newline.

variable1 - The first input (word) is assigned to the variable1.

variable2 - The second input (word) is assigned to the variable2.

Handling the input:

Example- cat shell1.sh

read -p "Enter your name : " name

echo "Hi, $name. Let us be friends!"

The variable read can be used along with $ sign.

Running the script: sh shell1.sh

Output: Enter your name : Sunethra

Hi, Sunethra. Let us be friends!

***File Handling Commands***

***ftp Command***

* The FTP (File Transfer Protocol) utility program is commonly used for copying files to and from other computers.
* These computers may be at the same site or at different sites thousands of miles apart.
* To connect your local machine to the remote machine, type ​

ftp machinename ​

where machinename is the full machine name of the remote machine.

* If the name of the machine is unknown, you may type ​

ftp machinennumber ​

where machinennumber is the net address of the remote machine.

* If the remote machine has been reached successfully, FTP responds by asking for a login name and password. ​
* When you enter your own login name and password for the remote machine, it returns the prompt ​

ftp> ​

* Then permits you access to your own home directory on the remote machine.
* You should be able to move around in your own directory and to copy files to and from your local machine using the FTP interface commands.​
* EX: > ftp remote-server-name​

connected to remote-server-name​

User-Name:​

Password:​

ftp>

**Anonymous FTP​**

* At times you may wish to copy files from a remote machine on which you do not have a loginname. This can be done using anonymous FTP. ​
* EX: > ftp remote-server-name​

connected to remote-server-name​

User-Name: anonymous​

Password: electronic mail address​

ftp>​

* This allows the remote site to keep records of the anonymous FTP requests. ​
* Once you have been logged in, you are in the anonymous directory for the remote machine. This usually contains a number of public files and directories. Again you should be able to move around in these directories. However, you are only able to copy the files from the remote machine to your own local machine; you are not able to write on the remote machine or to delete any files there. ​
* Copy file from remote machine to local machine.​

The get option is used to download or transfer a file from the remote system to the local system.​

ftp> get program1.bat​

* Copying multiple files from remote machine to local machine.​

You can use the mget to transfer multiple files from the remote host to local host.​

ftp>mget \*.png​  
This will download all the png images to the local machine.​

* Transferring file from local server to remote server​

The put option is used to copy the file from the local host to the remote host.​  
 ftp>put programs.rpm ​

* Transferring multiple files to the remote server.​

You can use the mput option to transfer more than one file from local system to the remote system.​

ftp> mput \*.rpm​

* Executing commands in remote machine.​

After connecting to the remote network machine using the ftp, you can run commands like ls to list the files, cd to change directory and many more.​

ftp> ls​

This will list the files and directories in the remote machines current directory.​

* Executing commands in local machine.​

To run the commands on local machine you need to exit from the ftp connection. Instead of this, there is a way to run commands on local host without exiting from the ftp connection. Use the ! symbol before the command you want to run.​

ftp> !ls​

* Changing the file transferring mode.​  
  You can change the file transfer modes to ascii and binary modes. Use the below commands to change the mode.​  
   ftp>ascii ​  
   ftp>binary​
* Deleting files on remote machine​  
  You can use the delete or mdelete to remove a single file or multiple files in the remote machine.​

ftp>delete pgm1.dat​

ftp>mdelete \*.dat​

* Disconnecting from ftp connection.​  
  Use the quit command to close the ftp connection.​

ftp>quit​

* Getting the help about ftp command.​  
  To know more about the ftp command, just type the help on the prompt. It will display the options/commands that you can use with ftp command.​  
   ftp>help​

***cp Command***

* **cp** stands for **copy**.
* This command is used to copy files or group of files or directory.
* It creates an exact image of a file on a disk with different file name.
* cpcommand require at least two filenames in its arguments.
* Syntax:

To copy Source file to Destination file

cp [OPTION] Source Destination

To copy Source file to Directory

cp [OPTION] Source Directory

To copy multiple Source files to Directory.

cp [OPTION] Source-1 Source-2 Source-3 Source-n Directory

* Options:

1. -i (interactive) :

a. i stands for Interactive copying.

b. With this option system first warns the user before overwriting the destination file.

c. cp prompts for a response, if you press **y** then it overwrites the file and with any other option leave it uncopied.

Example:

cat a.txt

hi

cat b.txt

hello

cp -i a.txt b.txt

cp: overwrite 'b.txt' ? y

cat b.txt

hi

2. -b (backup) :

With this option **cp** command creates the backup of the destination file in the same folder with the different name and in different format.

Example: cp -b a.txt b.txt

ls

a.txt b.txt b.txt~

3. -f (force) :

If the system is unable to open destination file for writing operation as user doesn’t have writing permission for this file then by using **-f** option with **cp** command, destination file is deleted first and then copying of content is done from source to destination file.

Example:

ls -l b.txt

-r-xr-xr-x+ 1 User User 3 Nov 24 08:45 b.txt

User, group and others doesn't have writing permission.

Without -f option, command is not executed

$ cp a.txt b.txt

cp: cannot create regular file 'b.txt': Permission denied

With -f option, command executed successfully

$ cp -f a.txt b.txt

4. -r or -R :

With this option **cp** command shows its recursive behavior by copying the entire directory structure recursively.

Example:

ls etc/

a.txt b.txt b.txt~ Folder1 Folder2

Without -r option, error

$ cp etc gfg

cp: -r not specified; omitting directory 'etc'

With -r, execute successfully

$ cp -r etc gfg

$ ls gfg/

a.txt b.txt b.txt~ Folder1 Folder2

5. -p (Preserve) :

The time of the last data modification and the time of the last access, the ownership (only if it has permissions to do this), and the file permission-bits, these characters are preserved when we use -p option along with cp.

Syntax: cp -p a.txt c.txt

***sort Command***

* SORT command is used to sort a file, arranging the records in a particular order.
* By default, the sort command sorts file assuming the contents are ASCII (ascending order ).
* SORT command sorts the contents of a text file, line by line.
* Blank space is the default field separator.
* Syntax: sort filename
* Options:

1. -o : This option is used to redirect output to a file

sort -o outputfile inputfile

2. -r : To reverse the sorting order ( descending order by default )

sort -r filename

3. -n : To sort file numerically

sort -n filename

4. -nr : To sort a file numerically in reverse order

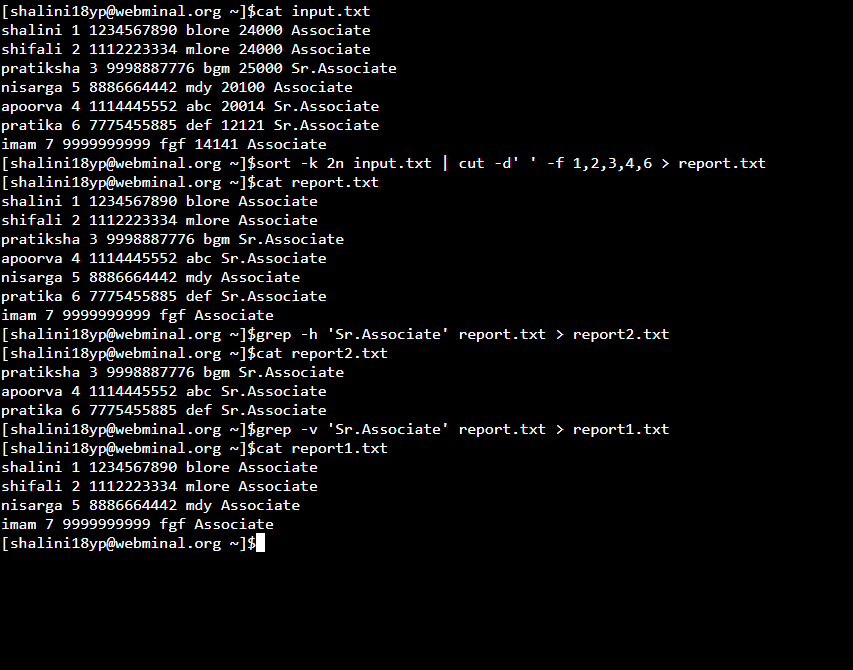
sort -nr filename

5. -k : To sort on the basis of a column number

sort -k 2 filename

This will sort on the basis of second column

Example:

****

6. -c : This is used to check if the file given for sorting is already sorted

This will write to standard output if there are lines that are out of order.

sort -c filename

7. -u : To sort as well as to remove duplicates from the given file

sort -u filename

8. -M : To sort the given file by month

sort -M filename

***cut Command***

* Used for cutting out the sections from each line of files and writing the result to standard output.​
* Syntax:​

cut OPTION... [FILE]...​

* Options:​

cat fruit.txt

apple

guava

musk melon

banana

1. -b(byte): ​

To extract the specific bytes, -b option with the list of byte numbers separated by comma.

Range of bytes can also be specified using the hyphen(-).

Tabs and backspaces are treated like as a character of 1 byte.

Example:

cut -b 1,2,3 fruit.txt

app

gua

mus

ban

2. -c (column):​

To cut by character use the -c option. This selects the characters given to the -c option.​

Syntax:​

cut -c [(k)-(n)/(k),(n)/(n)] filename

Example:

cut -c 1,3,5 fruit.txt

ape

gaa

ms

bnn

3. -f (field):​

cuts based on the fields

Syntax:​

cut -d "delimiter" -f (field number) file.txt ​

Example:

cut -d " " -f 2 fruit.txt

apple

guava

melon

banana

4. –complement:​

It complements the output.

Example:

cut --complement -d " " -f 1 fruit.txt

apple

guava

melon

banana

cut --complement -c 5 fruit.txt

​ appl

guav

muskmelon

banaa

5. –output-delimiter: ​

By default the output delimiter is same as input delimiter that we specify in the cut with -d option. To change the output delimiter use the option –output-delimiter=”delimiter”.​

Example:

cut -d " " -f 1,2 fruit.txt --output-delimiter='%' ​

apple

guava

musk%melon

banana

***locate Command***

* Find files by name.​
* Better and faster than FIND command.​
* This command searches the database for the file and not the file structure.
* Syntax: locate [option] [pattern]
* Exit Status :

exits with 0 exit status if search is succesful​

exits with status 1 is unsuccessful or if any error occurs​

* Options:

1. -c :displays the count of the files matching the pattern instead of listing the file.​

2. -i :ignore case in the pattern​

3. -e :display file existing in the system .​

locate produces results of files whose physical copies are deleted(whenever the database is not updated),to avoid this –e is used.​

4. -l :exits successfully after finding LIMIT entries.​

5. -d :change mlocate db location​

Default db location is /var/lib/mlocate/mlocate.db but if you wish to link locate to some other db in other loc use –d.​

Locate –d <newpath><filename>​

***find Command***​

* Searches through hierarchy finds all the files in directory & sub directory .​
* Syntax: find <search from> [options] <filename>​
* Options:

1. -i :ignore case​

2. -maxdepth : limit the search to few level of file structure.​

find –maxdepth 2 name “mypgm.c”​

This will search 2 levels of directories from the root directory.​

3. -mindepth : limits the search to in between levels of directory.​

find -mindepth 2–maxdepth 4 name “mypgm.c”​

This will search from level 2 to level 4 of directories.​

4. -not : files that donot match are displayed.​

5. -empty : search for empty file​

find ./ -empty​

6. -perm : search on the basis of file access permission​

find / -perm 477​

7. -inum n : finds the file with inode number n.​

***join Command***

* The join command in UNIX is a command line utility for joining lines of two files on a common field.
* join command is used to join the two files based on a key field present in both the files.
* Syntax:

join [options] filename1 filename2

* Options:

cat file1.txt

1 Udupi

2 Bangalore

3 Mangalore

4 Mumbai

5 Hubli

cat file2.txt

1 100

2 200

3 300

4 400

1. -a FILENUM:

join command by default prints only the pairable lines in the files, but if we want to print tghe unpairable lines as well then this option can be used.

Example:

join file1.txt file2.txt -a 1

1 Udupi 100

2 Bangalore 200

3 Mangalore 300

4 Mumbai 400

5 Hubli

2. -v :

To print only the unpairable lines.

Example:

join file1.txt file2.txt -v 1

5 Hubli

3. -1, -2 and -j option:

By default, join takes the first column of both the files as key

column for joining.

But if we want to take some other column as key then we can

use -1 option to indicate key column of first file and -2 to

indicate key for 2nd file.

If both the files have same key column then we can use -j option

Example:

join -1 2 -2 2 filename1 filename2

join -j2 filename1 filename2

4. -i :

This option ignores the case while joining

join -i file1 file2

5. --nocheck-order:

By default, join checks if the files are in sorted order and reports if its not.This option can be used to remove the warning

join --nocheck-order file1 file2

6. -t :

This option is used to specify delimiters.

join -t, file1.txt file2.txt

1,Udupi,100

2,Bangalore,200

3,Mangalore,300

4,Mumbai,400

***mail Command***

* The Mail command is used to send emails to the users, to read the received emails, to delete the emails etc.​
* Syntax:

mail [options] to-address [-- sendmail-options]​

* Options:

1. -v (Verbose mode) : Delivery details are displayed on the terminal.​

2. -s : Specify the subject of the mail​

3. -c : Send carbon copies of the mail to the list of users. This is like cc option in Microsoft outlook.​

4. -b : Send blind copies of the mail to the list of users. This is like bcc option in outlook.​

5. -f : Read the contents of the mailbox​

6. -r : Specify the from address in send mail options.​

* Examples - Sending Emails:​

1. Sending sample email to user​

echo "Mail body" | mail -s "Mail subject“ to@example.com​

2. Specifying the body in a file​

Using cat statement:​

cat body.txt | mail -s "Mail subject" to@example.com​

Using input redirection operator​

mail -s "Mail subject" to@example.com < body.txt​

3. Using the cc and bcc option You can copy the emails to more number of users by using the -c and -b options. An example is shown below: ​

mail -s "Mail subject" -c"ccuser@gmail.com" -bbc"bccuser@yahoo.com" "user@example.com" < body.txt​

4. Specifying the from address So far the above examples send the emails with from address as the logged in user. You can explicitly specify the from-address using the -r option. ​

cat body.txt | mail -s "Mail subject" "to-user@example.com" -- -r "from-user@example.com"​

5. Attaching files. The mail command does not provide an option for attaching files. There is a workaround for attaching files using the uuencode command. Pipe the output of uuencode command for attaching files. ​

uuencode attachment-file | mail -s "Mail subject" "to- user@example.com" < body.txt​

* Examples - Reading Emails: ​

1. Viewing all the received emails Simply type the mail and then press enter to view the received emails. ​

mail​

Another way of viewing the emails is using the -f option. This is shown below: ​

> mail -f /var/spool/mail/user​

Mail version 8.1 6/6/93. Type ? for help.​

"/var/spool/mail/user": 2 messages 2 new​

>N 1 root@hostname Tue May 17 00:00 21/1013 "Mail subject 1"​

N 2 root@hostname Wed May 18 00:00 21/1053 "Mail subject 2"​

&​

From the above output, you can see that, it displays the from-address, date and subject of the emails in the inbox. It also displays the ampersand (&) prompt at the end. ​

2. Reading an email.​

To read the Nth email, just enter the mail number at the ampersand prompt and press enter. This is shown below: ​

> mail -f /var/spool/mail/user​

&2​

This displays the second email details.​

3. Navigating through inbox emails.​

To go to the next email, enter the + symbol. To go back to the previous email, enter the - symbol at the ampersand prompt. ​

&-1​

This displays the first email details.

4. Replying email.​

Once you have read an email, you can give reply to the mail by typing "reply" and pressing enter. ​

&reply​

To: root@hostname​

root@hostname​

Subject: Re: Mail subject1​

5. Deleting emails. ​

You can delete a read email by typing the d and pressing enter. You can also specify the email numbers to d option for deleting them. ​

a. To delete read email​

&d​

b.To delete emails 1 and 2​

&d 1 2​

c. To delete range emails from 10 to 30​

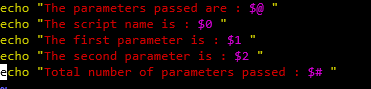
&d 10-30​

d. To delete all emails in the mbox (mail box)​

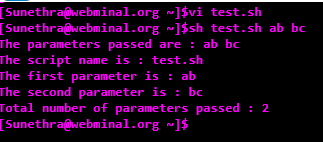
&d \*​

***Executing shell script by passing parameters***

* Example:



* Output:



Here, we have passed the parameters ab bc while running the script

***Executing shell script without passing parameters***

* Example:

A simple shell script without any parameters



* Output:



***ps Command***

* ps command is used for listing the process status.
* ps command has a lot of options. The option which we can use to list all the running jobs is -ef.
* -e is used to select all the processes
* -f can be used to do full format listing. It can be used with other options as well. It also causes the command arguments to be printed.
* Syntax : ps -ef

***diff and sdiff Command***

diff :

* diff stands for difference.
* This command displays the difference between files by comparing the files line by line.
* diff command tells us which lines in one file have to be changed to make the two files identical.
* diffuses certain special symbols and instructions that are required to make two files identical.

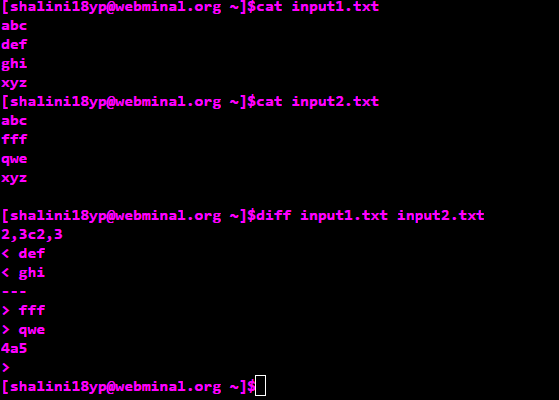
The special symbols are:

a : add

c : change

d : delete

* Syntax: diff [options] file1 file2
* Example:



The first line of the **diff** output will contain:

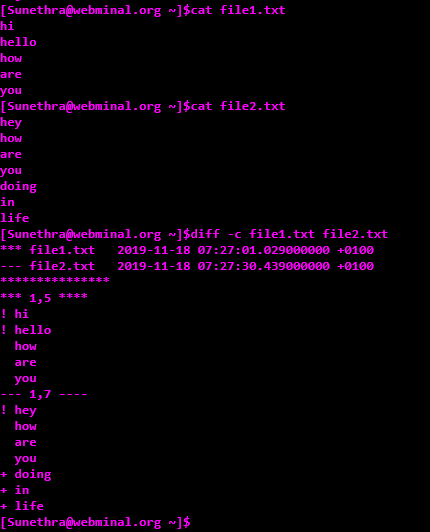
* Line numbers corresponding to the first file,
* A special symbol and
* Line numbers corresponding to the second file.

For example, the output shown above means-

* 1,2c1 indicates that we need to change (c) the first and second (1,2) lines of first file (<) i.e, hi and hello to hey, to match the second file(>).
* The three dashes (---) just seperate first and second file
* 5a5,7 indicates that we need to add doing, in, life to first file so as to match it with second file.
* Options:

1. -c (context) : To view differences in context mode, use the **-c** option.

*diff –c file1.txt file2.txt*



The first file is indicated by **\*\*\***, and the second file is indicated by **—**. The line with **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** is just a separator.

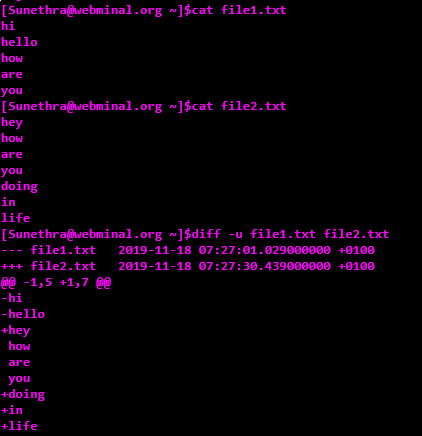
The first two lines of this output show us information about **file 1** and **file 2**.

(i) If the line needs to be unchanged, it is prefixed by two spaces.  
(ii) If the line needs to be changed, it is prefixed by an symbol and a space. The symbol means are as follows:

(a) + : It indicates a line in the second file that needs to be added to the first file to make them identical.  
(b) – : It indicates a line in the first file that needs to be deleted to make them identical.

2. -u (unified) : To view differences in unified mode, use the -u option. It is similar to context mode but it doesn’t display any redundant information or it shows the information in concise form.

*diff –u file1.txt file2.txt*



3. -i : By default this command is case sensitive. To make this command case in*-*sensitive use **-**i option with diff.

diff -i file1.txt file2.txt

4. –version : This option is used to display the version of diff which is currently running on your system.

sdiff:

* sdiff command is used to compare two files and then writes the results to standard output in a side-by-side format.
* It displays each line of the two files with a series of spaces between them if the lines are identical.
* It displays greater than sign if the line only exists in the file specified by the File2 parameter, and a | (vertical bar) for lines that are different.
* Syntax: sdiff [ -l | -s ] [ -o OutFile ] [ -w Number ] File1 File2
* Options:

1. -l : Displays only the left side when lines are identical.

2. -s : It does not display the lines that are identical.

3. -w Number : It sets the width of the output line. Default value of Number is 130 characters. Maximum value is 2048. Minimum width is 20.The sdiff command uses 2048 if a value greater than 2048 is specified.