UNIT-2

Additional File Management Command

- **Additional File Management Commands:**

  Creating Directories-mkdir, Removing Empty Directories-rmdir, Copying Files and Directories-cp, Removing Files and Nonempty Directories-rm, Renaming Files and Directories-mv, Comparing Two Files-cmp, What is Common-comm, Converting One File to Other-diff, Piping and Redirection andtee, File and Directory Permission and Privileges
  chmod, Locating Files-find.

- **EditingFiles:**

  Creating and Editing files using vi, picoand, emacseditors.

- **Basics of Shell Scripting Programming:**

  Creating Shell Scripts using various commands of Linux except Filters.; Interactive shell script using read and echo; Decision Statements: if then fi, if then elsefi, if then elif elsefi, caseesac; Test command; Logical Operators; Looping statements: for loop, while loop, until loop; Break, continue command; Arithmetic in Shell script using expr; Creating Shell Scripts to perform mathematical calculations.

Unit –2 Additional File Management Command

2.1 Additional File Management Command:

➢ Creating Directories - mkdir command:

mkdir command in Linux allows the user to create directories (also referred to as folders in some operating systems). This command can create multiple directories at once as well as set the permissions for the directories. *mkdir* stands for "make directory."

Syntax:

mkdir [options...] [directories ...]

Option / Syntax	Description
mkdir directory_name	Creates a directory in the current location
mkdir {dir1,dir2,dir3,dir4}	Creates multiple directories in the current location. Do not use spaces inside {}
mkdir –p directory/path/newdir	Creates a directory structure with the missing parent directories (if any)
mkdir –m777 directory_name	Creates a directory and sets full read, write, execute permissions for all users
mkdir –v directory_name(s)	Creates a directory in the current location and give feedback for successful operations. It is also called verification.

Examples:

Sr. No.	Command	Description / Output
1.	mkdir bca	Create bca directory.
2.	mkdir {sem1,sem2,sem3}	Create Multiple directories like sem1, sem2 and sem3.
3.	mkdir –p bca/bca6/linux	It creates bca6 and linux directory in bca directory.

4.	mkdir –m777 gujarat	It creates gujarat directory with all the permissions to all users.
5.	mkdir –v gujarat/ahm	It creates ahm directory in gujarat directory.

➢ **Remove Directories with rmdir Command:**

rmdir command is used only when deleting empty folders and directories in Linux. If a specified directory is not empty, the output displays an error.

The basic syntax used for **removing empty Linux folders/directories** is:

Syntax:

rmdir [dir_name]

Additionally, you can **delete multiple empty directories** at once by typing:

Syntax:

rmdir [dir_name1][dir_name2][dir_name3]

If the command finds content in one of the listed directories, it will skip it and move on to the next one.

Example:

Sr. No.	Command	Description / Output
1.	rmdir gujarat	Remove gujarat directory
2.	rmdir sem1 sem2 sem3	Removes sem1, sem2 and sem3 directory.

➢ **Remove Directory with rm Command:**

rm command removes a directory/folder along with all the files and sub-directories in it.

By adding the -r (-R) option to the rm command, you can remove a directory along with all its contents.

To **remove a directory** (and everything inside of it) use the –r option as in the command:

Syntax:

rm –r dir_name

This will prompt you for confirmation before deleting.

To **remove a directory without confirmation**:

rm –rf directory

Also, you can **delete more than one directory** or folder at a time:

rm –r dir_name1 dir_name2 dir_name3

Example:

Sr. No.	Command	Description / Output
1.	rmdir –r gujarat	Remove gujarat directory with all subdirectories and files.
2.	rmdir –r sem1 sem2 sem3	Removes sem1, sem2 and sem3 directory with subdirectories and files.

➢ **Copying Files and Directories:**

The cp command is the primary method for copying files and directories in Linux.

Syntax:

cp [additional_option] source_file target_file

Example:

cp my_file.txt my_file2.txt

This Linux command creates a copy of the *my_file.txt* file and renames the new file to ***my_file2.txt.***

By default, the cp command runs in the same directory you are working in. However, the same file cannot exist twice in the same directory.

⇨ **Copy File into another directory:**

To copy a file from the directory you're working in to a different location, use the command:

Syntax:

cp my_file.txt /new_directory

You don't need to rename the file unless there's already one with the same name in the target directory.

To specify a path for the source file:

cp /etc/my_file.txt /new_directory

This lets you copy without having to change directories. The cp command will create the **/new_directory** if it doesn't exist.

To rename and copy a file to a different path:

cp my_file.txt /new_directory/my_file2.txt

Example:

Sr. No.	Command	Description / Output
1.	cp myfile.txt myfile2.txt	Copy myfile.txt file into the same directory as a myfile2.txt
2.	cp myfile.txt /bca	Copy myfile.txt file into the bca directory as a myfile.txt
3.	cp myfile.txt /bca/myfile2.txt	Copy myfile.txt file into the bca directory as a myfile2.txt
4.	cp myfile.txt /bca/myfile2.txt	Copy myfile.txt file into the bca directory as a myfile2.txt

- **Renaming Files and Directories-mv:**

mv command is a command that is used primarily to move files and folder from one location to another. However, it can also be used to rename a file. It is also used to rename the directory.

Syntax:

mv (option) filename1 filename2

Example:

Sr. No.	Command	Description / Output
1.	mv ahmedabad.txt ahm.txt	Rename ahmedabad.txt file to ahm.txt
2.	mv –v ahmedabad.txt ahm.txt	Rename ahmedabad.txt file to ahm.txt as well as it display output (verbose) on your screen. Output: renamed ahmedabad.txt -> ahm.txt
3.	mv dir_india dir_hindustan	Rename dir_india directory to dir_hindustan.
4.	mv /home/baroda.txt /home/gujarat/bar.txt	Rename baroda.txt file to bar.txt and move this file into gujarat directory.

- **Comparing Two Files:**

cmp command in Linux/UNIX is used to compare the two files byte by byte and helps you to find out whether the two files are identical or not.

- When cmp is used for comparison between two files, it reports the location of the first mismatch to the screen if difference is found and if no difference is found *i.e* the files compared are identical.
- cmp displays no message and simply returns the prompt if the files compared are identical.

Syntax:

cmp (OPTION) FILE1 FILE2

Example:

Sr. No.	Command	Description / Output
	Suppose myfile.txt contains Hello How are you. ***Suppose myfile2.txt contains*** Hello I am fine.	
1.	cmp myfile.txt myfile2.txt	It compares both the file contents identically. If it is not same then it displays byte and line number. If it is same then no message will be displayed. Output: myfile.txt myfile2.txt differ: byte 7, line 1
2.	cmp -b myfile.txt myfile2.txt	It displays the differing bytes in the output when used with -b option Output: myfile.txt myfile2.txt differ: byte 7, line 1 is 110 H 111 I
3.	cmp -l file1.txt file2.txt	-l option displays the byte position and byte value for all differing bytes. 7 110 111 8 157 40 9 167 141 10 40 155 11 141 40 12 162 146

		13 145 151 14 40 156 15 171 145 16 157 56 17 165 12 In our example, it differs from byte 7 so it displays 7 then ascii value of (H)-110 from myfile.txt and ascii value of (I)-111 then continue for all other characters.
4.	cmp -i 7:17 myfile.txt myfile2.txt	-i option skip the 7th to 17th bytes from both the file so remaining characters are same (Hello) so it displays nothing (no message) means contents are matched.

➢ **Converting One File to Other – diff:**

diff stands for **difference**. This command is used to display the differences in the files by comparing the files line by line. This command displays which lines in one file is required to be changed to make the two files identical.

Special symbols are:

a : add

c : change

d : delete

Syntax:

diff (OPTION) FILE1 FILE2

The first line of the diff output contains:

- line numbers corresponding to the first file,
- a letter (a for *add*, c for *change*, or d for *delete*), and
- line numbers corresponding to the second file.

In our output above, "2,4c2,4" means: "Lines 2 through 4 in the first file need to be changed to match lines 2 through 4 in the second file." It then tells us what those lines are in each file:

- Lines preceded by a < are lines from the first file;
- lines preceded by > are lines from the second file.

The three dashes ("---") merely separate the lines of file 1 and file 2.

Example:

Suppose states.txt contains	*Suppose otherstates.txt contains*	*Suppose newstates.txt contains*	*Suppose latest.txt contains*
Gujarat Rajasthan	Gujarat Rajasthan	Gujarat Rajasthan Uttarakhand Jharkhand	Gujarat Rajasthan Kerala Uttarakhand Jharkhand

Sr. No.	Command	Description / Output
1.	diff states.txt otherstates.txt	It compares both the file contents identically. Contents of both the files are same so *no message* will be displayed.
2.	diff states.txt newstates.txt	It compares both the file contents identically. Contents of both the files are different so *it will show line number.* Output: 1,2c1,4 < Gujarat < Rajasthan --- > Gujarat > Rajasthan > Uttarakhand > Jharkhand
	In our output above, "1,2c1,4" means: "Lines 1 through 2 in the first file need to be changed to match lines 1 through 4 in the second file." It then tells us what those lines are in each file: • Lines preceded by < are lines from the first file; • Lines preceded by > are lines from the second file. The three dashes ("---") merely separate the lines of file 1 and file	
3.	diff newstates.txt latest.txt	Output: 2a3 > Kerala

	Here, 2*a*3 means requires to add Kerala in 3rd line of newstates.txt file to match the contents with latest.txt file.	
4.	diff latest.txt newstates.txt	3d2 < Kerala
	Here, 3*d*2 means requires to delete 3rd line (Kerala) from latest.txt file to match the contents with newstates.txt file.	

- **Piping – Redirection:**

Pipe is used to combine two or more commands. Redirection means to transfer of standard output to some other destination and for that pipe character '|' is used.

Syntax :

command_1 | command_2 | command_3 | .... | command_N

Example:

Suppose

latest.txt contains

Gujarat

Rajasthan

Kerala

Uttarakhand

Jharkhand

Sr. No.	Command	Description / Output
1.	ls – l | more	The more command takes the output of $ ls -l as its input. The net effect of this command is that the output of ls -l is displayed one screen at a time.
2.	ls -l > temp.txt	It stores the output of ls –l in temp file.
3.	cat latest.txt | head -3	Output: Gujarat Rajasthan Kerala
4.	cat latest.txt | tail -2	Output: Uttarakhand

		Jharkhand
5.	cat latest.txt | head -2 > temp.txt	Following output is stored in temp.txt file. Gujarat Rajasthan

➢ **Tee:**

The tee command is used to reads the standard input and writes it to one or more files, along with the standard output. The tee command is commonly used along with other commands with pipe (|).

Syntax :

tee (options) FILES…

Example:

Suppose

myfile.txt contains

Gujarat

Rajasthan

Gujarat

Uttarakhand

Gujarat

Sr. No.	Command	Description / Output
1.	echo "Hello How are you" | tee file1.txt	Output: Hello How are you It creates file1.txt and store Hello How are you.
2.	echo "Hello How are you" | tee file1.txt echo "_____" ls -l | grep file1 echo "_____" cat file1.txt	Output: Hello How are you _____ -rwxrwxrwx 1 root root 18 Dec 24 07:37 file1.txt _____ Hello How are you ********** It creates file1.txt and store Hello How are you.

3.	cat states.txt | grep "Gujarat" | tee file2.txt | wc –l	Output: 3 It creates file2.txt and stores Gujarat 3 times.

➢ **File and Directory Permission and Privileges – chmod:**

chmod command is used to change the access mode of a file. The name is an abbreviation of change mode. In Linux, each file is associated with an owner and a group and assigned with permission access rights for three different classes of users:

- The file owner
- The group members
- Others (everybody else)

There are three file permissions types that apply to each class. Each write, read, and execute permissions have number values which is another option to set the permissions.

- The read permission - Number value is 4
- The write permission - Number value is 2
- The execute permission - Number value is 1

chmod command allows you to specify which users are allowed to read the file, write to the file, or execute the file. The operator is used to specify how the modes of a file should be adjusted. The following operators are accepted:

Operator	Description
+	Adds the specified modes to the specified classes.
-	Removes the specified modes from the specified classes.
=	The modes specified are to be made the exact modes for the specified classes

Syntax:

chmod [reference][operator][mode] file...

Example:

Sr. No.	Command	Description / Output
1.	ls –l	It will display files and directories with permissions.
	-rwxrwxrwx 1 root root 21 Dec 28 12:03 gujarat.txt Here, - means it is file not directory and rwx*rwx*rwx means read, write and execution permission to user, *group* and others.	

2.	chmod 642 gujarat.txt ls –l	It will change the permission of gujarat.txt file as under User – Read & Write (4+2) Group – Read (4) Others – Write (2) -rw- r-- -w- 1 root root 21 Dec 28 12:04 gujarat.txt
3.	chmod u=rwx gujarat.txt	It gives user to read, write and execute permissions.
4.	chmod u+x gujarat.txt	It adds execute permissions to user.
5.	chmod u-r gujarat.txt	It removes the read permission from user.
6.	chmod u=rwx,g=r,o= filename	It gives read, write and execute permission to user, read permission to group and no permission to others.
7.	chmod 777 gujarat.txt	It gives read, write and execute permissions to user, group and others.

⇨ ***How to change the permission to directory recursively:***

To recursively operate on all files and directories under the given directory, use the –R option.

Example: chmod –R 755 /root/India

It will change the permission of all the files and subdirectories of the India directory recursively.

Locating Files using Find command:

The find command in Linux is a command line utility for walking a file hierarchy. It can be used to find files and directories and perform subsequent operations on them. It supports searching by file, folder, name, creation date, modification date, owner and permissions. By using the '-exec' other UNIX commands can be executed on files or folders found.

Syntax:

find [where to start searching from] [expression determines what to find] [-options] [what to find]

Examples:

1. Search a file with specific name.

find ./india -name gujarat.txt

It will search for gujarat.txt in india directory.

2. Search a file with pattern.

find ./india -name *.txt

It will give all files which have '.txt' at the end.

3. How to find and delete a file with confirmation.

find ./india -name gujarat.txt -exec rm -i {} \;

When this command is entered, a prompt will come for confirmation, if you want to delete gujarat.txt or not. if you enter 'Y/y' it will delete the file.

4. Search for empty files and directories.

find ./india -empty

This command find all empty folders and files in the entered directory or sub-directories.

5. Search for file with entered permissions.

find ./india -perm 664

This command find all the files in the india directory or sub-directory with the given permissions.

6. Search text within multiple files.

find ./ -type f -name "*.txt" -exec grep 'hello' {} \;

This command print lines which have 'hello' in them and '-type f' specifies the input type is a file.

2.2 Editing Files:

Editing Files using vi, pico and emacs editors:

There are two types of text editors in Linux:

1. Text Editors: are used to run in terminal windows.
2. Graphical Editors: are used to run in graphical windows with menus and mouse pointers.

❖ **Text Mode Editors:**

The three text mode editors of choice in LINUX are vi, emacs, and pico.

➢ **vi Editor:**

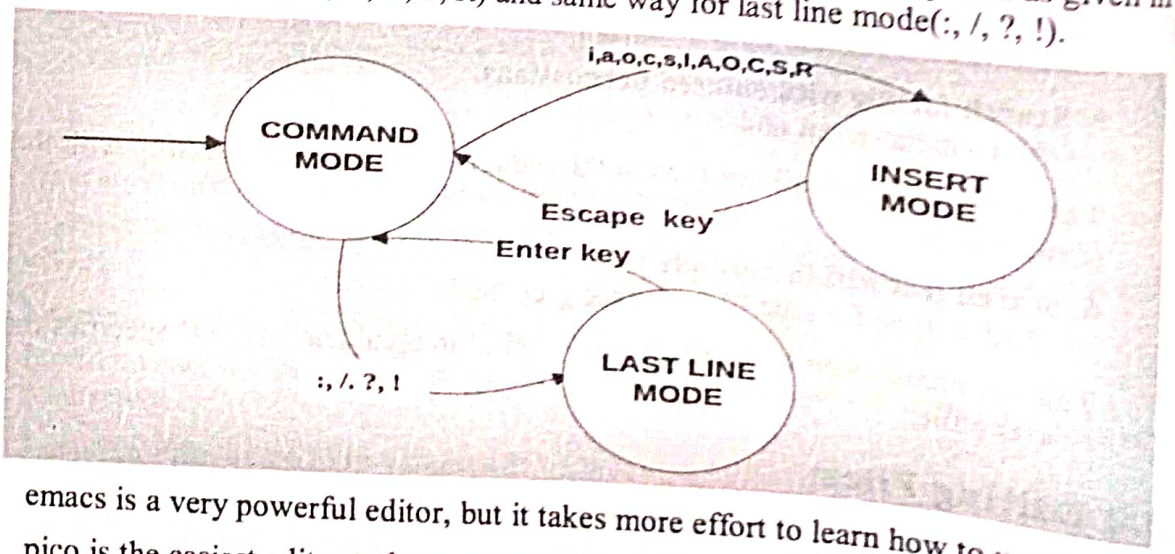
vi is the original editor for Unix. It is very fast, easy to use, and available on virtu every UNIX / LINUX system. The vi commands are the same as several other comm LINUX tools. The improved version of vi called vim (vi IMproved) was written a released in 1991. vim is upward-compatible from vi, so that everything in vi works vim. vim offers more features than vi, but it is also much harder to learn than vi.

vi has three modes, or states that is Command Mode, Insert Mode, Last-line Mode. A any given time it is in exactly one of these modes. With each key that you press, it eith stays in its current mode or changes to a different one.

vi is a CASE-SENSITIVE: everything you type must be in the correct case. vi makes copy of the file in a temporary location to use as its work buffer. Sometimes this cop is readable by other people who know how to find it.

⇨ **How to edit using vi Editor:**

vi editor works as per the image below. The arrows from one state to another are labelle by the keys that cause its state to change. As per the image, when user wish to write insert (Command mode to Insert mode) then required to press any options as given in th image(i, a, o, c, s, I, A, O, C, S, R) and same way for last line mode(:, /, ?, !).



emacs is a very powerful editor, but it takes more effort to learn how to use it.

pico is the easiest editor to learn and the least powerful.

2.3 Basics of Shell Scripting Programming:

User can use any of the editor like pico, emacs and vi to write the script. There are two types of comments.

1. Single line comment: '#' symbol is used to add single line comment in unix/linux.
2. Multi line comment: ':' and " ' " symbols are used to add multiline comment.

Example:

```
# First Shell Script
: '
   echo is used for display the message and goto the next line
   echo -n is used for display the message and -n means not go to the next line.
   and
   read is use to accept the value
'
echo "Hello How are you" # Display Message
echo -n "Enter the Rollno:"  # Display Message
read Rollno              # Accept Rollno from user
echo -n "Enter your Surname:"
read Surname
echo -n "Enter your Name:"
read Name
echo "Full Name is: $Name $Surname"
```

Output:

```
Hello How are you
Enter the Rollno:1
Enter your Surname:Shah
Enter your Name:Harshi
Full Name is: Harshi Shah
```

- **Expr command:**

The expr command in Linux/Unix evaluates a given expression and displays its corresponding output. It is used for the basic operations like addition, subtraction, multiplication, division and modulus on integers. It is also used for the evaluating regular expressions, string operations like substring, length of strings etc.

Syntax:

```
$expr expression
```

Example of expr using Console:

Expression	Output	Description
$ expr 12 + 8	20	Addition of two numbers
$ expr 12 - 8	12	Subtraction of two numbers

$ expr 12 * 8	96	Multiplication of two numbers
$ expr 12 / 8	1	Division of two numbers
$ expr 12 % 8	4	Modulus of two numbers

⇨ **Example of expr using Shell Script:**

```
# Write a shell script to accept two numbers and display sum.
echo -n "Enter the NO1:"
read no1
echo -n "Enter the NO2:"
read no2
 sum=`expr $no1 + $no2`
 echo "Sum is: " $sum
```

Output:

Enter the NO1:10
Enter the NO2:20
Sum is:30

⇨ **Example of expr to compare values using shell script:**

```
# Write a shell script to accept two numbers and compare it.
echo -n "Enter the NO1:"
read no1
echo -n "Enter the NO2:"
read no2
res1=`expr $no1 = $no2`
echo "Result using = operator is:" $res1
res2=`expr $no1 \> $no2`
echo "Result using > operator:" $res2
res3=`expr $no1 \!= $no2`
echo "Result using != operator:" $res3
```

Output:

Enter the NO1:10
Enter the NO2:20
Result is: 0 (Display 0 if both the numbers are different, otherwise it displays 1).
Result using > operator: 0 (Check 10>20, NO=0 and YES=1)

Result using != operator: 1 (Check 10!=20, NO=0 and YES=1)

⇨ **Example of expr for String Operations:**

```
# Write a shell script to accept your name and display length of name.
echo -n "Enter your Name:"
read name
len=`expr length $name`
echo "length is:" $len
sub=`expr substr $name 1 3`
echo $sub
```

Output:

```
Enter your Name:Keyur
length is: 5
Substring is: Key
```

⇨ **Conditional (Decision) Statements:**

There are total 5 conditional statements which can be used in unix / linux programming.

1. if statement
2. if-else statement
3. if..elif..else..fi statement (Else If ladder)
4. if..then..else..if..then..fi..fi..(Nested if)
5. switch statement

1. if statement:

This block will process if specified condition is true.

Syntax:

```
if [ expression ]
then
        statement
fi
```

Example:

```
# Write a Shell Script to check two numbers are equal or not.
a=10
b=20
```

```
#Check whether they are equal
if [ $a == $b ]
then
   echo "a is equal to b"
fi
#Check whether they are not equal
if [ $a != $b ]
then
   echo "a is not equal to b"
fi
```

Output: a is not equal to b

2. **if-else statement:**

If specified condition is not true in if part then else part will be execute

Syntax:

```
if [ expression ]
then
        statement1
else
        statement2
fi
```

Example:

```
# Write a Shell Script to check two numbers are equal or not.
a=20
b=20
if [ $a == $b ]
then
     echo "a is equal to b"  #If they are equal then print this.
else
   echo "a is not equal to b"    #else print this
fi
```

Output: a is equal to b

3. if..elif..else..fi statement (Else If ladder):

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:

: ' Shell Script to accept rollno, name and marks of english, maths and science from user and display sum, percentage and grade as follows:

```
# percentage >=80, "A", >=70, "B", >=60, "C" otherwise "F"
echo -n "Enter the Rollno:"  # Display Message
read rollno                  # Accept Rollno from user
echo -n "Enter your Name:"
read name
echo -n "Enter the marks of English :"
read eng
echo -n "Enter the marks of Maths :"
read maths
echo -n "Enter the marks of Science :"
read sci
sum=`expr $eng + $maths + $sci`
echo "Sum is:" $sum
```

```
per=`expr $sum / 3`
echo "Per is:"$per
if [ $per -gt 80 ]
then
   echo "Grade A"

elif [ $per -gt 70 ]
then
   echo "Grade B"
else
   echo "Grade C"

fi
```

Output:

```
Enter the Rollno:1
Enter your Name:HARSHI
Enter the marks of English :65
Enter the marks of Maths :97
Enter the marks of Science :88
Sum is: 250
Per is:83
Grade A
```

4. if..then..else..if..then..fi..fi..(Nested if):

Nested if-else block can be used when, one condition is satisfies then it again checks another condition. In the syntax, if expression1 is false then it processes else part, and again expression2 will be check.

Syntax:

```
if [ expression1 ]
then
  statement1
  statement2
  else
```

```
  if [ expression2 ]
  then
    statement3
   fi
fi
```

Example:

```
count=99
if [ $count -eq 100 ] # eq means equal to
then
 echo "Count is 100"
else
 if [ $count -gt 100 ] # gt means greater than
 then
  echo "Count is greater than 100"
 else
 echo "Count is less than 100"
 fi
fi
```

Output: Count is less than 100

5. **switch statement:**

Case statement works as a switch statement if specified value match with the pattern then it will execute a block of that particular pattern. When a match is found all of the associated statements until the double semicolon (;;) is executed.

A case will be terminated when the last command is executed. If there is no match, the exit status of the case is zero.

Syntax:

```
case in
  Pattern 1) Statement 1;;
  Pattern n) Statement n;;
esac
```

Example:

Write a shell script to accept two numbers and operator from user and display appropriate output.

```
echo -n "Enter the Number 1 :"
read no1
echo -n "Enter the Number 2 :"
read no2
echo -n "Enter the Operator (+,-,*,/) :"
read op
case $op in
  "+") ans=`expr $no1 + $no2`
     echo "Addition of two number is :" $ans ;;
  "-") ans=`expr $no1 - $no2`
     echo "Subtraction of two number is :" $ans ;;
  "*") ans=`expr $no1 \* $no2`
     echo "Multiplication of two number is :" $ans ;;
  "/") ans=`expr $no1 / $no2`
     echo "Division of two number is :" $ans ;;
esac
```

Output:

```
Enter the Number 1 :20
Enter the Number 2 :10
Enter the Operator (+,-,*,/) :/
Division of two number is : 2
```

➢ **Test Command:**

The test command is used to check file types and compare values. Test is used in conditional execution. It is used for:

- File attributes comparisons
- Perform string comparisons.
- Basic arithmetic comparisons.

Syntax:

```
test [EXPRESSION]
        or
[EXPRESSION]
```

Example:

Expression	Output	Description
test 100 -gt 99 && echo "Yes, that's true." || echo "No, that's false."	Yes, that's true.	Conditions is true so it gives Yes, that's true.
test 100 -lt 99 && echo "Yes." || echo "No."	No.	Condition is false so it gives No.
[100 -gt 101] ; echo $?	1	If condition is true than 0 otherwise it gives 1. Here, condition is false.
[100 -gt 99] ; echo $?	0	Condition is true so it gives 0 as output.
["Gujarat" = "Gujarat"]; echo $?	0	Condition is true so it gives 0 as output.
["Gujarat" = "gujarat"]; echo $?	1	Condition is False so it gives 1 as output.
test -f /etc/resolv.conf && echo "File /etc/resolv.conf found." || echo "File /etc/resolv.conf not found."	File /etc/resolv.conf found.	Check the file is available or not and give the output accordingly.

⇨ **Looping Statements, Break and Continue:** There are total 3 looping statements which can be used in unix / linux programming.

→ while statement

→ for statement

→ until statement

While statement: In while loop / statement, it works as per the evalution of condition / command and executes the statement accordingly.

Syntax

```
while command
do
   Statement to be executed
Done
```

Example:

```
i=1
while [ $i -le 10 ]
```

```
do
  echo Si
  i=`expr Si + 1`
done
```

Output: 12345678910

For statement: For loop is operate as per the start and end values. It will execute statement while variable value not reach it to the end value.

Syntax

```
for (( expr1; expr2; expr3 ))
do
 command1
 command2
 ..
Done
```

Example:

```
for (( i=1; i<=10; i++ ))
do
  echo -n Si
done
```

Output: 12345678910

⇨ **Until statement:**

The until loop is executed as many as times the condition/command evaluates to false. The loop terminates when the condition/command becomes true.

Syntax:

```
until command
do
  Statement to be executed until command is true
done
```

Example:

```
a=1
until [ ! Sa -lt 10 ]
do
  echo -n Sa
```

```
    a=`expr $a + 1`
  done
```

Output: 12345678910

- **Break and Continue:**

break: The break statement is used to terminate the loop and can be used within a while, for, until, and select loops. It is used to alter the flow of loop statements.

Syntax:

```
break
```

Example:

```
for (( i=1; i<=10; i++ ))
do
    if [ $i -eq 5 ]
    then
        break
    fi
    echo -n $i
done
```

Output: 1234

- **continue:** continue is a command which is used to skip the current iteration in for, while and until loop.

Syntax:

```
continue
```

Example:

```
for (( i=1; i<=10; i++ ))
do
    if [ $i -eq 5 ]
    then
        continue
    fi
    echo -n $i
done
```

Output: 1234678910