UNIT—9

SHELL PROGRAMMING

* Vi editor (Visual Editor)

The vi-editor is the most popular editor in Linux. The current version is really "vim", but to invoke it simply type "vi". The text format that vi uses is the most plain elementary text format that exists, the ASCII format. But files created by vi can be run through other programs that convert the files to other formats, if desired. In linux & other unix clone O.S the O.S and the editor do not care whether or not file names have suffixes or extensions. It is available in all Linux systems. It can handle files that contain text. It does not provide to include graphics, font formatting. It cannot do bold face, running header, footer. It doesnot in any error message if any thing goes wrong.

There are three (3) modes in which the editor works. Under each modes the same key press perform diff. task.

1) Command Mode :-

It is the default of vi editor. In this mode all the key press by user are interrupted to be an editor command. If we press 'h' the cursor is moved one position to the next. The key press on this mode are not display on the screen but perform it's defined a task.

2) Insertion Mode :-

This mode makes it possible to insert characters we capture inside of the document. To switch to insertion mode, just press the key Insert on our keyboard or, by default, the key i. When we press (i) in the command mode then vi editor is open in insert mode. It allow a user to edit, insert new record to the existing file. The insertion mode is also known as "Text Mode".

8) Replacement Mode :-

This mode allows you to replace existing text by the text you capture. Just hit r again to go to replacement mode & hit the key Esc to return to regular mode.

* Shell Variables :-

A variable in a shell script is a means of referencing a numeric or character value. And unlike formal programming languages, a shell script does not require you to declare a type for your variables. Thus, you could assign a number to the variable.

The Shell's environment is all the information that the shell will use as it runs.

Positional Parameters

In a program we need to convey information. A very convenient way of doing this these by specifying arguments at the command line. But how does the shell script know about what has been passed to it. For this, the shell uses something called Positional Parameter. There can be variables defined by shell. They are 9 (Nine) in number named $1 to $9.

It is possible to write a shell script that takes a number of parameters at the time you invoke it from the command line or from another shell script. These options are supplied to the shell program by Linux as positional parameters, which have special names provided by the system.

Command line Argument :-

The argument given to a program at command line is known as Command line Argument. These values are accepted by positional parameter use in program.

Shell Programming :-

1) Write a Shell Script to add two number.

```
echo "enter two number"
read a b
c= ' expr $a + $b '
echo $c.
```

- **#** :- Each comment line must be start with # (Symbol)
- **echo** :- It is a command used to display a message or value present in a variable the message must be enclosed in Double quote(" ").
- **read** :- It is a command use to Input a value to the variable at run-time of program

- **expr** :- It is used to perform an arithmetic operation in Linux.
- **$** :- The value of a variable can be access using $ symbol before variable.

Q.2) Write a Shell script to swap any two number.

Sol:

```
# swap two number
echo " enter two number "
read a b
temp = $a
a = $b
b = $temp
echo $a $b.
```

* **Relational operator in Shell program.**

-gt → greater than.

-ge → greater than or equal to

-l → less than

-le → less than equal.

-eq → equality (==)

-!eq → not equal to.

Logical Operation :-

-a → and

-o → or

! → not.

* Conditional Statment :-

1) If [condition] then

Statment 1

Statment 2.

fi

2) If [condition] then

Statment 1

" 2

else

Statment 1

" 2

fi

3) If [condition] then

If [condition] then

Statement

else

Statement

fi

else

fi.

Q.1) Write a Shell program to enter a number whether it is even or odd.

Sol:

```
echo "enter a number"
read a
If [ `expr $a % 2` -eq 0 ]
then
echo "even number"
else
echo "odd number"
fi.
```

Write a Shell Script to check whether an input no. is divisible by 5 and 3.

```
echo "enter a number"
read a
If [ `expr $a % 3` -eq 0 -a `expr % a %5`
                                -eq 0 ]
then
echo " no. is divisible by 3 and 5"
else
 echo " NOT "
fi.
```

* If - elif - fi :-

It is a counterpart of multiple if statement in Shell program.

```
If [ ------ ]
=  =  =
=  =  =

elif [      ]
=======

elif [      ]
-------
=======
```

elif [ ]

fi.

Q.1) Enter the Marks of 5 subjects for a student & display remarks according to following criteria.

i) 60>= -> First.

ii) 45>= -> 2nd

iii) 35>= -> 3rd

iv) Fail.

Sol:

```
echo "enter marks obtained in 5 subjects"
read a b c d e
tot = expr $a + $b + $c + $d + $e
per= 'expr $tot / 5 '
if [ per -ge 60 ] then
  echo "First"
elif [ per -ge 45 ] then
   echo "Second"
elif [ per -ge 35 ] then
   echo "Third"
else
```

```
        echo "fail"
    fi
```

* Switch statement used in Programming :-
  case ..... esac.

Syntax:-

```
case variable in  -> keyword.
    1) echo "Hello"
       ;;
    2) echo "How"
       ;;
    *) echo "wrong choice"
       ;;
esac
```

Write a shell program to use case esac statement.

```
echo "enter your choice"
read a
case a in
1) ls
;;
```

2) clear

;;

*) echo "wrong choice"

esac.

* **Case** :-)

It allow us to match an expression for more than one alternatives. The case 1st matches expression with pattern 1 if the match success then it executes command 1 if the match fail then pattern 2 is match & so on. Each command list is terminated with a pair of (;;) semicolons. The last option *, this represent the default clause of the case instruction. It gets executed when all other cases fail.

* **Looping** :-)

1) while condition

do

===

done.

To print 1 ---- 10.

```
a=1
while [ $a -le 10 ]
do
echo $a
a='expr $a + 1'
done
```

8). Write a Shell program enter a no. & check whether it is prime number or not.

```
# Prime no. or not
echo "enter a number"
read a
v=1
count=0
while [ $v -le $a ]
do
if ['expr $a % $v' -eq 0] then
count='expr $count + 1
fi
```

```
v = 'expr $v + 1'
done
if [ $count -eq 2 ] then
echo " Number is prime "
else
echo " Not prime "
fi.
```

Q). Write a shell program to reverse a Number.

Sol:

```
echo "enter a no."
read num
rev = 0
while [ num > 0 ]
do
dig = ' expr $num % 10 '
num = ' expr $num / 10
rev = ' expr $rev * 10 + dig.
done
echo $rev.
```

write a shell program to display the Fibonacci series of n elements:

```
echo "enter the no: of elements"
read n
x=0
y=1
temp=0
echo $x $y
while [ `expr $n - 2` -gt 0 ]
do
temp=`expr $x + $y`
echo $temp
x=$y
y=$temp
n=`expr $n - 1`
done
```

8) Write a Shell program to display all armstrong no. between 1 to 1000.

Sol.

```
x=1
while [ $x -le 1000 ]
do
p= $x
arm= 0
while [ $p -gt 0 ]
do
dig = 'expr $p %% 10'
arm = 'expr $arm + $dig *
p = 'expr $p /10'
done
if [ $arm -eq $x ]
 echo $x
fi
x= 'expr $x +1'
done
```

* Looping :-

i) while :-

The statement within the while loop executed till the exit status of the control command remains true. When the exit status of the control command turns out to be false. The control passes to the 1st command that follow the body of the while loop.

ii) until :-

The statement within the until loop keep on getting executed till the exit status of the control command remain false (1). When the exit status becomes true (0) the control passes through the 1st command that follows the body of the until to loop.

iii) for loop :-

The for loop allow us to specify a list of values which the control variable in the loop can take. The loop is executed for each value mentioned in the list.

8) Write a Shell program that will accept any digit no. & determine the length of the number.

(Ex-> 1 2 3 4 —> Input

4 —> Output.

Sol:

```
num=0
while [ $1 -gt 0 ]
do
dig = ' expr $1 % 10'
num = 'expr $num + 1'
$1 = 'expr $1 / 10'
done
echo $ num.
```