Shell Programming

Title: Shell Programming

AIM:

- **Implement addition, subtraction, multiplication and division using required control structures**

- **Convert a string from upper case to lower case or vice versa**

OBJECTIVE:

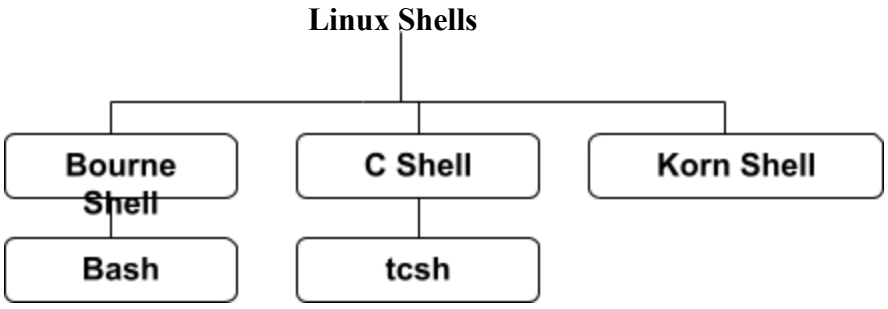
Understanding of Linux shell commands & shell programming.

THEORY:

- **Linux Shells**

Linux comes with various command interpreters called as shells in the Unix terminology. Actually shell sits in between the kernel of an operation system and the user. So whatever user wants to do through kernel it is available in terms of shell commands. Once you provide valid command for the required operation it hands over the request to the operating system kernel and finally job will be done by the system.

There are various shells available to use in the Linux environment but following shells are the standard shells.



The shells used in the Linux operating system has dual capability, in one had it is used as a tool which accepts commands interpret it and hands over it to the operating system kernel. Due to this capability it is called as command – line interpreter, another use of shell is it can be used as a programming language. Shell programming is interpretive by nature and mostly it is used to assist in system administration tasks.

- **Steps to interpret a shell script**

Assume a written shell script is stored in a script file named as **Example-1.** To execute this script we have two approaches: ***Make it as an executable*** and ***execute it as an argument to bash*** command.

Make it as an executable :

- By the use of **chmod** command one can set execute permission on it. It is shown below for our first script.
  chmod +x Example-1
- This script can be executed like any other shell command after associating the execute permission.
- The following command shows the execution step for our **Example-1** file.
  ./Example-1

Running it as an argument of bash :

This approach of script execution can be done to specify its name as the argument of the bashcommand.

- For example, the script named as **Example-1** can be executed by the use of following command.

bash Example-1

- **Basic Shell Commands**

- **read**

The command read reads one line from the standard input (or from a file) and assign the word to the variable.

Example

```
read var_year
echo "The year is: $var_year"
```

```
echo -n "Enter your name and press [ENTER]: "
read var_name
echo "Your name is: $var_name"
```

- **echo**

echo command in the bash shell writes its arguments to standard output**.**
The syntax for echo is

```
echo [option(s)] [string(s)]
```

The items in square brackets are optional. A *string* is any finite sequence of *characters* (i.e., letters, numerals, symbols and punctuation marks).

When used without any options or strings, echo returns a blank line on the display screen followed by the command prompt on the subsequent line.

For example, a variable named *x* can be created and its value set to 5 with the following command:

```
x=5
```

The value of x can subsequently be recalled by the following:

```
echo The number is $x.
```

- **expr**

It is an old Unix program that can evaluate math is **expr**. Since it is a command, command substitution is needed.

Example

```
z=`expr $z + 1`
echo $z
```

- **Control Structures in shell**

- The bash shell provides many options to control the flow of script executions, typically known as control structures. The Table below list out these structures.

if	If statement is used to execute commands based on certain conditions are met. It can be customized by the use of else to indicate what should happen if the condition does not satisfies.
case	This is used to handle multiple options in the script. It is same like switch statements used in C programming language.
for	This is a loop statement. The *for* statement is used to run a command for a given number of items.
while	Use while statement to execute the included statements as long as the specified condition is met.
until	This statement works exactly opposite of while statement. This can be used to execute a command until a certain condition is met.

If...Then...Else :

- The if…then…else construct is used for flow control facility in shell scripting. This is usually used in conjunction with test command.
- This can be used for various activities like to find out the existence of a file, whether a variable currently has a value etc. The basic construction of if…then…else… statement is shown in below program code :

General description of if…then construct

```
if condition expression
then
    command(s)
fi
```

- This means one can use it to check one specific condition, and in case of true the command or group of commands associated will be executed.

- The if…then…fi construct can be extended by including else statement to handle all cases where the condition was not met. The Fig. below shows the actual syntax of the construct along with test command.

```
if test condition expression
then
        command(s)
fi
```

Or

```
if [ condition expression ]
then
        command(s)
fi
```

Fig. If…then…fi statement

- **Syntax of if…else statement :**

  The else can be included in the basic if statement in the fashion shown Fig

```
if  condition expression
then
        command(s)
else
        command(s)
fi
```

Or

```
if  [  condition expression  ]
then
        command(s)
else
        command(s)
fi
```

Fig.If…then…else statement

- **Syntax of multiple else blocks using elif :**

  The if statement can have elif construct in case of multiple else statements included along with the if statement.

```
if  condition expression
then
        command(s)
elif  condition expression
then
        command(s)
else
       command(s)
fi
```

Or

```
if  [  condition expression  ]
then
        command(s)
elif  [  condition expression  ]
then
        command(s)
else
        command(s)
fi
```

Fig.: elif statement

Case Statement :

- The case statement is a multiple branch decision mechanism. The syntax of case statement is described in the following figure as well as its working is shown in the below Fig.

```
case word in
pattern1)
        command(s)
        ;;
pattern2)
        command(s)
        ;;
.....
patternN)
        command(s)
        ;;
*)
esac
```

Fig. Syntax of case statement

While Statement :

- The while statement is described with the nature says : As long as the condition is true, the commands between the do and the doneare executed.
- The syntax of while statement is described in the Fig. below.

```
while   condition
do
        command(s)
done
```

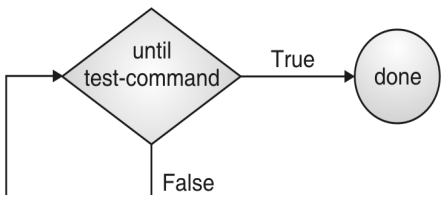
Fig. Syntax of while statement

Until Statement :

- The syntax of until statement is described in the following figure as well as its working is shown in the Fig
- This loop continues to execute the command(s) between the do and done until the condition is true.

```
until    condition
do
        command(s)
done
```

Fig.: Syntax of until statement



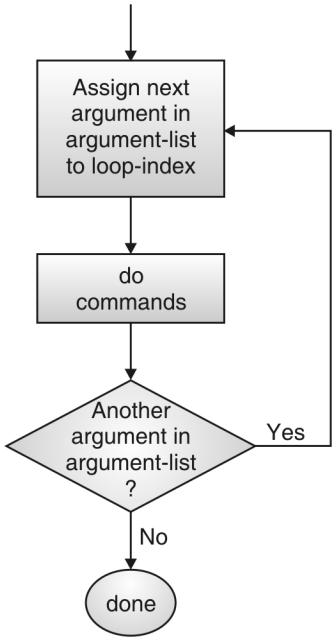
for…in Statement :

- The for…in control structure has the following syntax :

```
for index in argument-list
do
        command(s)
done
```

Fig: Syntax of for…in statement

- The **for…in** structure gets a value from the argument-list and assigns it into loop-index in every iterations and executes the command(s) between do and done statements. The do and done marks the beginning and end of the for loop statement.



Using for Control Statement :

- The for control structure has the following syntax :

```
for index
do
        command(s)
done
```

Fig. Syntax of for control structure

- In this for control structure the index variable takes value of the each of the command line arguments, one at a time.
- This structure is same as for…in structure; the only difference is here the loop index gets value from command line arguments instead of the separate argument list provided along with for.

Using for((…)) Loop Statement :

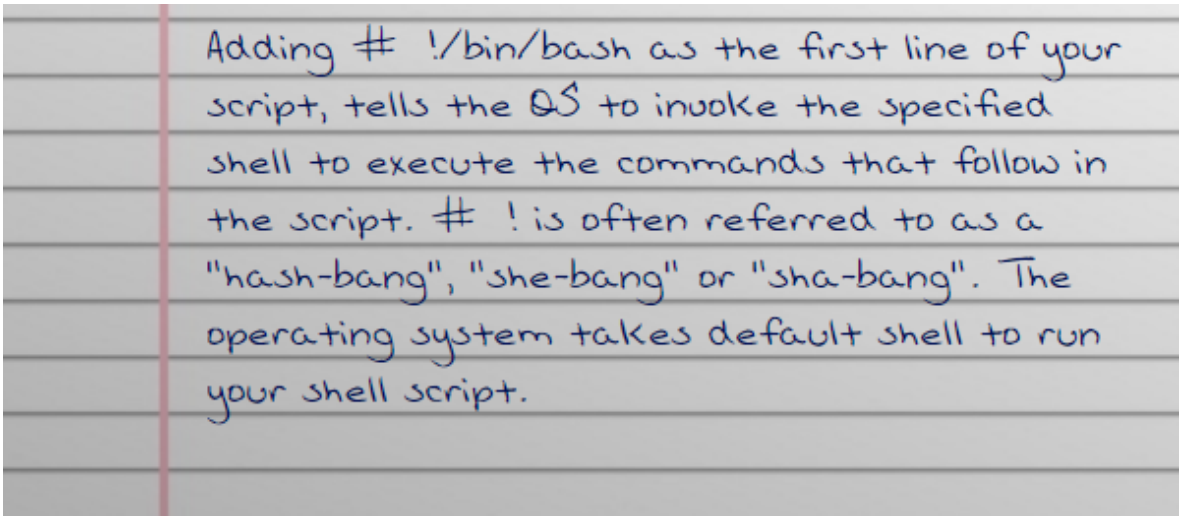
- The for loop is used to execute a series of commands, either for a limited number of times or for an unlimited number of times. This works like high level programming languages for loops.

```
for ((init ; condition ; increment))
do
        command(s)
done
```

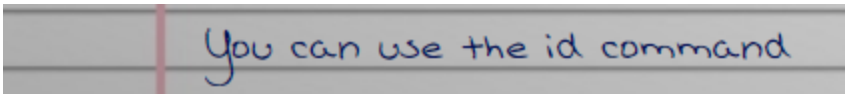
Fig: Syntax of for((…)) statement

- **INPUT: please specify input test case**
- **OUTPUT: please specify the expected output for the input test case**
- **FAQs:**

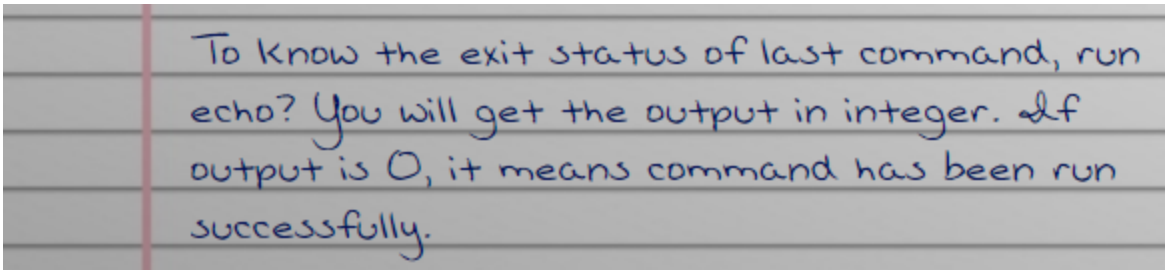
1. **Why does #!/bin/sh have to be the first line of my scripts?**



2. **How can I access the name of the current shell in my initialization scripts?**



3. **How can I determine whether a command executed successfully?**



- **CONCLUSION**

The Linux shell provides several commands and programming language constructs to implement various operations.

Code:

```
#!/bin/bash

echo "Enter Two numbers : "
read a
```

```
read b

# Input type of operation
echo "Enter Choice :"
echo "1. Addition"
echo "2. Subtraction"
echo "3. Multiplication"
echo "4. Division"
read ch

# Switch Case to perform
# calculator operations
case $ch in
  1)res=`echo $a + $b | bc`
  ;;
  2)res=`echo $a - $b | bc`
  ;;
  3)res=`echo $a \* $b | bc`
  ;;
  4)res=`echo "scale=2; $a / $b" | bc`
  ;;
esac
echo "Result : $res"
```

```
Enter Two numbers :
Enter Choice :
1. Addition
2. Subtraction
3. Multiplication
4. Division
Result : 7

[Execution complete with exit code 0]
```