Experiment 5: Shell Scripting Part I

1. Aim:

Understand the fundamental concepts of shell scripting

2. Requirements: Linux OS

3. Related Theory:

In Linux, a command is a program that you can run. To run a command in Linux, you type its name and press Enter. E.g. > date [ENTER]. But how does the computer know that you wanted to run the command date? The computer uses a special program called the **shell** to figure this out. The shell provides you with an interface to the Linux system. It gathers input from you and executes programs based on that input. When a program finish executing, it displays that program's output. For this reason, the shell is often referred to as the Linux system's command interpreter. For users familiar with Windows, the Linux shell is similar to the DOS shell. The real power of the Linux shell lies in the fact that it is much more than a command interpreter. It is also a powerful programming language, complete with conditional statements, loops, and functions.

Shell Scripts

A shell script ends with extension .sh. To run the script by typing its name, we need to do two things: a) Make it executable, b) Make sure that the right shell is used when the script is run.

Typically, to make this script executable, do the following:
> chmod +x filename.sh
chmod (Change Mode) - Using chmod we can change the access permissions to file system objects. +x makes the file executable.

To ensure that the correct shell is used to run the script, you must add the following "magic" line to the beginning of the script: #!/bin/bash. The magic line causes a new shell (in this case, /bin/bash) to be called to execute the script. Without the magic line, the current shell is always used to evaluate the script, regardless of which shell the script was written for.
Note that the #!/bin/bash must be the first line of a shell script in order for bash to be used to run the script. If this appears on any other line, it is treated as a comment and ignored by all shells. A **comment** is a statement that is embedded in a shell script but should not be executed by the shell. In shell scripts, comments start with the # character. Everything between the # and end of the line are considered part of the comment and are ignored by the shell.

An example script (tmp.sh) is

```
#!/bin/bash
# print out the date
date
```

To execute this script, on the command prompt type
> bash s1.sh

Arithmetic Operators:

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator	`expr $a + $b` will give 30
- (Subtraction)	Subtracts right hand operand from left hand operand	`expr $a - $b` will give -10
* (Multiplication)	Multiplies values on either side of the operator	`expr $a \* $b` will give 200
/ (Division)	Divides left hand operand by right hand operand	`expr $b / $a` will give 2
% (Modulus)	Divides left hand operand by right hand operand and returns remainder	`expr $b % $a` will give 0
= (Assignment)	Assigns right operand in left operand	a = $b would assign value of b into a
== (Equality)	Compares two numbers, if both are same then returns true.	[$a == $b] would return false.
!= (Not Equality)	Compares two numbers, if both are different then returns true.	[$a != $b] would return true.

Relational Operators:

Operator	Description	Example
-eq	Checks if the value of two operands are equal or not; if yes, then the condition becomes true.	[$a -eq $b] is not true.
-ne	Checks if the value of two operands are equal or not; if values are not equal, then the condition becomes true.	[$a -ne $b] is true.
-gt	Checks if the value of left operand is greater than the value of right operand; if yes, then the condition becomes true.	[$a -gt $b] is not true.
-lt	Checks if the value of left operand is less than the value of right operand; if yes, then the condition becomes true.	[$a -lt $b] is true.
-ge	Checks if the value of left operand is greater than or equal to the value of right operand; if yes, then the condition becomes true.	[$a -ge $b] is not true.
-le	Checks if the value of left operand is less than or equal to the value of right operand; if yes, then the condition becomes true.	[$a -le $b] is true.

Boolean Operators:

Operator	Description	Example
!	This is logical negation. This inverts a true condition into false and vice versa.	[! false] is true.
-o	This is logical **OR**. If one of the operands is true, then the condition becomes true.	[$a -lt 20 -o $b -gt 100] is true.
-a	This is logical **AND**. If both the operands are true, then the condition becomes true otherwise false.	[$a -lt 20 -a $b -gt 100] is false.

String Operators:

Operator	Description	Example
=	Checks if the value of two operands are equal or not; if yes, then the condition becomes true.	[$a = $b] is not true.
!=	Checks if the value of two operands are equal or not; if values are not equal then the condition becomes true.	[$a != $b] is true.
-z	Checks if the given string operand size is zero; if it is zero length, then it returns true.	[-z $a] is not true.
-n	Checks if the given string operand size is non-zero; if it is nonzero length, then it returns true.	[-n $a] is not false.
str	Checks if **str** is not the empty string; if it is empty, then it returns false.	[$a] is not false.

While:

The while loop enables you to execute a set of commands repeatedly until some condition occurs. It is usually used when you need to manipulate the value of a variable repeatedly.

Syntax

```
while command
do
      Statement(s) to be executed if command is true
done
```

Break and Continue:

The break and continue statements can be used to control the loop execution. The break statement terminates the current loop and passes program control to the command that follows the terminated loop. It is usually used to terminate the loop when a certain condition is met.

For:

The for loop operate on lists of items. It repeats a set of commands for every item in a list. Here var is the name of a variable and word1 to wordN are sequences of characters separated by spaces (words). Each time the for loop executes, the value of the variable var is set to the next word in the list of words, word1 to wordN.

Syntax

```
for var in word1 word2 ...wordn
do
   Statement to be executed
done
```

Until:

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
```

Bash Calculator:

The Linux **bc** program functions as a convenient desktop calculator or as a mathematical scripting language. It's as easy as calling the bc command through a terminal.

Usage:

```
echo '6.5 / 2.7' | bc
```

Note the pipe operator, the expression will be given to the bash calculator which executes the expression.

```
add=`echo "scale=3; $x + $y" | bc`
echo $add
```

Here, scale represents how many decimal places to display in the output.

Case statements:

The basic syntax of the case...esac statement is to give an expression to evaluate and to execute several different statements based on the value of the expression. The interpreter checks each case against the value of the expression until a match is found. If nothing matches, a default condition will be used.

```
case word in
   pattern1)
      Statement(s) to be executed if pattern1 matches;;
   pattern2)
      Statement(s) to be executed if pattern2 matches;;
   pattern3)
      Statement(s) to be executed if pattern3 matches;;
   *)
     Default condition to be executed;;
esac
```

Select:

Select command is a very useful bash command for bash menu creation. Different types of menu generation task, creating menu-based director list, creating a menu from file content etc. can be done using bash select command.

Syntax

```
select v in data_list
do
      statement1
      statement2
done
```

4. Laboratory Exercise:

Write shell scripts for the following:

i) print hello world, ii) take 2 arguments to print hello world, iii) read a number, iv) read name, v) add two numbers using command line, vi) adding two numbers in the shell script, vii) multiplication of two numbers, viii) division of two numbers, ix) check whether a number is positive or negative, x) find the maximum of two numbers, xi) example usage of while with break and continue, xii) example usage of for, xiii) example usage of until, xiv) example usage of case, xv) example usage of select.

5. Post-Experiment Exercise:

1. Write a shell script to define any four variables (of different data types) and display them using echo. For example:
   myval=45
   echo "Value of myval is $myval"

2. Accept the age of user from input. Write a shell script to determine if the user is eligible for voting.

3. Write a shell script that accepts two numbers from users. Display the addition, subtraction, multiplication, division (quotient), remainder results using echo; without and with expr.

4. Write a shell script to determine the maximum of three numbers.

5. Accept two strings from user. Write a shell script to check if the two strings are same or not.

6. Write a shell script to validate the course marks using nested if and specify the class. Accept marks for three courses – Physics, Chemistry, Mathematics. Compute the total marks and average marks. If any course has less than 35 marks, display "Failed". If the average marks is >= 75, display "Distinction". If the average marks is >=60 but <=75, display "First Class. If the average marks is >=50 but <=60, display "Second Class. If the average marks is >=35 but <=50, display "Third Class.

7. Display "Linux" i times, where i goes from 1 to 8. Demonstrate using while loop and for loop.

8. Write an infinite loop using for loop (use CTRL + C to exit).

9. Write a shell script to print a number in reverse order using while loop. It should support the following requirements. The script should accept the input from the command line. If you don't input any data, then display an error message to execute the script correctly.
   Hint:
   a. Suppose the input number is n.
   b. Set reverse to 0 and digit to 0 (i.e., rev=0, digit=0).
   c. The expression (n % 10) will give the single leftmost digit i.e., digit.
   d. To reverse the number, use this expression rev * 10 + digit.
   e. Decrease the input number (n) using n / 10.
   f. If n is greater than 0, then go to step no. 3. Else, execute the step no. g.
   g. Print the result.

10. Write a shell script to reverse a user-defined string. Hint: use pipe operator and rev.

11. Write a shell script to reverse the word order in a list of strings. For example, if the input is Hello World, output should be World Hello.

12. Write a shell script using case to either create a new file or delete an existing file.
    bash script.sh --create newfile.txt should create this new file and
    bash script.sh --delete newfile.txt should delete this existing file.
    Display "Not a valid argument" if neither --create nor --delete is specified. Hint: Use case $1 to determine the argument option.

13. Write a shell script for the following: accept a city name from user; using case statements, determine the country of this city. Provide multiple city names in each case statement. Example of one case could be "Mumbai" | "Delhi" | "Pune") echo "The country is India" ;;. If options do not match, write one case statement as *) echo "To be added soon!".

14. Write a shell script to convert user-given temperature in Celsius to Fahrenheit using bash calculator.

15. Write a shell script to take two numbers from user and choice of arithmetic operations, i.e. add, subtract, multiply, divide, exponentiation, and return the corresponding result. Use case statements and bash calculator.

Submission Instructions:

For each question, students need to submit the question and answer with screenshot of the code and output of the code.

Conclusion:

 # Summarize your experience about the skills acquired from this experiment.