Introduction to shell scripting

horizontal line

**INTRODUCTION TO SHELL SCRIPT**

A shell script is a computer program which is run by command line interpreter i.e shell . Shell script includes multiple operations like printing text, file manipulation . It is a real programming language which is complete with variables , control structure and so on. A Simple shell script is a list of commands executed sequentially .

It involves writing a series of commands in a plain text file (usually with a .sh extension) that the shell interpreter, typically Bash (Bourne Again SHell), can execute sequentially. These scripts can range from simple, one-liners to complex, multi-step processes.

Empty shell script is a list of commands executed sequentially.

Following is the script of Shell Programming:

As you will create **test.sh script** all the script of shell program will have **.sh** extension and before starting the script we need to use shebang(#!)

For example-

#!/bin/sh

This tells the system that command to be executed by born shell.

Now the shell is ready to be executed, to execute the shell script we will use the following command –

1- Write a shell script to print hello:-

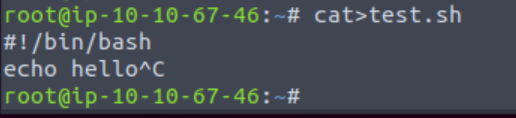
* **Create a file by using cat command:**

cat>test.sh

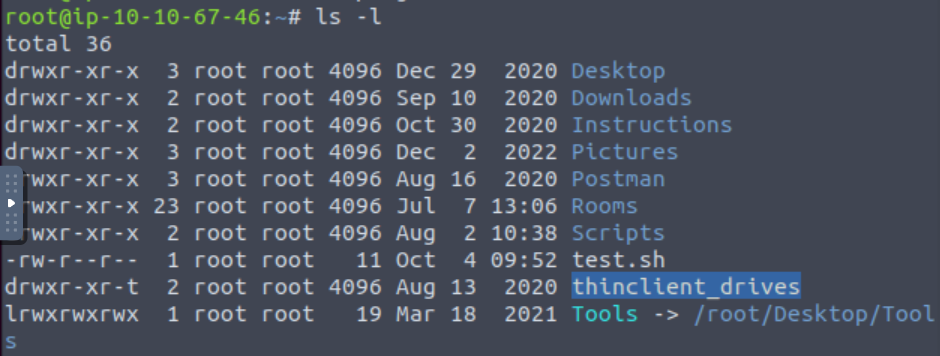
* **Define the Shebang:** Start your script by specifying the shebang line, which tells the system which shell interpreter to use. For Bash,use the following line:

#!/bin/bash

* **Write your script:** Below the shebang line, you can start writing your shell script. You can include various shell commands, variables, control structures, and comments to perform specific tasks. Here's an example of a simple script that prints "Hello, world!" to the terminal:

echo hello

See the content of your file

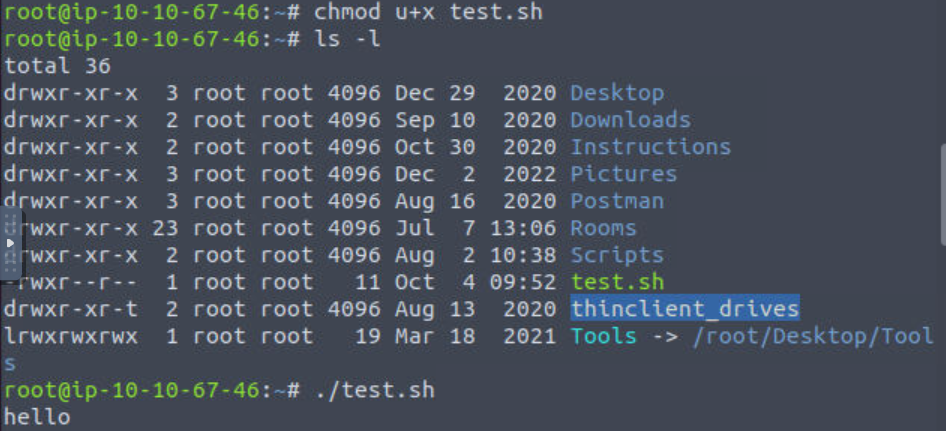
ls -l

* **Make the Script Executable:** Before you can run the script, you need to make it executable. In your terminal, navigate to the directory where your script is located and use the chmod command:

chmod u+x test.sh

* **Execute the Script:** You can now run your shell script by typing its name in the terminal:

**./test.sh**



**Shell Script Variables**: When a shell is running there are three main types of variables are present and they are,

1. Local Variable = It is one that is present within the current instance of a shell.

**Scope**: Local variables are limited in scope to a specific block of code, typically within a function or a script's code block. They are not accessible outside of the block where they are defined.

**Declaration**: You can declare a local variable within a function or script block without any special keyword. It's automatically considered local.

1. Environment Variable = It is available to any child process of shell, usually shell script defines only those variables that are needed by programs.

**Scope**: Environment variables are available to all processes running on the system, including shell sessions and child processes of the shell. They have a global scope and persist between sessions.

**Declaration**: Environment variables are typically set outside of the shell script, often in configuration files or by the system itself.

**Example**: Common environment variables include $HOME, $PATH, $USER, and $PWD. You can access their values in your shell script like any other variable.

1. Shell Variable = It is a special variable that is set by shell, some of those variables are environment variables while others are local variables.

**Scope**: Shell variables are variables explicitly defined within the shell or shell script, and they have a limited scope, typically within the shell session or script where they are declared.

**Declaration**: You can declare a shell variable by assigning a value to it directly.

Shell variables can be made available to child processes (subshells) by exporting them using the export command. Once exported, they become environment variables for the child processes.

# Unix / Linux - Shell Basic Operators

There are various operators supported by each shell and we will discuss them below:-

Bourne shell didn't originally have any mechanism to perform simple arithmetic operations but it uses external programs, either **awk** or **expr**.

**1.Arithmetic Operators**

The following arithmetic operators are supported by Bourne Shell.

| **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. |

## 2.Relational Operators

Bourne Shell supports the following relational operators that are specific to numeric values. These operators do not work for string values unless their value is numeric.

| **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

The following Boolean operators are supported by the Bourne Shell.

| **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

The following string operators are supported by Bourne Shell.

| **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 |

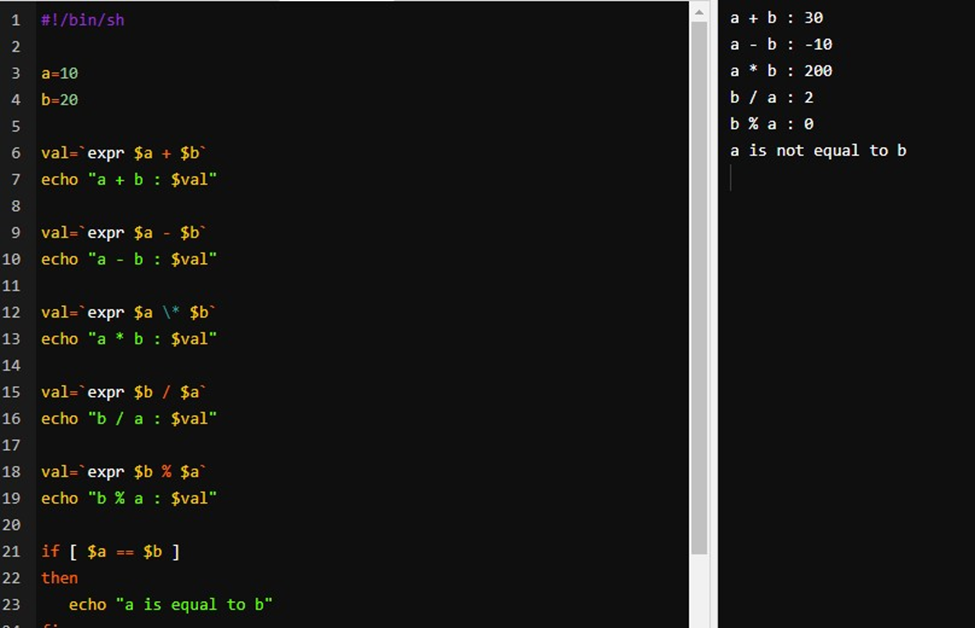
## 

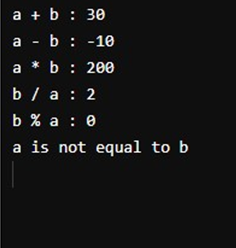
## File Test Operators

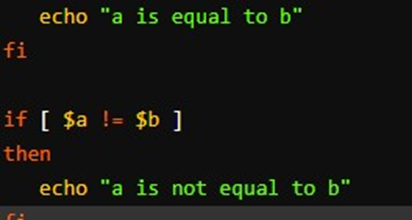
We have a few operators that can be used to test various properties associated with a Unix file.

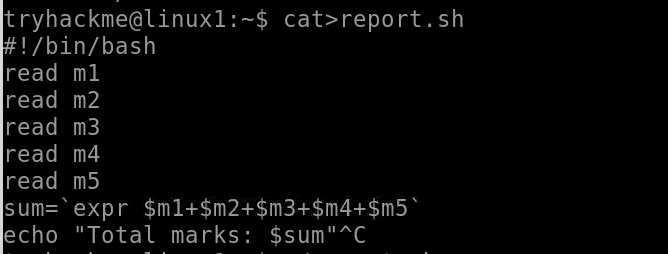
Assume a variable **file** holds an existing file name "test" the size of which is 100 bytes and has **read**, **write** and **execute** permission on –

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| **-b file** | Checks if file is a block special file; if yes, then the condition becomes true. | [ -b $file ] is false. |
| **-c file** | Checks if file is a character special file; if yes, then the condition becomes true. | [ -c $file ] is false. |
| **-d file** | Checks if file is a directory; if yes, then the condition becomes true. | [ -d $file ] is not true. |
| **-f file** | Checks if file is an ordinary file as opposed to a directory or special file; if yes, then the condition becomes true. | [ -f $file ] is true. |
| **-g file** | Checks if file has its set group ID (SGID) bit set; if yes, then the condition becomes true. | [ -g $file ] is false. |
| **-k file** | Checks if file has its sticky bit set; if yes, then the condition becomes true. | [ -k $file ] is false. |
| **-p file** | Checks if file is a named pipe; if yes, then the condition becomes true. | [ -p $file ] is false. |
| **-t file** | Checks if file descriptor is open and associated with a terminal; if yes, then the condition becomes true. | [ -t $file ] is false. |
| **-u file** | Checks if file has its Set User ID (SUID) bit set; if yes, then the condition becomes true. | [ -u $file ] is false. |
| **-r file** | Checks if file is readable; if yes, then the condition becomes true. | [ -r $file ] is true. |
| **-w file** | Checks if file is writable; if yes, then the condition becomes true. | [ -w $file ] is true. |
| **-x file** | Checks if file is executable; if yes, then the condition becomes true. | [ -x $file ] is true. |
| **-s file** | Checks if file has size greater than 0; if yes, then condition becomes true. | [ -s $file ] is true. |
| **-e file** | Checks if file exists; is true even if file is a directory but exists. | [ -e $file ] is true. |

**Q) Perform shell scripting for arithmetic operations**



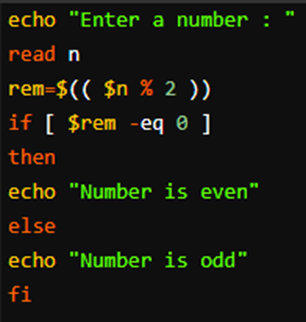
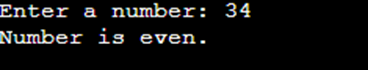


**Q) Perform shell scripting for printing a marksheet**

**Q) Program to copy data of one file into another**

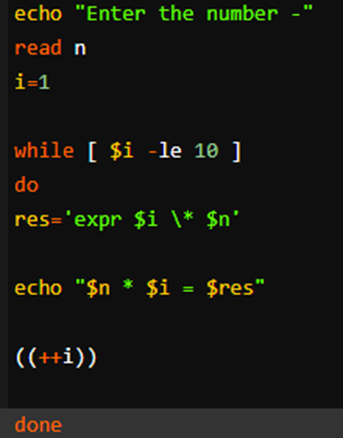
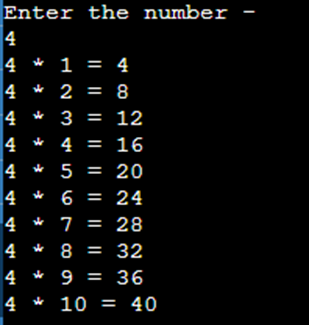
****

**Q) Program to find whether a number is even or odd**

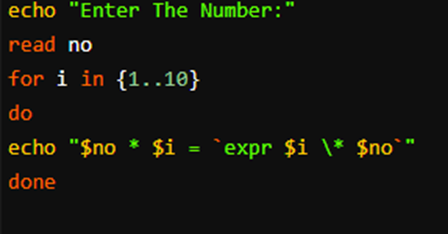
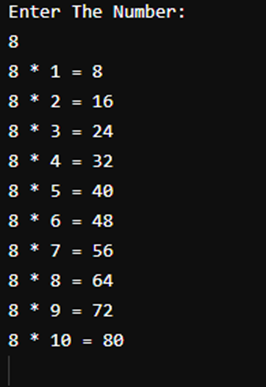
****

**Q) Program to print table**

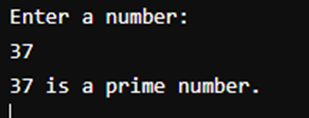
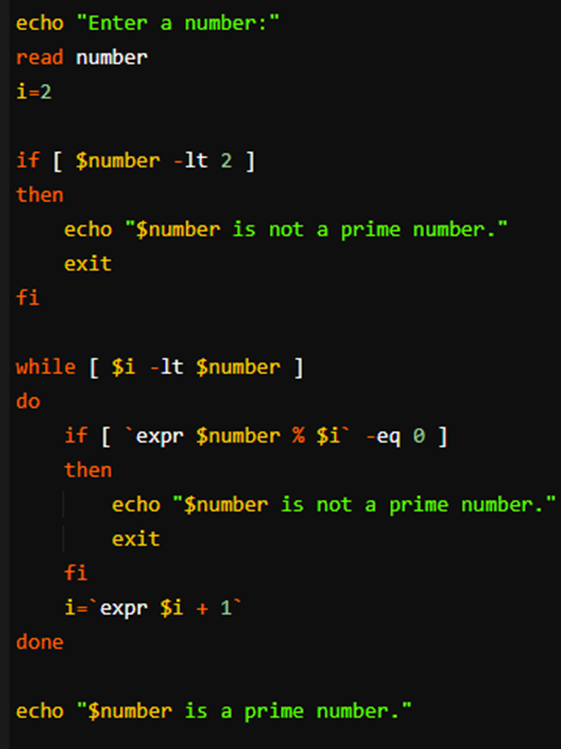
**a)Using while loop-**



**b)Using for loop**



**Q)Program to find whether a number is prime or not**



**Q) Program to check whether a number is <,> or = 100**

