**Create and Execute First BASH Program:**

1. Open any editor to create a bash file. Here, **nano** editor is used to create the file and filename is set as ‘**First.sh’**

**Excepted result “hello world”**

**Answer**

A screenshot of a computer

Description automatically generated with medium confidence

### Use of echo command:

1. When you use ‘**echo**’ command without any option then a newline is added by default. **‘-n’** option is used to print any text without new line and **‘-e’** option is used to remove backslash characters from the output.  
   Write text and use echo with -n and -e

Text

Description automatically generated

### Use of comment:

**‘#’** symbol is used to add single line comment in bash script. Create a new file named ‘**comment\_example.sh’**

Write program that sum two numbers and print the rustle with comment on top of the script document.

Graphical user interface

Description automatically generated with low confidence

### Use of Multi-line comment:

Logo, company name

Description automatically generated with medium confidence

### Using While Loop:

Create a bash file with the name, **‘while\_example.sh’,** to know the use of **while** loop. In the example, **while** loop will iterate for **5** times. The value of **count** variable will increment by **1** in each step. When the value of **count** variable will 5 then the **while** loop will terminate.

Graphical user interface, text, application

Description automatically generated

### Using For Loop:

The basic **for** loop declaration is shown in the following example. Create a file named ‘**for\_example.sh**’ and add the following script using **for** loop. Here, **for** loop will iterate for **10** times and print all values of the variable, **counter** in single line.

Graphical user interface, text, application

Description automatically generated

### Get User Input:

**read**’ command is used to take input from user in bash. Create a file named ‘**user\_input.sh**’ and add the following script for taking input from the user. Here, one string value will be taken from the user and display the value by combining other string value.

Graphical user interface, text, application

Description automatically generated

### Using if statement:

You can use if condition with single or multiple conditions. Starting and ending block of this statement is define by **‘if’** and **‘fi’**. Create a file named ‘**simple\_if.sh**’ with the following script to know the use **if** statement in bash. Here, **10** is assigned to the variable, **n**. if the value of **$n** is less than 10 then the output will be “**It is a one digit number**”, otherwise the output will be “**It is a two digit number**”. For comparison, **‘-lt’** is used here. For comparison, you can also use **‘-eq’** for **equality**, **‘-ne’** for **not equality** and **‘-gt’** for **greater than** in bash script.

Graphical user interface, text, application

Description automatically generated

### Using if statement with AND logic:

Different types of logical conditions can be used in if statement with two or more conditions. How you can define multiple conditions in if statement using **AND** logic is shown in the following example. **‘&&’**is used to apply **AND** logic of **if** statement. Create a file named **‘if\_with\_AND.sh’** to check the following code. Here, the value of **username** and **password**variables will be taken from the user and compared with ‘**admin**’ and ‘**secret**’. If both values match then the output will be “**valid user**”, otherwise the output will be “**invalid user**”.

Text

Description automatically generated

### Using if statement with OR logic:

‘**||**’ is used to define **OR** logic in **if** condition. Create a file named **‘if\_with\_OR.sh’**with the following code to check the use of **OR** logic of **if** statement. Here, the value of **n** will be taken from the user. If the value is equal to **15** or **45**then the output will be “**You won the game**”, otherwise the output will be “**You lost the game**”.

Text

Description automatically generated

### Using else if statement:

The use of **else if** condition is little different in bash than other programming language. ‘**elif**’ is used to define **else if** condition in bash. Create a file named, ‘**elseif\_example.sh**’ and add the following script to check how **else if** is defined in bash script.

Text

Description automatically generated

### Get Arguments from Command Line:

Bash script can read input from command line argument like other programming language. For example, **$1** and **$2** variable are used to read first and second command line arguments. Create a file named “**command\_line.sh**” and add the following script. Two argument values read by the following script and prints the total number of arguments and the argument values as output.

Graphical user interface, text, application

Description automatically generated

### Combine String variables:

You can easily combine string variables in bash. Create a file named “**string\_combine.sh**” and add the following script to check how you can combine string variables in bash by placing variables together or using **‘+’** operator.

Graphical user interface, text, application

Description automatically generated

### Add Two Numbers:

You can do the arithmetical operations in bash in different ways. How you can add two integer numbers in bash using double brackets is shown in the following script. Create a file named ‘**add\_numbers.sh**’ with the following code. Two integer values will be taken from the user and printed the result of addition.

Graphical user interface, text

Description automatically generated

### Create Function:

How you can create a simple function and call the function is shown in the following script. Create a file named ‘**function\_example.sh**’ and add the following code. You can call any function by name only without using any bracket in bash script.

Graphical user interface, text, application

Description automatically generated

### Create function with Parameters:

Bash can’t declare function parameter or arguments at the time of function declaration. But you can use parameters in function by using other variable. If two values are passed at the time of function calling then $1 and $2 variable are used for reading the values. Create a file named ‘**function|\_parameter.sh**’ and add the following code. Here, the function, ‘**Rectangle\_Area’** will calculate the area of a rectangle based on the parameter values.

Graphical user interface, text, application

Description automatically generated

### Make Directory:

Bash uses ‘**mkdir**’ command to create a new directory. Create a file named ‘**make\_directory.sh**’ and add the following code to take a new directory name from the user. If the directory name is not exist in the current location then it will create the directory, otherwise the program will display error.

Graphical user interface, text, application

Description automatically generated

### Make directory by checking existence:

If you want to check the existence of directory in the current location before executing the ‘**mkdir**’ command then you can use the following code. **‘-d**’ option is used to test a particular directory is exist or not. Create a file named, ‘**directory\_exist.sh’** and add the following code to create a directory by checking existence.

Graphical user interface, text, application

Description automatically generated

### Read a File:

You can read any file line by line in bash by using loop. Create a file named, ‘**read\_file.sh**’ and add the following code to read an existing file named, ‘**book.txt**’.

Graphical user interface, text, application, chat or text message

Description automatically generated

### Append to File:

New data can be added into any existing file by using **‘>>’** operator in bash. Create a file named **‘append\_file.sh**’ and add the following code to add new content at the end of the file. Here, ‘**Learning Laravel 5**’ will be added at the of ‘**book.txt’** file after executing the script.

Text

Description automatically generated