Experiment – 9

# INTRODUCTION TO SHELL SCRIPT

Shell scripts are a series of commands written in order of execution. These scripts can contain functions, loops, commands, and variables. Scripts are useful for simplifying a complex series of commands and repetitive tasks.

This is followed by the program that tells the operating system which interpreter to use to parse the rest of the file. In the example below, the program is /bin/bash which is a Unix shell.

# [Getting Started](https://www.digitalocean.com/community/tutorials/execute-command-shell-script#getting-started)

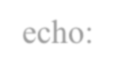
* A shell script needs to be saved with the extension .sh.
* The file needs to begin with the ***shebang line* (#!)** to let the Linux system know which interpreter to use for the shell script.
* For environments that support bash, use: **#!/bin/bash**

# Concept:

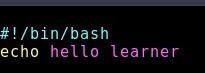
* **Variables and data types in Bash**: Variables let you store data. You can use variables to read, access, and manipulate data throughout your script. Eg- Assign the value directly country = India.

### . Shell Script to print “Hello Learner”.

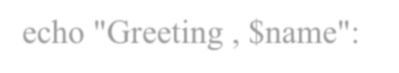
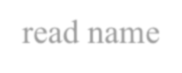
* + Use (#!/bin/bash) : Specifies Bash shell for script execution.



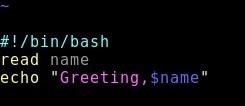
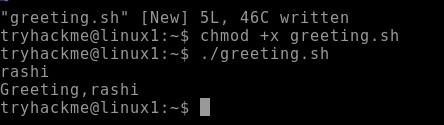
* + echo: echo command displays text or variables in the terminal.



1. **. Shell Script to take runtime argument from user and print “Greeting $your name”** - (#!/bin/bash): Specifies Bash shell for script execution.



* + read name: Reads input from the user and stores it in the variable name.
  + echo "Greeting , $name": Displays a greeting message that includes the value of the name variable, effectively saying "Greeting, [user's input]".

# Concept: Arithmetic Operators

## It uses external programs, either awk or expr.

#!/bin/sh val=`expr 2 + 2` echo "Total value : $val"

* There must be spaces between operators and expressions. For example, 2+2 is not correct; it should be written as 2 + 2.
* The complete expression should be enclosed between ‘ ‘, called the backtick.

+ (Addition) - `expr $a + $b` will give 30

* + (Subtraction) - `expr $a - $b` will give -10

\*(Multiplication) - `expr $a \\* $b` will give 200

/ (Division) - `expr $b / $a` will give 2

% (Modulus) - % (Modulus) **Arithmetic Operations using expr**:

* n= (expr 8 + 4): Performs addition (8 + 4) and stores the result in n.
* n1= (expr 5 - 7): Performs subtraction (5 - 7) and stores the result in n1.
* n2= (expr 9 / 6): Performs division (9 / 6) and stores the result in n2.
* n3= (expr 4 \\* 12): Performs multiplication (4 \\* 12) and stores the result in n3 • n4= (expr 5 % 2): Performs modulus operation (5 % 2) and stores the result in n4.

### Shell Script to perform arithmetic operations.

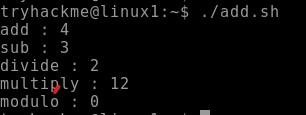
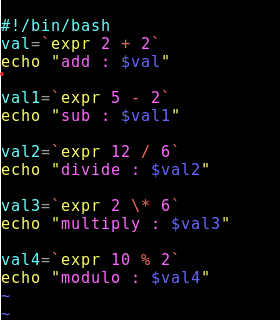
* + Specifies Bash shell for script execution.



(#!/bin/bash):

### Displaying Results using echo:

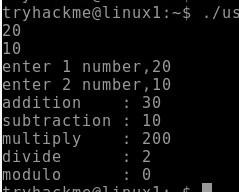
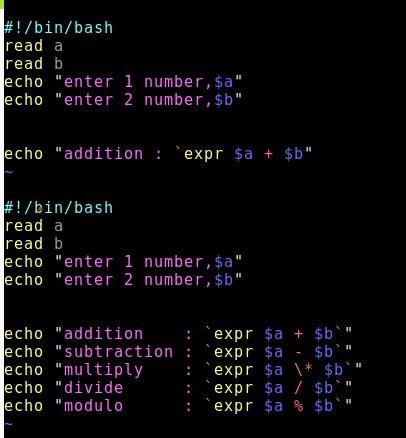
* + - Prints a header: "Arithmetic Operations:".
    - Displays the results of each operation (addition, subtraction, division, multiplication, modulus)



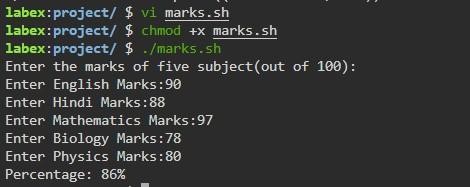
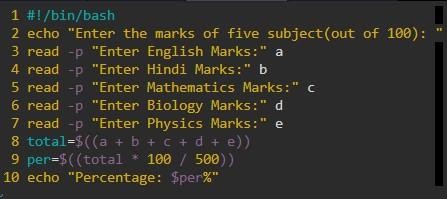
### Shell Script to perform arithmetic operations by taking values in run time. -

**(#!/bin/bash)**: Specifies Bash shell for script execution.

* + read: This command takes user input and stores it in a variable.
  + **echo:** echo command displays text or variables in the terminal..

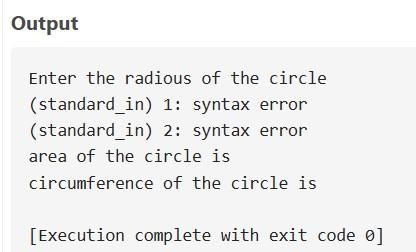
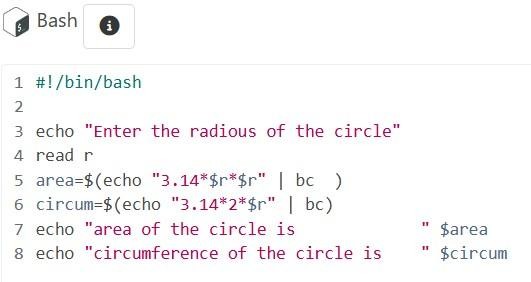


1. **Shell Script to take 5 subject marks and print percentage of it.** - (#!/bin/bash): Specifies the Bash shell for script execution.
   * **Calculating Total Marks**: Uses arithmetic expansion $(( ... )) to add up marks from the five subjects.
   * **Calculating Percentage**: Computes percentage by multiplying total marks by 100 and dividing by 500 (total possible marks).
   * **Displaying Results**: Output the percentage using echo.



# Shell Script to find the area and circumference of a circle:-

* **Echo:-** The echo command is used to output text to the terminal. Here, it simply tells the user what to input next.
* **Read:**- read is a command that captures user input.
* **echo "3.14\*$r\*$r":** The echo command constructs the mathematical expression for the area, using 3.14 as an approximation for π (π ≈ 3.14). The expression 3.14 \* r \* r calculates π \* r^2.
* **echo "3.14\*2\*$r":** This constructs the expression 2 \* 3.14 \* r, which represents the formula for the circumference.
* **echo outputs the string** "area of the circle is " followed by the value of the variable area.
* **echo outputs the string** "circumference of the circle is " followed by the value of the variable circum.



# Concept : If..else Condition

We use if-else in shell scripts when we wish to evaluate a condition, then decide to execute one set between two or more sets of statements using the result. This essentially allows us to choose a **response to the result which our conditional expression evaluates to.**

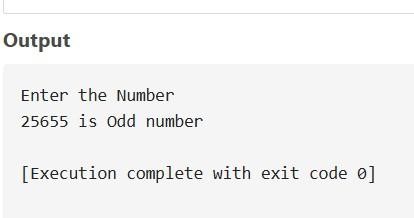
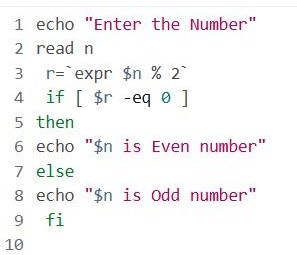
**if [condition] then statement1 else statement2**

fi

### Shell Script to take input from user and check whether number is even or odd.

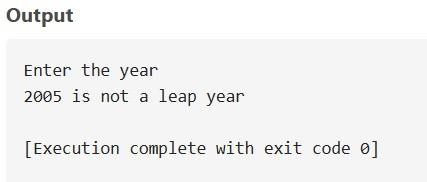
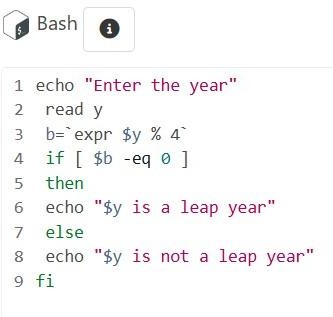
* **echo:** The echo command is used to display text on the terminal.
* **read:** The read command waits for the user to input something from the terminal and stores the input in the variable specified.
* **expr**: The expr command is used for evaluating expressions in Bash. In this case, it is used to calculate the **modulus** (remainder) of the division of n by 2.
* **r=**: The result of this modulus operation is stored in the variable r.
* **if**: The if statement starts a conditional block. It checks if the condition inside the brackets [ ] is

### true.



1. **Shell Script to check whether the year is a leap year or not.:-**

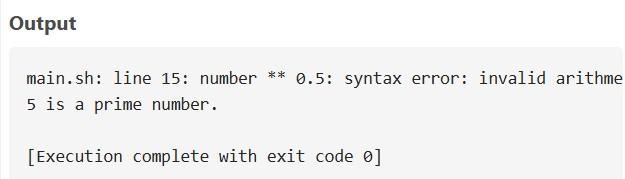
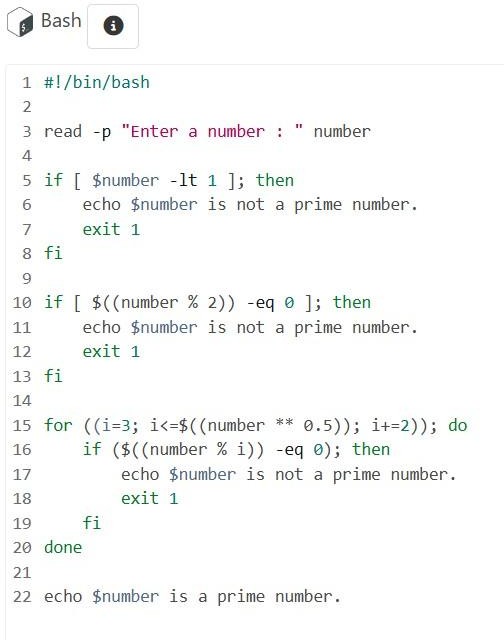
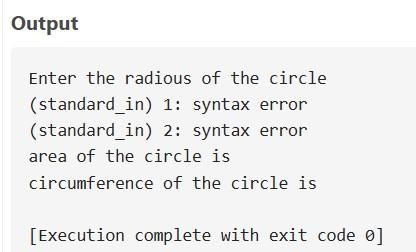
* **Purpose:** This line prints the message "Enter the year" to the terminal, prompting the user to input a year.
* **Explanation:** echo is a command used to display text or output to the terminal.
* **Read:** takes input from the user and stores it in the variable following it (in this case, y). The user is expected to enter a year after the prompt.
* **If**:This line starts an if conditional statement that checks if the value of b is equal to 0.
* **Else:**This line introduces the else block, which contains commands that will execute if the condition in the if statement is false.



### Shell Script to check no is prime or not.

**(/bin/bash):-**This line tells the system that the script should be interpreted using the Bash shell. It specifies the path to the Bash interpreter (/bin/bash), so the script is run correctly when executed in a Unix-like environment.

* **Read:-** This line prompts the user to input a number and stores the input in the variable number.
* **[ $number -lt 1 ]:-**This conditional block checks if the input number is less than 1, in which case it prints a message and exits the script.
* **$((number % 2)):-**This block checks if the number is even (i.e., divisible by 2). If the number is even, it can't be prime (except for 2), so the script prints a message and exits.
* **For loop:**- This block loops through odd numbers starting from 3 up to the square root of the number to check if any of them divides the number exactly. If any divisor is found, the number is not prime, and the script exits.
* **Eco &number:**- If the loop completes without finding a divisor, this line prints that the number is prime.



**Concept: Do while loop:-**

In Linux shell scripting (specifically in Bash), a do-while loop doesn't exist as a built-in structure like in other programming languages. However, you can emulate a do-while loop using a while loop with a slightly different structure, where the condition is checked at the end of the loop.

# Syntax:-

#!/bin/bash

# Emulating do-while loop in Bash while true; do

# Commands to execute echo "This will execute at least once"

# Condition to exit loop (it checks after the first iteration) read

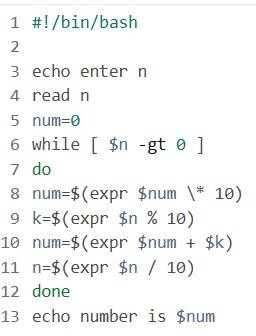
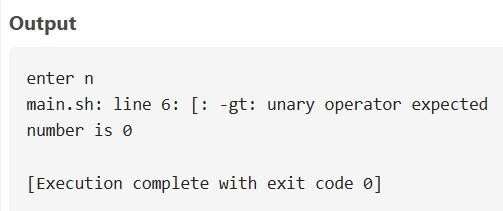
-p "Do you want to continue? (yes/no): " choice if [ "$choice"

!= "yes" ]; then break fi

done

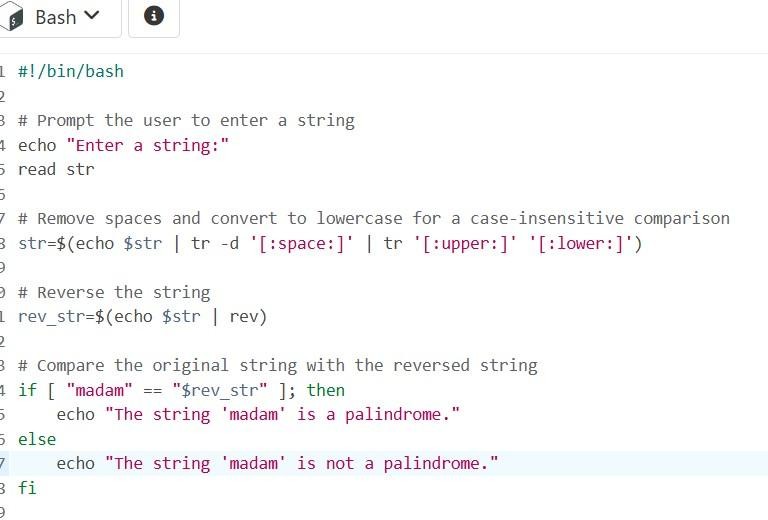
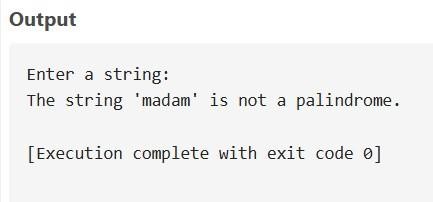
### Shell Script to check the given number and its reverse are same.

* **echo**:- The echo command simply displays the text "enter n", which acts as a prompt for the user to input a number
* **read**: The read command takes the user's input from the terminal and assigns it to the variable n..
* **num:-** is the accumulator that will be used to build the reversed number as the digits of n are processed one by one.
* **condition [ $n -gt 0 ] checks** :-if the value of n is greater than 0. The loop continues to run as long as there are still digits left in n to reverse.
* **The expr $num \\* 10:-**The expr $num \\* 10 performs the multiplication. The backslashes (\) are used to escape the \* operator so that it is treated correctly by expr.
* **expr $n % 10 calculates** :-the remainder when n is divided by 10, which gives the last digit of the number.

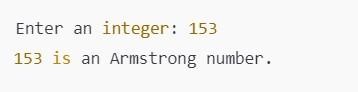
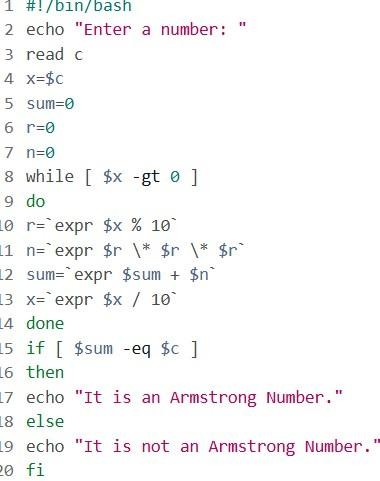
# Shell Script to check the given string is palindrome or not

* echo $str: Prints the value of str (the user input).
* echo $str: Prints the string.
* if [ "madam" == "$rev\_str" ]:- The condition checks if the string "madam" is **equal to** the value of rev\_str.
* echo "The string 'madam' is a palindrome.":- The script will output that the string "madam" is a palindrome. However, this message is hardcoded, meaning it will always display the same message regardless of the user input.

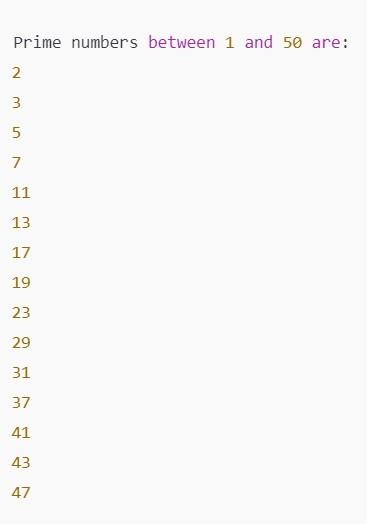
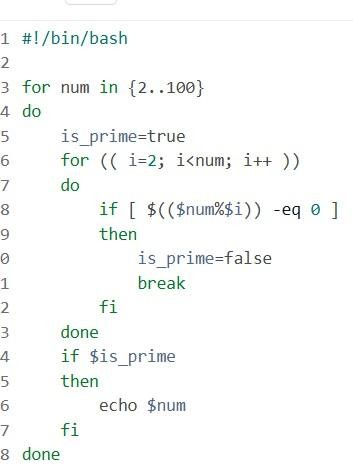
### Shell Script to check the given integer is Armstrong number or not.

* **x=$c:-** After reading the number, x is used to process and extract digits, while c will store the original number for comparison later**.**
* **sum=0 r=0 n=0:- I**nitializing variables is essential for the calculation to start with a clean slate.
* **while [ $x -gt 0 ]:-** The expression expr $x % 10 divides x by 10 and returns the remainder (i.e., the last digit).r will store this last digit of x.
* **n=`expr $r \\* $r \\* $r`:-** expr $sum + $n calculates the sum of the current sum and the cube of the digit n. sum stores the running total of the cubes of the digits.
* **x=`expr $x / 10`:-** The expression expr $x / 10 divides x by 10 and assigns the result back to x.This operation effectively shifts all digits of x to the right, dropping the last digit (since we're working with integers).



# Shell Script to generate prime numbers between 1 and 50.

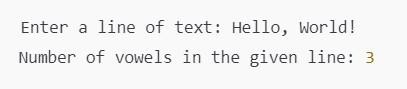
* **{2..50}:-** is a brace expansion in Bash, which expands into a list of numbers from 2 to 100.
* **is\_prime=true:-** Initially, we assume that the number (num) is prime. If we find that it is divisible by any number other than 1 and itself, we will set is\_prime to false.
* **for (( i=2; i<num; i++** )):- (( i=2; i<num; i++ )) is a C-style for loop. It initializes i to 2, runs while i is less than num, and increments i by 1 in each iteration.
* **if [ $(($num%$i)) -eq 0 ]:-** $(($num % $i)) calculates the **remainder** of dividing num by i (using the modulo operator %).
* **then is\_prime=false break:-**is\_prime=false indicates that num is **not prime** because it is divisible by i.break exits the inner loop as we no longer need to check other divisors once.



# Shell script to count the number of vowels in a line of text.

**text=$(echo "$text" | tr '[:upper:]' '[:lower:]'):-** echo "$text" outputs the user's input text. tr '[:upper:]' '[:lower:]' is a command that translates all uppercase letters to lowercase.

* **for (( i=0; i<${#text}; i++ )):-** ${#text} gives the length of the string stored in text, so the loop runs from i=0 to i being one less than the string's length.



# Concept: elif function:-

In Linux shell scripting (specifically in Bash), the elif (short for "else if") statement is used to specify additional conditions in an if-else construct. It allows you to check multiple conditions sequentially. If the first condition fails, it checks the next condition, and so on.

# Syntax:-

if [ condition1 ]; then

# Commands to execute if condition1 is true elif [ condition2 ]; then

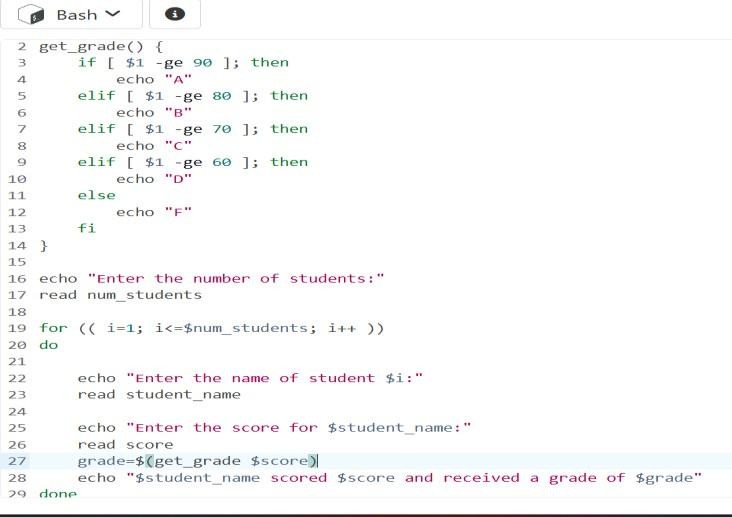
# Commands to execute if condition2 is true elif [ condition3 ]; then

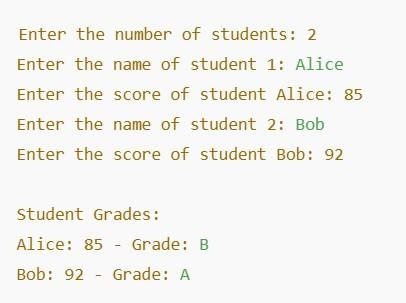
# Commands to execute if condition3 is true else # Commands to execute if all conditions are false

Fi

# Shell Script to display student grades:-

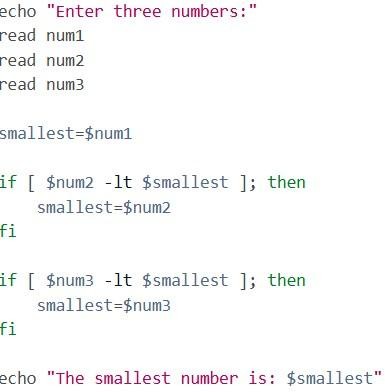
* **get\_grade() { ... }:** This is the function definition. The function get\_grade checks a numeric score passed to it and returns the corresponding grade
* **echo "Enter the number of students**:": Displays a message asking for the number of students.
* **for (( i=1; i<=$num\_students; i++ )):-**i=1: Initializes the loop variable i to 1, which will track the student number.i<=$num\_students: The loop will run as long as i is less than or equal to num\_students.

• 



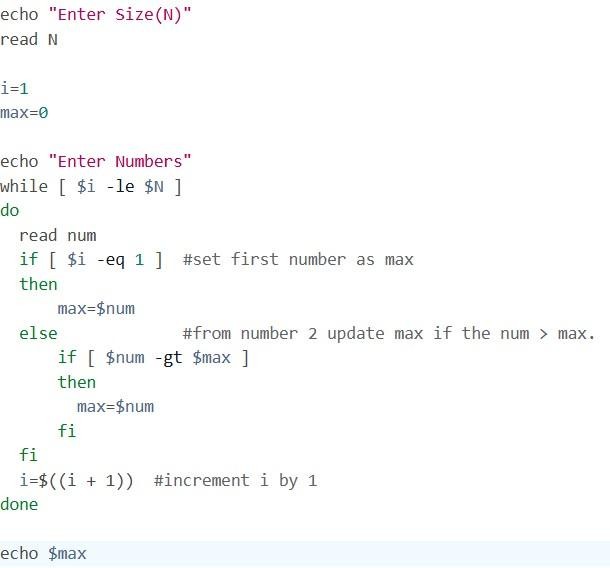
# Shell Script to find the smallest number from a set of numbers:-

* **smallest=$num1:-** The variable smallest is set to the value of num1, meaning the script starts by assuming the first number is the smallest**.**
* **if [ $num2 -lt $smallest ]; then:-** [ $num2 -lt $smallest ] is a comparison expression. It checks if the value of num2 is less than the value of smallest using the -lt (less than) operator.
* **smallest=$num2:-** If num2 is indeed smaller than smallest, then smallest is now set to the value of num2.
* **Fi:-** The fi keyword in Bash marks the end of the if block. It tells the script that the logic for comparing num2 has been completed.

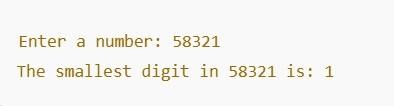
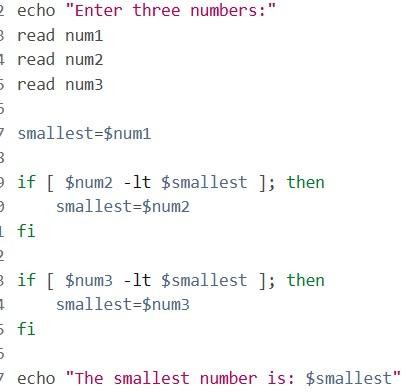
 

# Shell Script to find the largest number from a set of numbers:-

* **while [ $i -le $N ]:-** [ $i -le $N ] is a test condition. It checks if the current value of i is less than or equal to N (i.e., if we have more numbers to enter).
* **read num:-** The read command waits for the user to input a number and assigns that value to the variable num, which will be compared to max to determine if it's the largest number.
* **if [ $i -eq 1 ]:-** [ $i -eq 1 ] checks if i is equal to 1 (i.e., if it's the first number).



# Shell Script to find the smallest digit from a number.:-

* **smallest=$num1:-** The variable smallest is set to the value of num1, meaning the script starts by assuming the first number is the smallest**.**
* **if [ $num2 -lt $smallest ]; then:-** [ $num2 -lt $smallest ] is a comparison expression. It checks if the value of num2 is less than the value of smallest using the -lt (less than) operator.
* **smallest=$num2:-** If num2 is indeed smaller than smallest, then smallest is now set to the value of num2.
* **Fi:-** The fi keyword in Bash marks the end of the if block. It tells the script that the logic for comparing num2 has been completed.

## •