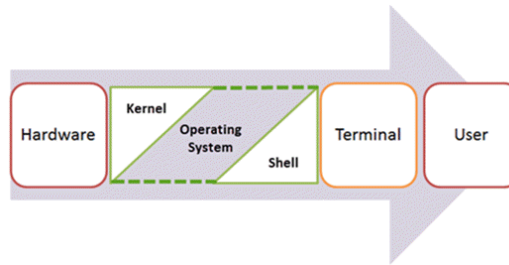


	Experiment No : 3	Date :
Title	Introduction to Shell Script (Control Statement)	
Aim	<p>Write a shell script for the following</p> <ol style="list-style-type: none"> Write Shell script to find Area, circumference and volume of the sphere Write shell script to find Simple interest Write code to demonstrate File Operators, Read file and directory anme from user Write shell script to Swap to numbers without using 3rd variable Write script for Menu Driven program to perform different arithmetic operation(Case) Write script to find maximum of three numbers(if-elif-else-fi) Write script to demonstrate all command line argument Given sides of Triangle and decide whether the triangle is isosceles, equilateral, scalene, obtuse, acute, and right Write script for same. Write script to Find whether entered number is even or odd Write script to perform following String Operation <ul style="list-style-type: none"> Find Length of String Find and Replace String To Concatenate String Reversing the string Write to check whether entered string is palindrome or not Write shells script to display menu for Celsius to Fahrenheit and Fahrenheit to Celsius, Read the temperature accordingly and convert. 	
Hardware Requirement	Personal Computer	
Software Requirement	Linux Operating System(Ubuntu 20.04) , Shell-Interpreter Nano or Vi or Vim or gedit text editor	
Theory	<p>Shell is a UNIX term for an interface between a user and an operating system service. Shell provides users with an interface and accepts human-readable commands into the system and executes those commands which can run automatically and give the program's output in a shell script.</p> <p>An Operating is made of many components, but its two prime components are –</p>	

- Kernel
- Shell



Components of Shell Program

Kernel

A Kernel is at the nucleus of a computer. It makes the communication between the hardware and software possible. While the Kernel is the innermost part of an operating system, a shell is the outermost one.

Shell

A shell in a Linux operating system takes input from you in the form of commands, processes it, and then gives an output. It is the interface through which a user works on the programs, commands, and scripts. A shell is accessed by a terminal which runs it.

When you run the terminal, the Shell issues a command prompt (usually \$), where you can type your input, which is then executed when you hit the Enter key. The output or the result is thereafter displayed on the terminal.

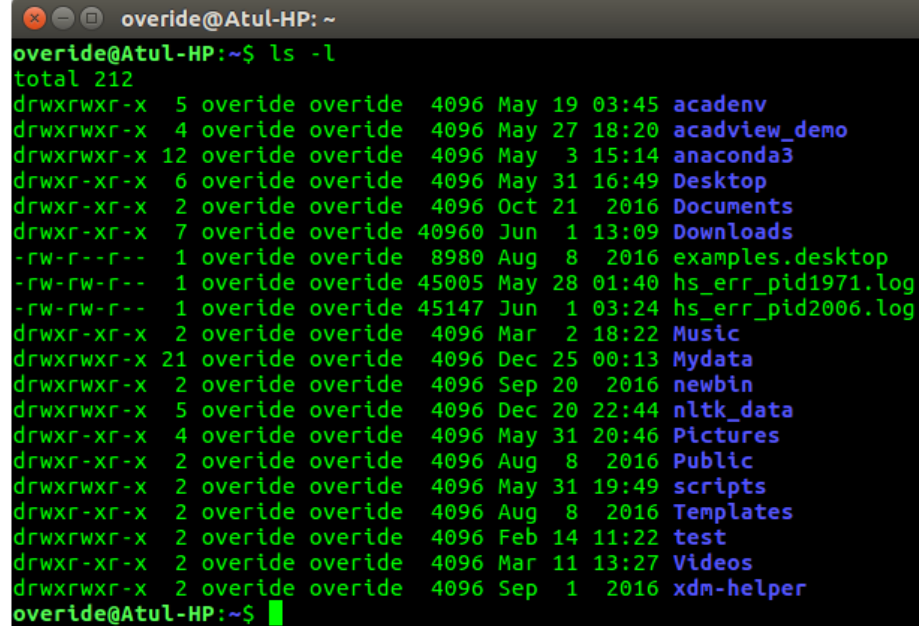
The Shell wraps around the delicate interior of an Operating system protecting it from accidental damage. Hence the name Shell.

Shell is broadly classified into two categories –

- Command Line Shell
- Graphical shell

Command Line Shell

Shell can be accessed by users using a command line interface. A special program called Terminal in Linux/macOS, or Command Prompt in Windows OS is provided to type in the human-readable commands such as "cat", "ls" etc. and then it is being executed. The result is then displayed on the terminal to the user. A terminal in Ubuntu 20.02 system looks like this –



```

override@Atul-HP: ~
override@Atul-HP:~$ ls -l
total 212
drwxrwxr-x  5 override override 4096 May 19 03:45 acadenv
drwxrwxr-x  4 override override 4096 May 27 18:20 acadview_demo
drwxrwxr-x 12 override override 4096 May  3 15:14 anaconda3
drwxr-xr-x  6 override override 4096 May 31 16:49 Desktop
drwxr-xr-x  2 override override 4096 Oct 21 2016 Documents
drwxr-xr-x  7 override override 40960 Jun  1 13:09 Downloads
-rw-r--r--  1 override override 8980 Aug  8 2016 examples.desktop
-rw-rw-r--  1 override override 45005 May 28 01:40 hs_err_pid1971.log
-rw-rw-r--  1 override override 45147 Jun  1 03:24 hs_err_pid2006.log
drwxr-xr-x  2 override override 4096 Mar  2 18:22 Music
drwxrwxr-x 21 override override 4096 Dec 25 00:13 Mydata
drwxrwxr-x  2 override override 4096 Sep 20 2016 newbin
drwxrwxr-x  5 override override 4096 Dec 20 22:44 nltk_data
drwxr-xr-x  4 override override 4096 May 31 20:46 Pictures
drwxr-xr-x  2 override override 4096 Aug  8 2016 Public
drwxrwxr-x  2 override override 4096 May 31 19:49 scripts
drwxr-xr-x  2 override override 4096 Aug  8 2016 Templates
drwxrwxr-x  2 override override 4096 Feb 14 11:22 test
drwxr-xr-x  2 override override 4096 Mar 11 13:27 Videos
drwxrwxr-x  2 override override 4096 Sep  1 2016 xdm-helper
override@Atul-HP:~$ █

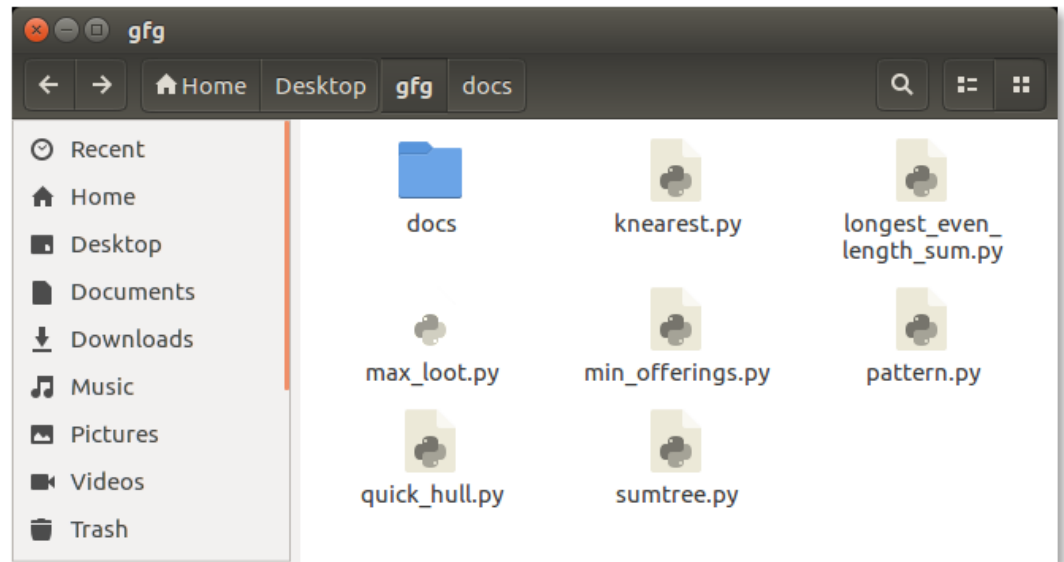
```

linux command line

In the above screenshot “**ls**” command with “**-l**” option is executed. It will list all the files in the current working directory in a long listing format. Working with a command line shell is a bit difficult for beginners because it’s hard to memorize so many commands. It is very powerful; it allows users to store commands in a file and execute them together. This way any repetitive task can be easily automated. These files are usually called batch files in Windows and **Shell Scripts** in Linux/macOS systems.

Graphical Shells

Graphical shells provide means for manipulating programs based on the graphical user interface (GUI), by allowing for operations such as opening, closing, moving, and resizing windows, as well as switching focus between windows. Window OS or Ubuntu OS can be considered as a good example which provides GUI to the user for interacting with the program. Users do not need to type in commands for every action. A typical GUI in the Ubuntu system –



GUI Shell

There are several shells available for Linux systems like –

- **BASH (Bourne Again SHell)** – It is the most widely used shell in Linux systems. It is used as default login shell in **Linux systems** and in **macOS**. It can also be installed on **Windows OS**.
- POSIX shell also is known as sh
- **CSH (C SHell)** – The C shell's syntax and its usage are very similar to the **C programming language**.
- **KSH (Korn SHell)** – The Korn Shell was also the base for the **POSIX Shell** standard specifications etc.

Each shell does the same job but understands different commands and provides different built-in functions.

The C shell: The prompt for this shell is %, and its subcategories are:

- C shell also is known as csh
- Tops C shell also is known as tcsh

Terminal

A program which is responsible for providing an interface to a user so that he/she can access the shell. It basically allows users to enter commands and see the output of those commands in a text-based interface. Large scripts that are written to automate and perform complex tasks are executed in the terminal. To access the terminal, simply search in search box "terminal" and double-click it.

Shell Scripting

Usually, shells are interactive, which means they accept commands as input from users and execute them. However, sometimes we want to execute a bunch of commands routinely, so we have to type in all commands each time in the terminal.

As a shell can also take commands as input from file, we can write these commands in a file and can execute them in shell to avoid this repetitive work. These files are called **Shell Scripts** or **Shell Programs**. Shell scripts are similar to the batch file in MS-DOS. Each shell script is saved with `.sh`` file extension e.g., **myscript.sh**.

A shell script has syntax just like any other programming language. If you have any prior experience with any programming language like Python, C/C++ etc. It would be very easy to get started with it.

A shell script comprises the following elements –

- **Shell Keywords** – if, else, break etc.
- **Shell commands** – cd, ls, echo, pwd, touch etc.
- **Functions**
- **Control flow** – if..then..else, case and shell loops etc.

Need of shell scripts

There are many reasons to write shell scripts:

- To avoid repetitive work and automation
- System admins use shell scripting for routine backups.
- System monitoring
- Adding new functionality to the shell etc.

Some Advantages of shell scripts

- The command and syntax are exactly the same as those directly entered in the command line, so programmers do not need to switch to entirely different syntax
- Writing shell scripts are much quicker
- Quick start
- Interactive debugging etc.

Some Disadvantages of shell scripts

- Prone to costly errors, a single mistake can change the command which might be harmful.
- Slow execution speed
- Design flaws within the language syntax or implementation
- Not well suited for large and complex task

- Provide minimal data structure unlike other scripting languages. etc.

Significance of #!/bin/bash:

This is known as **shebang** in Unix. Shebang is a collection of characters or letters that consist of a number sign and exclamation mark, that is **(#!)** at the beginning of a script. Shebang is a combination of bash # and bang ! followed the the bash shell path. The main function of a shell is to interpret the UNIX commands given by the user. Unix consists of numerous shells out of that bash is one of them that is widely used. It is the default shell assigned by Linux-based operating systems.

A script comprises several UNIX commands. Now, in each of the scripts, users are required to explicitly specify the type of shell they want to use to run their scripts. Now to explicitly specify the type of shell used by the script, Shebang is used. So we can use shebang, that is, **#!/bin/bash** at the start or top of the script to instruct our system to use bash as a default shell.

Difference between #!/bin/bash and #!/bin/sh

The shebang, **#!/bin/bash** when used in scripts is used to instruct the operating system to use bash as a command interpreter. Each of the systems has its own shells which the system will use to execute its own system scripts.

This system shell can vary from OS to OS(most of the time it will be bash). Whereas, when the shebang, **#!/bin/sh** used in scripts instructs the internal system shell to start interpreting scripts.

Below are some of the shebangs used for different purposes in shell scripts:

#!/bin/sh: It is used to execute the file using sh, which is a Bourne shell, or a compatible shell

#!/bin/csh: It is used to execute the file using csh, the C shell, or a compatible shell.

#!/usr/bin/perl -T: It is used to execute using Perl with the option for taint checks.

#!/usr/bin/php: It is used to execute the file using the PHP command-line interpreter.

#!/usr/bin/python -O: It is used to execute using Python with optimizations to code.

#!/usr/bin/ruby: It is used to execute using Ruby.

	<p>Difference between single quotes (') and double quotes (") in shell scripting</p> <p>Single quotes (') and double quotes (") are used to enclose strings in shell scripting, but they have different behaviors:</p> <ul style="list-style-type: none"> • Single quotes: Everything between single quotes is treated as a literal string. Variable names and most special characters are not expanded. • Double quotes: Variables and certain special characters within double quotes are expanded. The contents are subject to variable substitution and command substitution. <p>Comments in bash scripting</p> <p>Comments start with a # in bash scripting. This means that any line that begins with a # is a comment and will be ignored by the interpreter. Comments are very helpful in documenting the code, and it is a good practice to add them to help others understand the code.</p> <p>Variable naming conventions</p> <p>In Bash scripting, the following are the variable naming conventions:</p> <ul style="list-style-type: none"> • Variable names should start with a letter or an underscore (_). • Variable names can contain letters, numbers, and underscores (_). • Variable names are case-sensitive. • Variable names should not contain spaces or special characters. • Use descriptive names that reflect the purpose of the variable. • Avoid using reserved keywords, such as if, then, else, fi, and so on as variable names.
Commands	<p>Command used in shell script</p> <p>read command</p> <p>The read command takes the user input and splits the string into fields, assigning each new word to an argument. If there are fewer variables than words, read stores the remaining terms into the final variable. Specifying the argument names is optional. The command stores a user's input into the \$REPLY variable by default.</p>

- **read -p**

The '-p' option is used for the **prompt text**. It reads the data along with some hint text. This hint text helps us while entering the text such as what to enter.

- **Default Behaviour**

If we pass the read command without any argument, it will take a line as user input and store it in a built-in variable '**REPLY**'.

- **Specify the variables to store the values**

We can specify the variables to store the input. If the number of specified variables is lower than the entered words, it will store all the remaining words in the last variable by default.

- **The Internal Field Separator**

The internal field separator (IFS) is used to separate the output fields and determines the word boundaries in a given line. We can set it according to our needs.

- **read -n**

The '-n' option limits the length of the character in the entered text. It will not let you enter text more than the specified number of characters. After reaching the limit of characters, it automatically stops reading.

- **read -s**

The '-s' option is used for **security purpose**. It is used to read the sensitive data. By using this option, the entered text won't appear in the terminal. We can use other options with this option. Characters are read in this option. Primarily, it is used to read the passwords from the keyboard.

Echo command

- In Linux, the echo command can be used for displaying a line of string/text that is passed as the **arguments**. This command is a built-in that is mostly and widely used in various batch files and shell scripts to outcome status test to a file and screen.
- Using option '**\b**' – backspace with backslash interpreter '**-e**' which removes all the spaces in between.
- Using option '**\n**' – New line with backspace interpreter '**-e**' treats new line from where it is used.
- Using option '**\t**' – horizontal tab with backspace interpreter '**-e**' to have horizontal tab spaces.

- Using option '\v' – vertical tab with backspace interpretor '-e' to have vertical tab spaces.

Chmod command

Linux chmod command is used to change the access permissions of files and directories. It stands for **change mode**. It can not change the permission of symbolic links. Even, it ignores the symbolic links come across recursive directory traversal.

In the Linux file system, each file is associated with a particular owner and have permission access for different users. The user classes may be:

- owner
- group member
- Others (Everybody else)

The file permissions in Linux are the following three types:

- read (r)
- write (w)
- execute (x)

The following table represents the digits and their permissions:

Digits	Permissions
000	No permission
001	Execute permission
010	Write permission
011	Write and execute permissions
100	Read permission
101	Read and execute permissions
110	Read and write permissions
111	Read, write, and execute permissions

Command Line Arguments

- Command-line arguments are parameters that are passed to a script while executing them in the bash shell.
- They are also known as positional parameters in Linux.
- We use command-line arguments to denote the position in memory

where the command and it's associated parameters are stored.

- A bash shell script have parameters. These parameters start from **\$1** to **\$9**.
- When we pass arguments into the command line interface, a positional parameter is assigned to these arguments through the shell.
- The first argument is assigned as \$1, second argument is assigned as \$2 and so on...
- If there are more than 9 arguments, then **tenth** or onwards arguments can't be assigned as \$10 or \$11.



Shell Parameters

Parameters	Function
\$1-\$9	Represent positional parameters for arguments one to nine
\${10}-\${n}	Represent positional parameters for arguments after nine
\$0	Represent name of the script
\$*	Represent all the arguments as a single string
\$@	Same as \$*, but differ when enclosed in ("")
\$#	Represent total number of arguments
\$\$	PID of the script
\$?	Represent last return code

Control Statement in Shell Script

Conditional Statements

There are total 5 conditional statements which can be used in bash programming

1. if statement
2. if-else statement
3. if..elif..else..fi statement (Else If ladder)
4. if..then..else..if..then..fi..fi..(Nested if)
5. switch statement

Their description with syntax is as follows:

if statement

This block will process if specified condition is true.

Syntax

```
if [ expression ]
then
    statement
fi
```

if-else statement

If specified condition is not true in if part then else part will be execute.

Syntax

```
if [ expression ]
then
    statement1
else
    statement2
fi
```

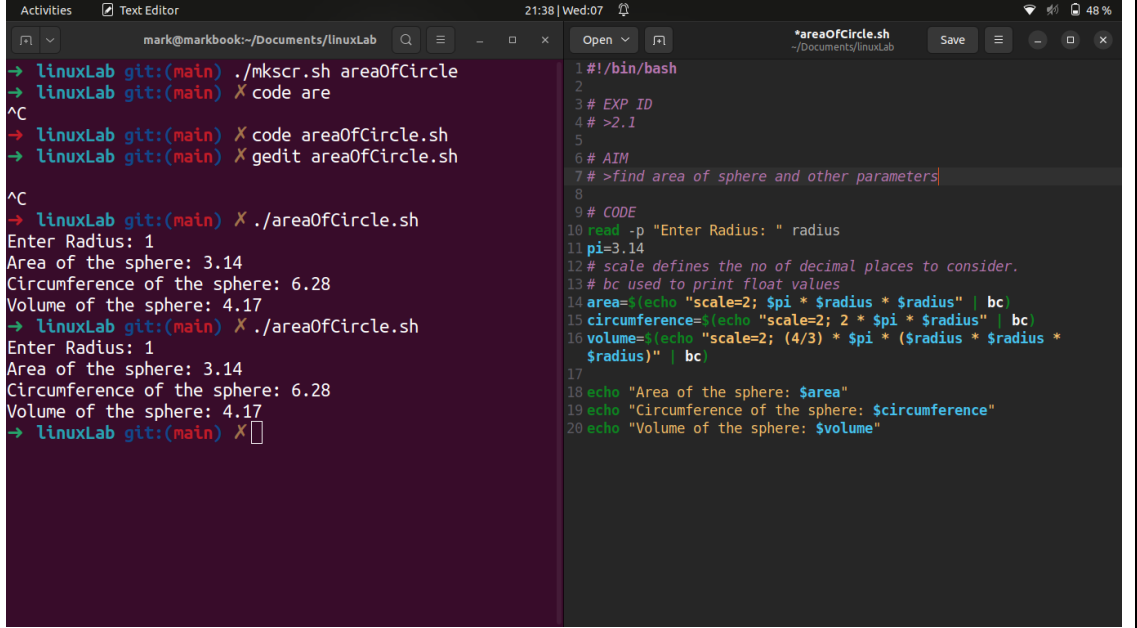
if..elif..else..fi statement (else If ladder)

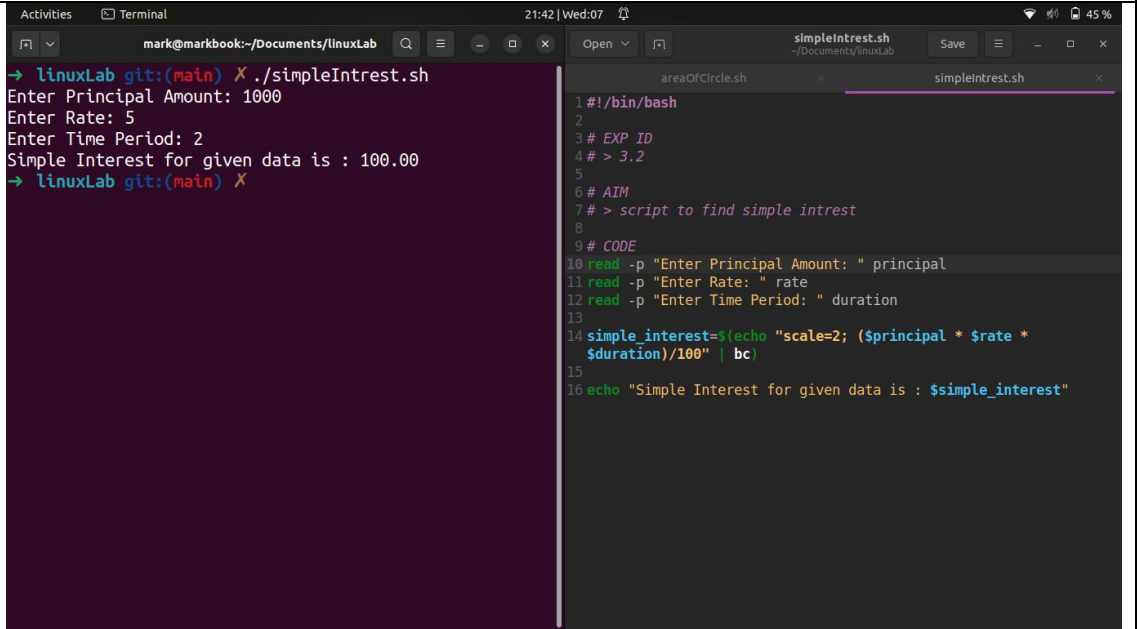
To use multiple conditions in one if-else block, then elif keyword is used in shell. If expression1 is true then it executes statement 1 and 2, and this process continues. If none of the condition is true then it processes else part.

Syntax

```
if [ expression1 ]
then
    statement1
    statement2
    .
    .
elif [ expression2 ]
then
    statement3
    statement4
    .
```

	<pre> . else statement5 fi </pre> <p><u>if..then..else..if..then..fi..fi..(Nested if)</u></p> <p>Nested if-else block can be used when, one condition is satisfies then it again checks another condition. In the syntax, if expression1 is false then it processes else part, and again expression2 will be check.</p> <p><i>Syntax:</i></p> <pre> if [expression1] then statement1 statement2 . else if [expression2] then statement3 . fi fi </pre> <p><u>switch statement</u></p> <p>case statement works as a switch statement if specified value match with the pattern then it will execute a block of that particular pattern When a match is found all of the associated statements until the double semicolon (;;) is executed. A case will be terminated when the last command is executed. If there is no match, the exit status of the case is zero.</p> <p><i>Syntax:</i></p> <pre> case in Pattern 1) Statement 1;; Pattern n) Statement n;; esac </pre>
Script Statement	a.Write Shell script to find Area, circumference and volume of the sphere
Script Code	#!/bin/bash

	<pre> # EXP ID # >3.1 # AIM # >find area of sphere and other parameters # CODE read -p "Enter Radius: " radius pi=3.14 # scale defines the no of decimal places to consider. # bc used to print float values area=\$(echo "scale=2; \$pi * \$radius * \$radius" bc) circumference=\$(echo "scale=2; 2 * \$pi * \$radius" bc) volume=\$(echo "scale=2; (4/3) * \$pi * (\$radius * \$radius * \$radius)" bc) echo "Area of the sphere: \$area" echo "Circumference of the sphere: \$circumference" echo "Volume of the sphere: \$volume" </pre>
Output	 <pre> Activities Text Editor 21:38 Wed:07 mark@markbook:~/Documents/linuxLab → linuxLab git:(main) ./mkscri.sh areaOfCircle → linuxLab git:(main) X code are ^C → linuxLab git:(main) X code areaOfCircle.sh → linuxLab git:(main) X gedit areaOfCircle.sh ^C → linuxLab git:(main) X ./areaOfCircle.sh Enter Radius: 1 Area of the sphere: 3.14 Circumference of the sphere: 6.28 Volume of the sphere: 4.17 → linuxLab git:(main) X ./areaOfCircle.sh Enter Radius: 1 Area of the sphere: 3.14 Circumference of the sphere: 6.28 Volume of the sphere: 4.17 → linuxLab git:(main) X </pre> <pre> 1 #!/bin/bash 2 3 # EXP ID 4 # >3.1 5 6 # AIM 7 # >find area of sphere and other parameters 8 9 # CODE 10 read -p "Enter Radius: " radius 11 pi=3.14 12 # scale defines the no of decimal places to consider. 13 # bc used to print float values 14 area=\$(echo "scale=2; \$pi * \$radius * \$radius" bc) 15 circumference=\$(echo "scale=2; 2 * \$pi * \$radius" bc) 16 volume=\$(echo "scale=2; (4/3) * \$pi * (\$radius * \$radius * \$radius)" bc) 17 18 echo "Area of the sphere: \$area" 19 echo "Circumference of the sphere: \$circumference" 20 echo "Volume of the sphere: \$volume" </pre>
Script Statement	b.Write shell script to find Simple interest
Script Code	<pre> #!/bin/bash # EXP ID </pre>

	<pre> # > 3.2 # AIM # > script to find simple intrest # CODE read -p "Enter Principal Amount: " principal read -p "Enter Rate: " rate read -p "Enter Time Period: " duration simple_interest=\$(echo "scale=2; (\$principal * \$rate * \$duration)/100" bc) echo "Simple Interest for given data is : \$simple_interest" </pre>
Output	 <p>The screenshot shows a terminal window with the following content:</p> <pre> → linuxLab git:(main) X ./simpleIntrest.sh Enter Principal Amount: 1000 Enter Rate: 5 Enter Time Period: 2 Simple Interest for given data is : 100.00 → linuxLab git:(main) X </pre> <p>On the right side of the terminal window, the script's source code is visible, matching the code in the first row of the table.</p>
Script Statement	<p>m. Write code to demonstrate File Operators, Read file and directory name from user</p> <pre> #!/bin/bash # EXP ID # > 3.3 # AIM # > Write code to demonstrate File Operators, Read file and directory name from user # CODE read -p "Enter a file or directory name: " file_or_directory </pre>

```
if [ -e "$file_or_directory" ]; then
    echo "File or directory '$file_or_directory' exists."

    if [ -f "$file_or_directory" ]; then
        echo "'$file_or_directory' is a regular file."

        if [ -s "$file_or_directory" ]; then
            echo "'$file_or_directory' is not empty."
        else
            echo "'$file_or_directory' is empty."
        fi

        if [ -x "$file_or_directory" ]; then
            echo "You have execute permission on '$file_or_directory'."
        else
            echo "You do not have execute permission on '$file_or_directory'."
        fi
    fi

    if [ -d "$file_or_directory" ]; then
        echo "'$file_or_directory' is a directory."

        if [ -x "$file_or_directory" ]; then
            echo "You have execute permission on '$file_or_directory'."
        else
            echo "You do not have execute permission on '$file_or_directory'."
        fi
    fi

    if [ -r "$file_or_directory" ]; then
        echo "You have read permission on '$file_or_directory'."
    else
        echo "You do not have read permission on '$file_or_directory'."
    fi

    if [ -w "$file_or_directory" ]; then
        echo "You have write permission on '$file_or_directory'."
    else
        echo "You do not have write permission on '$file_or_directory'."
    fi
else
    echo "File or directory '$file_or_directory' does not exist."
```

	<div>fi</div> 
Script Statement	<p>n.</p> <p>o. Write shell script to Swap to numbers without using 3rd variable</p> <pre>#!/bin/bash # EXP ID # > 3.4 # AIM # > Swap two numbers without using 3rd variable # CODE read -p "Enter the first number: " num1 read -p "Enter the second number: " num2 echo "Before swapping: num1=\$num1, num2=\$num2" num1=\$((num1 + num2)) num2=\$((num1 - num2)) num1=\$((num1 - num2)) echo "After swapping: num1=\$num1, num2=\$num2"</pre>

	 <pre> → linuxLab git:(main) X ./mkscr.sh swapNumbers → linuxLab git:(main) X ./swapNumbers.sh Enter the first number: 23 Enter the second number: 16 Before swapping: num1=23, num2=16 After swapping: num1=16, num2=23 → linuxLab git:(main) X 1 #!/bin/bash 2 3 # EXP ID 4 # > 3.4 5 6 # AIM 7 # > Swap two numbers without using 3rd variable 8 9 # CODE 10 read -p "Enter the first number: " num1 11 read -p "Enter the second number: " num2 12 13 echo "Before swapping: num1=\$num1, num2=\$num2" 14 15 num1=\$((num1 + num2)) 16 num2=\$((num1 - num2)) 17 num1=\$((num1 - num2)) 18 19 echo "After swapping: num1=\$num1, num2=\$num2" </pre>
Script Statement	<p>p. Write script for Menu Driven program to perform different arithmetic operation(Case)</p> <p>q.</p>
	<pre> #!/bin/bash # EXP ID # > 3.5 # AIM # > Menu Driven program to perform different arithmetic operation(Case) # CODE while true; do echo "Menu:" echo "1. Addition" echo "2. Subtraction" echo "3. Multiplication" echo "4. Division" echo "5. Exit" read -p "Enter your choice (1-5): " choice case \$choice in 1) read -p "Enter first number: " num1 </pre>

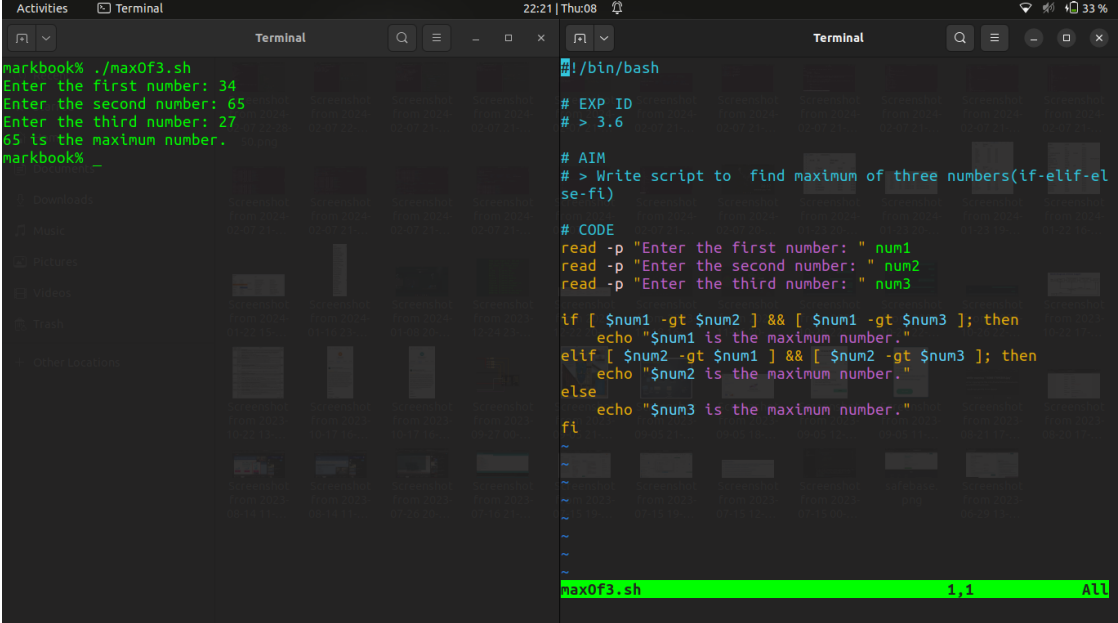
```

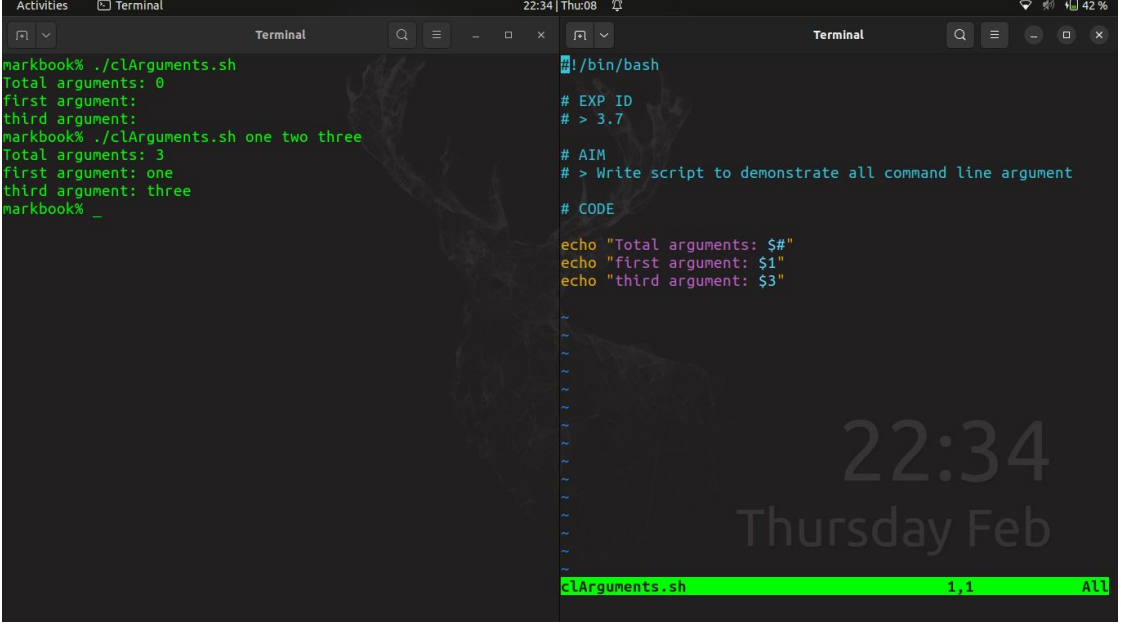
        read -p "Enter second number: " num2
        result=$((num1 + num2))
        echo "Result: $result"
        ;;
2)
    read -p "Enter first number: " num1
    read -p "Enter second number: " num2
    result=$((num1 - num2))
    echo "Result: $result"
    ;;
3)
    read -p "Enter first number: " num1
    read -p "Enter second number: " num2
    result=$((num1 * num2))
    echo "Result: $result"
    ;;
4)
    read -p "Enter dividend: " num1
    read -p "Enter divisor: " num2
    # result=$((num1 / num2))
    echo "Result: "
    result= printf "%f \n" $((10**3 * $num1/$num2))e-3

    ;;
5)
    # Exit
    echo "Exit."
    exit 0
    ;;
*)
    # Invalid choice
    echo "Invalid choice. Please enter a number between 1 and 5."
    ;;
esac
done

```

	 <pre> → linuxLab git:(main) X ./arithmeticOperation.sh Menu: 1. Addition 2. Subtraction 3. Multiplication 4. Division 5. Exit Enter your choice (1-5): 4 Enter dividend: 23 Enter divisor: 5 4.600000 Result: Menu: 1. Addition 2. Subtraction 3. Multiplication 4. Division 5. Exit Enter your choice (1-5): r Invalid choice. Please enter a number between 1 and 5. Menu: 1. Addition 2. Subtraction 3. Multiplication 4. Division 5. Exit Enter your choice (1-5): </pre> <pre> #!/bin/bash 2 3 # EXP ID 4 # > 3.5 5 6 # AIM 7 # > Menu Driven program to perform different arithmetic operation(Case) 8 9 # CODE 10 while true; do 11 echo "Menu:" 12 echo "1. Addition" 13 echo "2. Subtraction" 14 echo "3. Multiplication" 15 echo "4. Division" 16 echo "5. Exit" 17 18 read -p "Enter your choice (1-5): " choice 19 20 case \$choice in 21 1) 22 read -p "Enter first number: " num1 23 read -p "Enter second number: " num2 24 result=\$((num1 + num2)) 25 echo "Result: \$result" 26 ;; 27 2) 28 read -p "Enter first number: " num1 29 read -p "Enter second number: " num2 </pre>
	<p>r. Write script to find maximum of three numbers(if-elif-else-fi)</p> <p>s.</p>
	<pre> #!/bin/bash # EXP ID # > 3.6 # AIM # > Write script to find maximum of three numbers(if-elif-else-fi) # CODE read -p "Enter the first number: " num1 read -p "Enter the second number: " num2 read -p "Enter the third number: " num3 if [\$num1 -gt \$num2] && [\$num1 -gt \$num3]; then echo "\$num1 is the maximum number." elif [\$num2 -gt \$num1] && [\$num2 -gt \$num3]; then echo "\$num2 is the maximum number." else echo "\$num3 is the maximum number." fi </pre>

	 <pre> markbook% ./max0f3.sh Enter the first number: 34 Enter the second number: 65 Enter the third number: 27 65 is the maximum number. markbook% _ #!/bin/bash # EXP ID # > 3.6 # AIM # > Write script to find maximum of three numbers(if-elif-else-fi) # CODE read -p "Enter the first number: " num1 read -p "Enter the second number: " num2 read -p "Enter the third number: " num3 if [\$num1 -gt \$num2] && [\$num1 -gt \$num3]; then echo "\$num1 is the maximum number." elif [\$num2 -gt \$num1] && [\$num2 -gt \$num3]; then echo "\$num2 is the maximum number." else echo "\$num3 is the maximum number." fi </pre>
	<p>t. Write script to demonstrate all command line argument u.</p>
	<pre> #!/bin/bash # EXP ID # > 3.7 # AIM # > Write script to demonstrate all command line argument # CODE echo "Total arguments: \$#\" echo "first argument: \$1\" echo "third argument: \$3\" </pre>

	
	<p>v. Given sides of Triangle and decide whether the triangle is isosceles, equilateral, scalene, obtuse, acute, and right Write script for same.</p> <p>w.</p>
	<pre>#!/bin/bash # EXP ID # > 3.x # AIM # > Given sides of Triangle and decide whether the triangle is isosceles, equilateral, scalene, obtuse, acute, and right Write script for the same. # CODE read -p "Enter the length of side 1: " side1 read -p "Enter the length of side 2: " side2 read -p "Enter the length of side 3: " side3 if ["\$side1" -eq "\$side2" -a "\$side2" -eq "\$side3"]; then echo "Equilateral Triangle" elif ["\$side1" -eq "\$side2" -o "\$side2" -eq "\$side3" -o "\$side1" -eq "\$side3"]; then echo "Isosceles Triangle" else echo "Scalene Triangle" fi if ["\$side1" -gt "\$side2"]; then</pre>

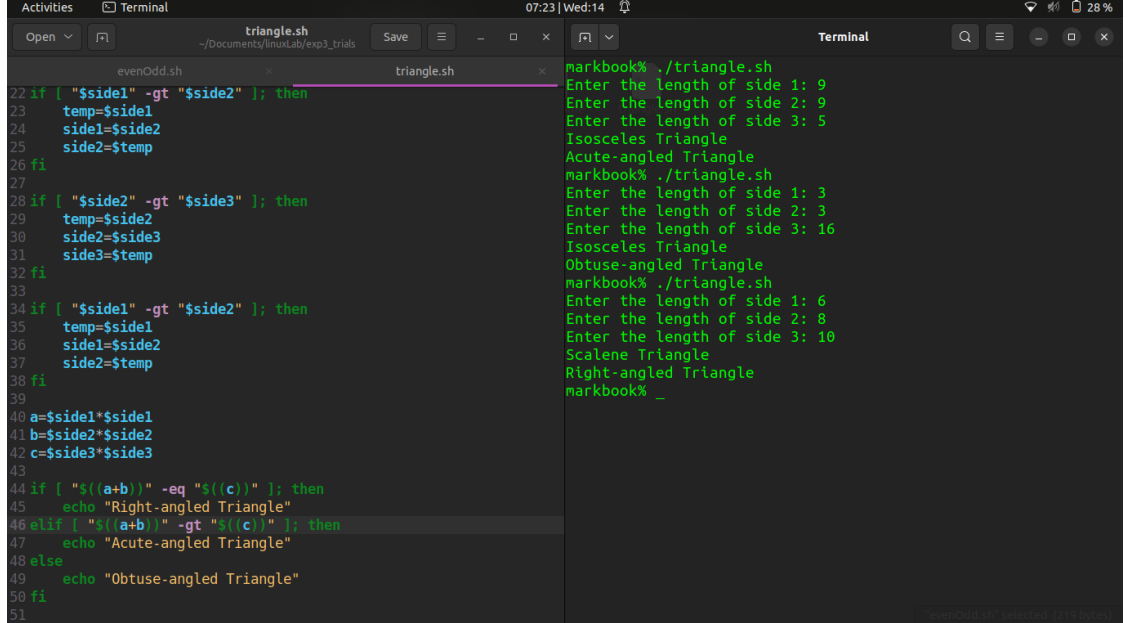
```
temp=$side1
side1=$side2
side2=$temp
fi

if [ "$side2" -gt "$side3" ]; then
temp=$side2
side2=$side3
side3=$temp
fi

if [ "$side1" -gt "$side2" ]; then
temp=$side1
side1=$side2
side2=$temp
fi

a=$side1*$side1
b=$side2*$side2
c=$side3*$side3

if [ "$((a+b))" -eq "$((c))" ]; then
echo "Right-angled Triangle"
elif [ "$((a+b))" -gt "$((c))" ]; then
echo "Acute-angled Triangle"
else
echo "Obtuse-angled Triangle"
fi
```

	 <pre> triangle.sh ~/Documents/linuxLab/exp3_trials 22 if ["\$side1" -gt "\$side2"]; then 23 temp=\$side1 24 side1=\$side2 25 side2=\$temp 26 fi 27 28 if ["\$side2" -gt "\$side3"]; then 29 temp=\$side2 30 side2=\$side3 31 side3=\$temp 32 fi 33 34 if ["\$side1" -gt "\$side2"]; then 35 temp=\$side1 36 side1=\$side2 37 side2=\$temp 38 fi 39 40 a=\$side1*\$side1 41 b=\$side2*\$side2 42 c=\$side3*\$side3 43 44 if ["\${(a+b)}" -eq "\${(c)}"]; then 45 echo "Right-angled Triangle" 46 elif ["\${(a+b)}" -gt "\${(c)}"]; then 47 echo "Acute-angled Triangle" 48 else 49 echo "Obtuse-angled Triangle" 50 fi 51 markbook% ./triangle.sh Enter the length of side 1: 9 Enter the length of side 2: 9 Enter the length of side 3: 5 Isosceles Triangle Acute-angled Triangle markbook% ./triangle.sh Enter the length of side 1: 3 Enter the length of side 2: 3 Enter the length of side 3: 16 Isosceles Triangle Obtuse-angled Triangle markbook% ./triangle.sh Enter the length of side 1: 6 Enter the length of side 2: 8 Enter the length of side 3: 10 Scalene Triangle Right-angled Triangle markbook% _ </pre>
	<p>Write script to Find whether entered number is even or odd</p>
	<pre> #!/bin/bash # EXP ID # > 3.x # AIM # > Write script to Find whether entered number is even or odd # CODE read -p "Enter a number: " number if ["\$(number % 2)" -eq 0]; then echo "Even" else echo "Odd" fi </pre>

```
Activities Terminal 07:19 | Wed:14
evenOdd.sh
~/Documents/exp3_trials
Save
Terminal

1 #!/bin/bash
2
3 # EXP ID
4 # > 3.X
5
6 # AIM
7 # > Write script to Find whether entered number is even or odd
8
9 # CODE
10 read -p "Enter a number: " number
11
12 if [ "${number%2}" -eq 0 ]; then
13     echo "Even"
14 else
15     echo "Odd"
16 fi

markbook% ./evenOdd.sh
Enter a number: 978
Even
markbook% ./evenOdd.sh
Enter a number: 45
Odd
markbook%
```

Write script to perform following String Operation

- Find Length of String
- Find and Replace String
- To Concatenate String
- Reversing the string

```
#!/bin/bash
```

```
# EXP ID
```

```
# >
```

```
# AIM
```

```
# > Write script to perform following String Operation
```

```
#Find Length of String
```

```
#Find and Replace String
```

```
#To Concatenate String
```

```
#reverse the string
```

```
# CODE
```

```
read -p "Enter string for length: " input_string
```

```
i=${#input_string}
```

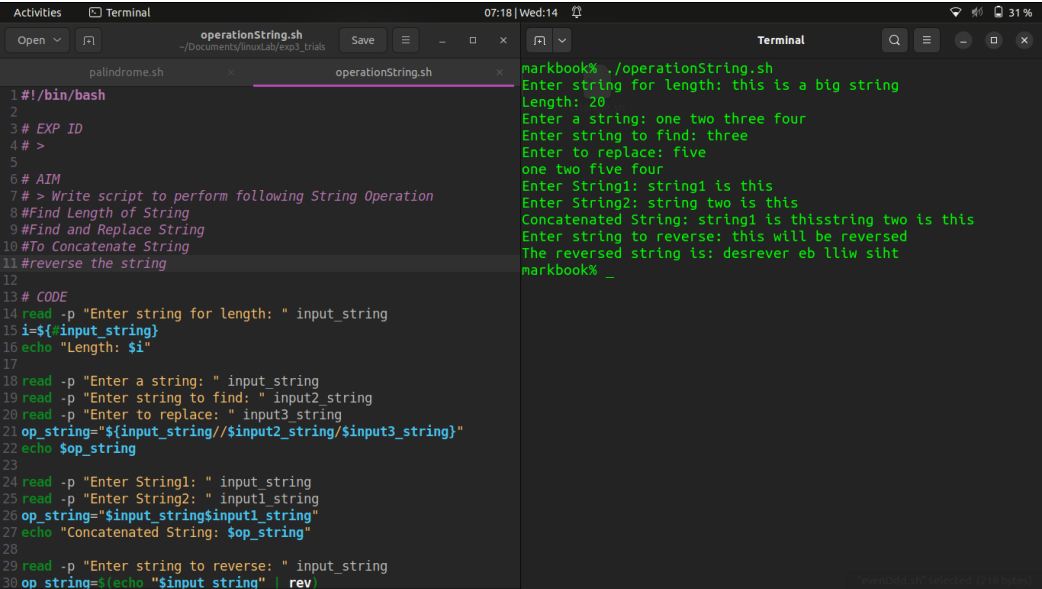
```
echo "Length: $i"
```



```
read -p "Enter a string: " input_string
read -p "Enter string to find: " input2_string
read -p "Enter to replace: " input3_string
op_string="${input_string//${input2_string}/${input3_string}}"
echo $op_string

read -p "Enter String1: " input_string
read -p "Enter String2: " input1_string
op_string="$input_string$input1_string"
echo "Concatenated String: $op_string"

read -p "Enter string to reverse: " input_string
op_string=$(echo "$input_string" | rev)
echo "The reversed string is: $op_string"
```

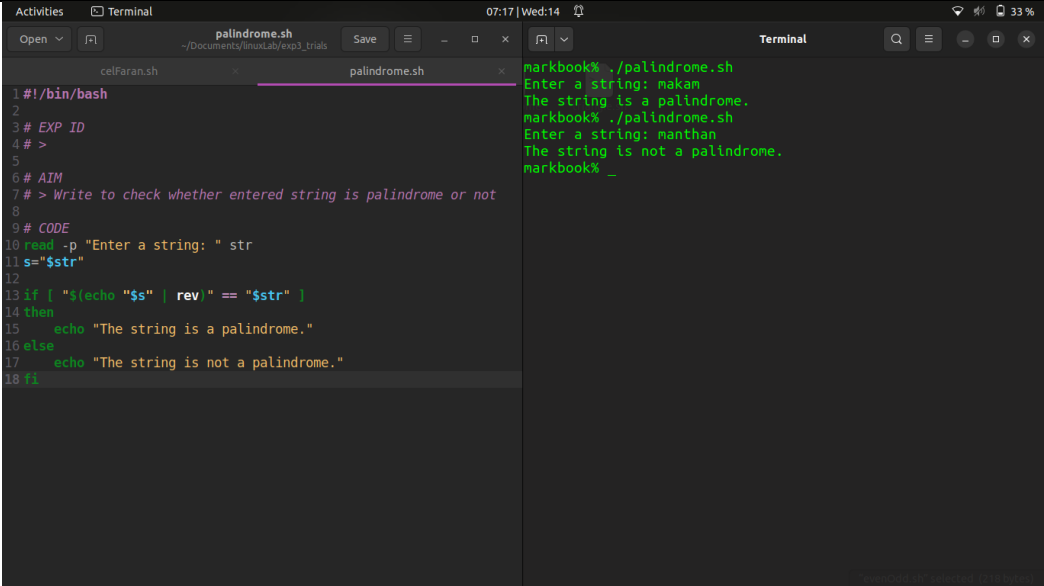


Write to check whether entered string is palindrome or not

```
#!/bin/bash

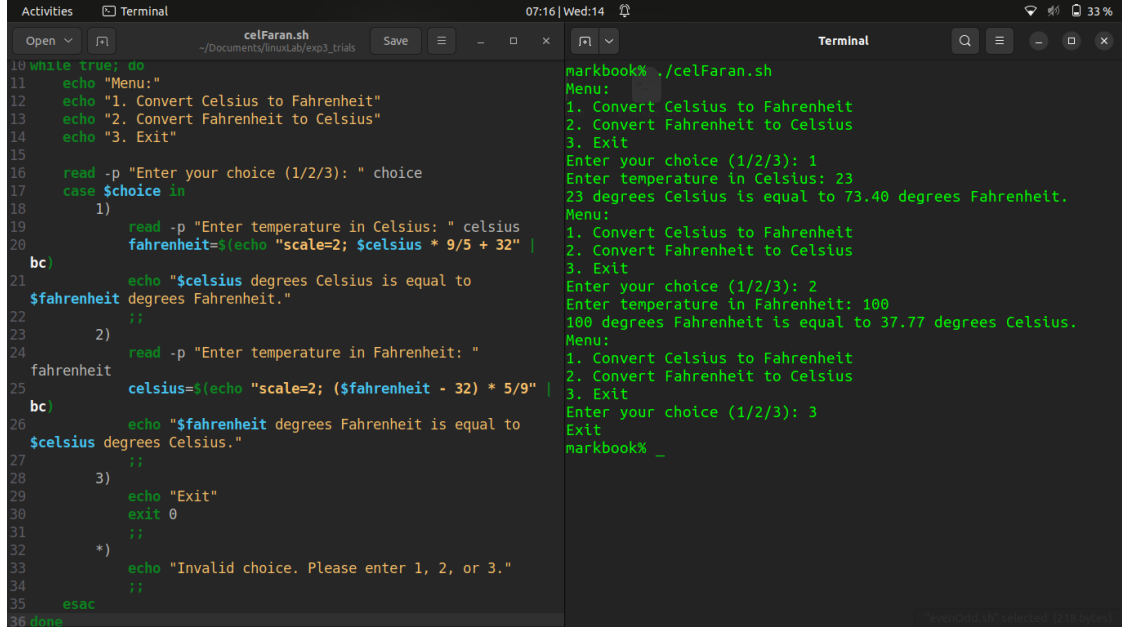
# EXP ID
# >

# AIM
# > Write to check whether entered string is palindrome or not
```

	<pre># CODE read -p "Enter a string: " str s="\$str" if ["\$(echo "\$s" rev)" == "\$str"] then echo "The string is a palindrome." else echo "The string is not a palindrome." fi</pre>
	 <p>The screenshot shows a terminal window with the following content:</p> <pre>#!/bin/bash # EXP ID # > # AIM # > Write to check whether entered string is palindrome or not # CODE 10 read -p "Enter a string: " str 11 s="\$str" 12 13 if ["\$(echo "\$s" rev)" == "\$str"] 14 then 15 echo "The string is a palindrome." 16 else 17 echo "The string is not a palindrome." 18 fi</pre> <p>Terminal output:</p> <pre>markbook% ./palindrome.sh Enter a string: makam The string is a palindrome. markbook% ./palindrome.sh Enter a string: manthan The string is not a palindrome. markbook% _</pre>
	<p>Write shells script to display menu for Celsius to Fahrenheit and Fahrenheit to Celsius, Read the temperature accordingly and convert.</p>
	<pre>#!/bin/bash # EXP ID # > # AIM # > Write shell script to display menu for Celsius to Fahrenheit and Fahrenheit to Celsius, Read the temperature accordingly and convert. # CODE while true; do</pre>

```
echo "Menu:"
echo "1. Convert Celsius to Fahrenheit"
echo "2. Convert Fahrenheit to Celsius"
echo "3. Exit"

read -p "Enter your choice (1/2/3): " choice
case $choice in
    1)
        read -p "Enter temperature in Celsius: " celsius
        fahrenheit=$(echo "scale=2; $celsius * 9/5 + 32" | bc)
        echo "$celsius degrees Celsius is equal to $fahrenheit degrees
Fahrenheit."
        ;;
    2)
        read -p "Enter temperature in Fahrenheit: " fahrenheit
        celsius=$(echo "scale=2; ($fahrenheit - 32) * 5/9" | bc)
        echo "$fahrenheit degrees Fahrenheit is equal to $celsius degrees
Celsius."
        ;;
    3)
        echo "Exit"
        exit 0
        ;;
    *)
        echo "Invalid choice. Please enter 1, 2, or 3."
        ;;
esac
done
```

	 <pre> 10 while true; do 11 echo "Menu:" 12 echo "1. Convert Celsius to Fahrenheit" 13 echo "2. Convert Fahrenheit to Celsius" 14 echo "3. Exit" 15 16 read -p "Enter your choice (1/2/3): " choice 17 case \$choice in 18 1) 19 read -p "Enter temperature in Celsius: " celsius 20 fahrenheit=\$(echo "scale=2; \$celsius * 9/5 + 32" 21 bc 22 echo "\$celsius degrees Celsius is equal to 23 \$fahrenheit degrees Fahrenheit." 24 ;; 25 2) 26 read -p "Enter temperature in Fahrenheit: " 27 fahrenheit 28 celsius=\$(echo "scale=2; (\$fahrenheit - 32) * 5/9" 29 bc 30 echo "\$fahrenheit degrees Fahrenheit is equal to 31 \$celsius degrees Celsius." 32 ;; 33 3) 34 echo "Exit" 35 exit 0 36 ;; 37 *) 38 echo "Invalid choice. Please enter 1, 2, or 3." 39 ;; 40 esac 41 done </pre> <p>markbook% ./celFaran.sh Menu: 1. Convert Celsius to Fahrenheit 2. Convert Fahrenheit to Celsius 3. Exit Enter your choice (1/2/3): 1 Enter temperature in Celsius: 23 23 degrees Celsius is equal to 73.40 degrees Fahrenheit. Menu: 1. Convert Celsius to Fahrenheit 2. Convert Fahrenheit to Celsius 3. Exit Enter your choice (1/2/3): 2 Enter temperature in Fahrenheit: 100 100 degrees Fahrenheit is equal to 37.77 degrees Celsius. Menu: 1. Convert Celsius to Fahrenheit 2. Convert Fahrenheit to Celsius 3. Exit Enter your choice (1/2/3): 3 Exit markbook% _</p>
Date	
Conclusion	Performed basic syntax related operations and control operations in bash shell scripting.
Signature	
Grade	