LAB Assign 2: Advanced Linux Command

Name: Sanket Kumbhar

Roll No.: 14

PRN No.: 12110235

1. What is meant by Shell?

I. Definition: A shell is a user interface that allows interaction with an operating system.

II. Types: There are two main types of shells:

III. Command-line shell: Text-based interface where users type commands.

IV. Graphical shell: Visual interface with windows, icons, and menus.

V. Function: The shell interprets user commands and passes them to the OS for execution.

VI. Examples: Command-line shells are common in Unix-like systems (e.g., Linux), while graphical shells are used in Windows and macOS.

VII. Purpose: Shells enable users to control and manage the OS by issuing commands and receiving output or feedback.

2. What is mean by the Shell Scripting?

I. A shell script is a text file that contains a sequence of commands for a UNIXbased operating system.

II. It is called a shell script because it combines a sequence of commands, that would otherwise have to be typed into the keyboard one at a time, into a single script.

3. Features of Shell Scripting.

Shell scripting offers several features that make it a versatile and powerful tool for automating tasks and managing operating systems. Here are some key features of shell scripting:

I. Automation: Shell scripts allow you to automate repetitive tasks, reducing manual intervention and saving time and effort.

II. Scripting Languages: Different shells provide scripting languages, such as Bash, Zsh, and PowerShell, with various programming constructs, including variables, loops, conditionals, functions, and more.

III. Interactivity: Shell scripts can be interactive, allowing users to provide input during script execution, making them adaptable to different scenarios.

IV. System Access: Shell scripts have access to system-level utilities and commands, enabling management and manipulation of files, directories, processes, and more.

V. Environment Control: Shell scripts can set up and modify the environment variables, making them useful for configuring system settings or application environments.

VI. Conditional Execution: Shell scripts support conditional statements (if-else) that enable the execution of specific commands based on certain conditions.

VII. Loops: You can use loops in shell scripts to repeat a set of commands multiple times, simplifying repetitive operations.

VIII. Error Handling: Shell scripts can handle errors and exceptions, allowing you to control the behaviour in case of failures.

IX. Command Pipelines: Shell scripts can chain multiple commands together using pipelines (e.g., `command1 | command2`), passing output from one command as input to another.

X. Script Files: Shell scripts are plain text files, making them easy to create, modify, and share. They can be version-controlled using tools like Git.

XI. Task Scheduling: Shell scripts can be scheduled to run at specific times or intervals using cron (Unix-like systems) or Task Scheduler (Windows).

XII. Portability: Shell scripts can be written to be relatively portable across different systems and environments if compatible shell features are used.

XIII. Integration: Shell scripts can integrate with other tools and programs, allowing you to build complex workflows and automate entire processes.

XIV. Rapid Prototyping: Shell scripts are quick to write, making them ideal for rapid prototyping and testing concepts before implementing them in other languages.

XV. Debugging: Shell scripts offer debugging options to help identify and fix issues, making the development process more efficient.

Overall, shell scripting provides a straightforward and efficient way to automate tasks, manage systems and streamline workflows in various computing environments.

4. Difference between scripting and programming.

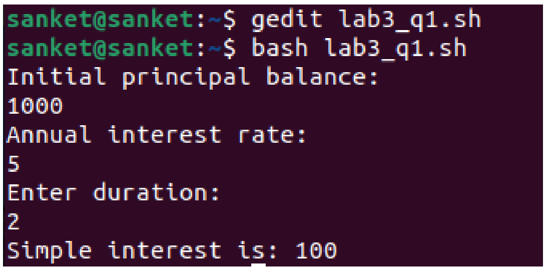
Scripting Language	Programming Language
1. A scripting language is a language that uses a naïve method to bring codes to a runtime environment.	1. A programming language is a language that is used by humans to navigate their communication with computers.
2. These are made for a particular runtime environment.	2. Programming languages are of three types: • Low-level programming language • Middle-level programming language • High-level programming language.
3. They are used to create dynamic web applications.	3. Programming languages are used to write computer programs.
4. Scripting languages can be easily ported among various operating systems.	4. Programming languages are translation-free languages.
5. These languages require a host	5. These languages are self-executable.
6. Do not create a .exe file	6. These generate .exe file
7. It does not create any binary file.	7. Creates Binary file
8. It is run inside another program	8. It is independently run

1. Calculate the Simple Interest (assuming own input) and check whether it is greater than 10000 or not (use if statement).

Code:

```
#!/bin/bash

# This is the first shell script

echo "Initial principal balance: "

read P

echo "Annual interest rate: "

read r

echo "Enter duration: "

read t

SM=$((P * r * t))

SM=$((SM / 100))

echo "Simple interest is: $SM"
```

Output:



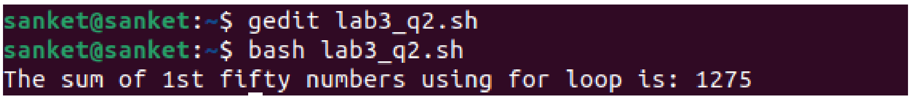
```
sanket@sanket:~$ gedit lab3_q1.sh
sanket@sanket:~$ bash lab3_q1.sh
Initial principal balance:
1000
Annual interest rate:
5
Enter duration:
2
Simple interest is: 100
```

2. Display the sum of first 50 numbers using the for loop.

Code:

```
#!/bin/bash

sum=0

for((i =1; i<=50; i++)); do

sum=$((sum +i))

done

echo "The sum of 1st fifty numbers using for loop is: $sum"
```

Output:



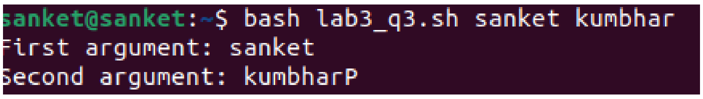
```
sanket@sanket:~$ gedit lab3_q2.sh
sanket@sanket:~$ bash lab3_q2.sh
The sum of 1st fifty numbers using for loop is: 1275
```

3. Accept and display the data through command line Arguments.

Code:

```
#!/bin/bash

# Check if there are at least two command-line arguments

if [ "$#" -lt 2 ]; then

 echo "Usage: $0 <argument1> <argument2>"

 exit 1

fi

# Access command-line arguments using $1, $2, $3, and so on

argument1="$1"

argument2="$2"

# Display the arguments

echo "First argument: $argument1"

echo "Second argument: $argument2"P
```

Output:



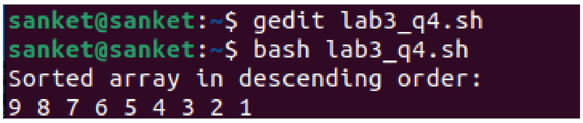
```
sanket@sanket:~$ bash lab3_q3.sh sanket kumbhar
First argument: sanket
Second argument: kumbharP
```

4. Sort array elements in descending order.

Code:

```
#!/bin/bash

# Define an array

array=(9 3 7 1 5 4 8 2 6)

# Function to sort array in descending order

sort_descending() {

 local IFS=$'\n'

 local sorted_array=($(sort -nr <<<"${array[*]}"))

 array=("${sorted_array[@]}")

}

# Call the function to sort the array in descending order

sort_descending

# Display the sorted array

echo "Sorted array in descending order:"

echo "${array[@]}"
```

Output:



```
sanket@sanket:~$ gedit lab3_q4.sh
sanket@sanket:~$ bash lab3_q4.sh
Sorted array in descending order:
9 8 7 6 5 4 3 2 1
```