	SHRI SANT GAJANAN MAHARAJ COLLEGE ENGG. SHEGAON		LABORATORY MANUAL
	PRACTICAL EXPERIMENT INSTRUCTION SHEET		
	EXPERIMENT TITLE: **Shell Scripting in Linux**		
EXPERIMENT NO.: SSGMCE/WI/ASH/01/1A6/01		ISSUE NO.:	**ISSUE DATE:**26.08.23
REV. DATE: 26.08.23	REV. NO.: 00	DEPTT. : INFORMATION TECHNOLOGY.	
LABORATORY: OPERATING SYSTEMS		SEMESTER : IV	PAGE NO. 01 OF 05

Date:

SHELL SCRIPTING IN LINUX

01. AIM :

To study shell scripting in Linux Operating System.

02. FACILITIES :

Hardware: Linux-based system with terminal access.
Software: Linux OS, text editor (Nano/Vim), Bash shell.
Resources: Shell commands, script execution tools (chmod, terminal).

03. SCOPE :

Learn shell scripting basics: commands, variables, loops, conditionals.
Automate system tasks, improve productivity, and interact with the OS efficiently.

04 THEORY:

What is a Shell?

A **shell** is a command-line interface that allows users to interact with the **operating system**. It acts as an **interpreter** between the user and the system kernel. When a user enters a command, the shell interprets it and instructs the system to execute it.

Functions of Shell:

The shell performs the following key functions:
1. **Command Execution** – Executes user commands and displays output.
2. **Scripting** – Automates tasks using shell scripts.
3. **Process Management** – Starts, stops, and monitors processes.
4. **File Management** – Allows navigation, creation, and deletion of files.
5. **Input/Output Redirection** – Redirects input/output between files and commands.

Types of Shells in UNIX/Linux:

There are several types of shells available in UNIX/Linux:

Shell	Description	Command to Check
Bourne Shell (sh)	Original UNIX shell, basic features	sh --version
Bash (Bourne	Most widely used, improved version	bash --version

Again Shell)	of sh	
C Shell (csh)	Uses C-like syntax, suited for programming	csh --version
Korn Shell (ksh)	Combines features of sh and csh	ksh --version
Z Shell (zsh)	Advanced, highly customizable shell	zsh --version

To check the default shell: `echo $SHELL`

Shell Architecture

The shell interacts with the user, system kernel, and hardware as follows:

User ↔ Shell ↔ Kernel ↔ Hardware

- **User** enters commands.
- **Shell** interprets and sends commands to the kernel.
- **Kernel** interacts with **hardware** to execute commands.

Features of a Shell:

i) Command Execution

Users can execute commands by typing them in the shell.
Example: `ls -l`
Displays a list of files in the current directory.

ii) Shell Scripting

A shell script is a text file containing multiple commands to automate tasks.
Example:

```
#!/bin/bash
echo "Hello, World!"
```

Save it as `script.sh`, make it executable `(chmod +x script.sh)`, and run it `(./script.sh)`.

iii) Wildcards

Wildcards are used for pattern matching in filenames.

Wildcard	Description	Example
*	Matches all characters	`ls *.txt`
?	Matches a single character	`ls file?.txt`
[ ]	Matches any character inside brackets	`ls file[12].txt`

iv) Input/Output Redirection

Redirection allows sending output to files or taking input from files.

Operator	Description	Example
>	Redirect output to a file (overwrite)	`ls > output.txt`
>>	Append output to a file	`ls >> output.txt`
<	Take input from a file	`sort < file.txt`
`	`	Pipe output from one command to another

v) Process Management

Shell manages processes using commands like:

- ps – List running processes.
- kill PID – Terminate a process by ID.
- & – Run a process in the background.

Example: `sleep 10 &`

Runs the sleep command in the background.

Shell Scripting:

Shell scripting automates repetitive tasks. A **shell script** contains a sequence of commands.

Execution Steps

1. Save the script as `script.sh`.
2. Make it executable: `chmod +x script.sh`
3. Run it: `./script.sh`

```
echo "Enter your name"
read name
echo "Hello, $name"
```

```
localhost:~# bash script.sh
Enter your name
Jarvis
Hello, Jarvis!
localhost:~#
```

Shell Variables:

Variables in shell scripting allow you to store data like numbers or strings, which can later be referenced or manipulated. Variables are typically assigned using = without spaces.

```
name="Alice"
echo "Hello, $name!"
```

```
Hello, Alice!
```

User Input:

To accept input from the user in shell scripts, we use the `read` command.

```
echo "What is your name?"
read name
echo "Hello, $name!"
```

```
What is your name?
Alice
Hello, Alice!
```

Conditional Statements:

Conditional statements allow decision-making in scripts. Commonly used are **`if`**, **`else`**, and **`elif`**.

```
echo "Enter a number:"
read number
if [ $number -gt 10 ]; then
    echo "Number is greater than 10."
else
    echo "Number is 10 or less."
fi
```

```
Enter a number:
15
Number is greater than 10.
```

Operator	Meaning	Operator	Meaning
-eq	Equal to	-lt	Less than
-ne	Not equal to	-ge	Greater than or equal to
-gt	Greater than	-le	Less than or equal to

Loops:

Loops in shell scripting allow you to repeat a block of code multiple times. Common loops are **`for`**, **`while`**, and **`until`**.

```
for i in {1..5}
do
    echo "Number: $i"
done
```

```
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
```

```
count=1
while [ $count -le 5 ]
do
    echo "Count: $count"
    ((count++))
done
```

```
Count: 1
Count: 2
Count: 3
Count: 4
Count: 5
```

Functions:

Functions are used to group commands into a reusable block of code.

```
greet() {
    echo "Hello, $1!"
}

greet "Bob"
```

OUTPUT:

```
Hello, Bob!
```

Case Statements:

`case` statements are used for multiple conditions. It's an alternative to multiple `if` statements when checking against a variable with many possible values.

```
echo "Enter a fruit:"
read fruit
case $fruit in
    "apple")
        echo "You chose apple!"
        ;;
    "banana")
        echo "You chose banana!"
        ;;
    *)
        echo "Unknown fruit."
        ;;
esac
```

OUTPUT:

```
Enter a fruit:
apple
You chose apple!
```

Arrays:

Arrays allow you to store multiple values in a single variable. Indices for arrays in shell scripting start from 0.

```
fruits=("apple" "banana" "cherry")
echo "First fruit: ${fruits[0]}"
echo "Second fruit: ${fruits[1]}"
```

OUTPUT:

```
First fruit: apple
Second fruit: banana
```

Arguments:

Arguments are values passed to a script when it is executed. These values can be accessed using special variables like $1, $2, etc.

```
echo "Script name: $0"
echo "Argument 1: $1"
echo "Argument 2: $2"
```

```
Script name: ./script.sh
Argument 1: arg1
Argument 2: arg2
```

08 CONCLUSION:

In conclusion, shell scripting in Linux automates tasks and streamlines system management. Through examples like conditionals, loops, and functions, it enhances efficiency and simplifies repetitive processes, making it a valuable tool for system administration. It also improves command-line proficiency and offers flexibility for a wide range of tasks.

09 VIVA QUESTIONS:

PREPARED BY **H. P. Amle**	**APPROVED BY** **HOD**