

Lab Exercise – 5

▮ AIM :: Introduction to Shell Scripting

Introduction

Shell scripting is a fundamental aspect of Unix-like operating systems (such as Linux and macOS) and serves as a bridge between users and the system kernel. A shell script is a sequence of commands written in a file, allowing users to automate tasks, run complex programs, and manipulate files and processes. The term "shell" refers to the command-line interpreter that facilitates interaction between the user and the operating system.

Shell scripts streamline routine system administration tasks, automate repetitive jobs, and enable the execution of multiple commands in sequence. This not only saves time but also reduces errors that could occur from manual execution. Shell scripts are often employed by system administrators, developers, and users to manage files, backups, network configurations, and more.

• **Purpose of Shell Scripting**

The primary purpose of shell scripting is automation. It enables users to create efficient workflows for repetitive tasks. For example, rather than executing several commands manually each time you need to back up data or clean a directory, a shell script can be used to automate these tasks. This reduces both time and effort and minimizes the chance of human error.

Another key purpose is system management. Shell scripts are used to configure servers, manage networks, and control system processes, making them an essential tool for system administrators. Moreover, shell scripting enhances task reproducibility, ensuring that procedures are consistently followed without variation.

Shell scripts also serve as a powerful tool for creating utilities and simple programs that can automate complex tasks. Developers and data scientists use them to preprocess data, compile code, or even manage and deploy software environments.

- **How Shell Scripting Works**

Shell scripting revolves around the use of commands that are interpreted by the shell, which can be thought of as a layer between the user and the operating system. The shell reads the script file line by line and executes each command in the order it appears. Shell scripts are typically written in plain text and can be created using any text editor. The most common shell interpreters are Bash (Bourne Again Shell), sh (Bourne Shell), and zsh (Z Shell).

Steps to Create and Execute a Shell Script

1. **Create a script file:** Use any text editor like vim, nano, or gedit to create a file with a .sh extension.
2. **Make the file executable:** After writing the script, you must grant it execution permissions. This can be done using the command:

```
chmod +x script_name.sh
```

3. **Run the script:** Execute the script by typing:

```
./script_name.sh
```

When a script is executed, the shell runs each command within the script sequentially. The shell interpreter also supports variables, loops, conditional statements, and functions, making scripts flexible and powerful.

- **Basic Shell Commands**

Here are a few fundamental shell commands that form the basis of shell scripting:

1. **echo:** Prints output to the terminal.

```
echo "Hello, World!"
```

2. **ls:** Lists the files and directories in the current directory.

```
ls -l
```

3. **cd:** Changes the current working directory.

```
cd /path/to/directory
```

4. **pwd:** Prints the current working directory.

```
Pwd
```

5. **cp**: Copies files or directories.

```
cp source_file  
destination_directory
```

6. **mv**: Moves or renames files and directories.

```
mv old_name new_name
```

7. **rm**: Removes files or directories.

```
rm file_name
```

8. **cat**: Displays the content of a file.

```
cat file_name
```

9. **if statements**: Used for conditional execution.

```
if [ condition ]; then  
    # Commands  
fi
```

10. **for loops**: Used for iterating over items.

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

Conclusion

Shell scripting is an essential skill for automating tasks in Unix-based systems, enabling users to execute sequences of commands, manage systems efficiently, and reduce manual effort. Its flexibility makes it ideal for a wide range of use cases, from simple file operations to complex system administration tasks.

Understanding basic commands and how to structure scripts empowers users to automate and streamline workflows, ultimately boosting productivity. Whether you're a developer, system administrator, or enthusiast, mastering shell scripting offers a powerful way to interact with and control your system more effectively.