1. What is Shell?

Introduction

Every operating system has a way for users to interact with it. In Linux, this is done through the **shell**. Think of it as the "translator" between us and the kernel.

Explanation

- The **shell** is a **command-line interpreter**.
- It takes the command you type, interprets it, and asks the kernel to execute it.
- Once executed, the shell shows the result back to you.

Without a shell, you would need to give instructions in binary or machine code — which is not practical.

Example

When you type ls in the terminal:

1. The shell interprets ls.
2. It asks the kernel to fetch the list of files.
3. The kernel gets the data from the file system.
4. The shell displays it on your screen.

Types of Shells

- **Bash (Bourne Again Shell):** Default in most Linux systems, feature-rich, located at /bin/bash.
- **Zsh (Z Shell):** Modern, supports plugins, themes, and better auto-completion.
- **Csh (C Shell):** Syntax similar to C programming, supports aliases and history.

To see your default shell:

echo $SHELL

Task

- Check the shells installed on your system using:

  cat /etc/shells

- Find out your current default shell using:

  echo $SHELL

DevOps Relevance

In DevOps, most automation is written in **Bash scripts** because it is available by default in almost every Linux distribution.

2. Writing Shell Scripts

Introduction

Typing commands one by one is fine, but what if you want to repeat tasks daily? That's where **shell scripts** help.

Explanation

- A **shell script** is a file containing a list of commands.
- The system executes them one after another.
- It makes automation possible.

Key Points

1. **Script File Naming Conventions:**
    - Not mandatory, but best practice is to use .sh (e.g., backup.sh).
2. **Comments:**
    - Start with #.
    - Help document your script.
3. **Shebang (#!/bin/bash):**
    - Tells the system which shell to use to run the script.

Example

```
#!/bin/bash
# This script prints a greeting
echo "Hello, Linux World!"
```

Steps to Run

1. Save the file as myscript.sh.
2. Make it executable:

   chmod +x myscript.sh
3. Run it:

   ./myscript.sh

Task

- Write a script called welcome.sh that prints:
  "Welcome to Shell Scripting!"

DevOps Relevance

In DevOps pipelines, shell scripts are used to set up servers, deploy applications, and run automated jobs.

3. Scripting Elements

Introduction

Shell scripts are more than just lists of commands — they support variables, input arguments, escape characters, and arithmetic operations.

A. Variables

- **Local variables**: available only inside the script/session.

  name="Student"

  echo "Hello $name"

- **Environment variables**: accessible to all programs.

  export PATH=$PATH:/home/student/bin

In scripts, always quote variables "${var}" to avoid errors.

B. Command-Line Arguments

- $0 → script name
- $1, $2 → first and second arguments
- $@ → all arguments
- $# → number of arguments

Example:

#!/bin/bash

echo "Script name: $0"

echo "First argument: $1"

echo "Total arguments: $#"

Run:

./test.sh one two

Output:

Script name: ./test.sh

First argument: one

Total arguments: 2

C. Escape Characters

- \n → new line
- \t → tab

Example:

echo -e "Name:\tJohn\nAge:\t25"

D. Arithmetic Operations

- Using expr:

  expr 5 + 3
- Using let:

  let sum=5+3

  echo $sum
- Using (( )):

  a=10; b=20

  echo $((a+b))

Task

- Write a script math.sh that:
  - Accepts two numbers as input.
  - Prints their sum, difference, product, and quotient.

DevOps Relevance

Arithmetic and variables are often used in scripts to manage server resources, loop over files, or calculate metrics during deployments.

4. String Handling

Introduction

In shell scripting, strings are as important as numbers. Logs, configs, and system outputs are often text-based.

Explanation

1. **Concatenation:**

   ```
   str1="Hello"
   str2="World"
   echo "$str1 $str2"
   ```

2. **Substring Extraction:**

   ```
   str="LinuxScripting"
   echo ${str:0:5}   # Linux
   ```

3. **String Length:**

   ```
   str="HelloWorld"
   echo ${#str}   # 10
   ```

Pattern Matching Tools

- **grep:** Search inside files.

  ```
  grep "error" logfile.txt
  ```

- **cut:** Extract fields by delimiter.

  ```
  cut -d"," -f1 file.csv
  ```

- **awk:** Process columns of text.

  ```
  awk '{print $2}' file.txt
  ```

- **sed:** Find and replace.

  ```
  sed 's/old/new/g' file.txt
  ```

Task

- Write a script stringops.sh that:
    - Accepts a string.
    - Prints its length.
    - Prints first 5 characters.
    - Replaces the word "Linux" with "Unix" in a text file.

DevOps Relevance

- Scripts parse logs with grep and awk.
- Config files are updated automatically using sed.
- cut and awk are used in monitoring scripts to extract CPU, memory, or disk usage.

Exercise

Combine everything learned into a script report.sh that:

1. Prints the system date and time.
2. Displays the current user.
3. Prints the number of files in the current directory.
4. Searches for "error" in /var/log/syslog.
5. Prints a custom greeting with your name.