**Complete Bash Shell Scripting with Real Industry Use Cases**

**What is Shell & Shell Scripting?**

A **Shell** is a command-line interpreter that provides a user interface to interact with the operating system. It interprets user commands, executes them, and displays the results. The shell acts as an intermediary between the user and the operating system's kernel.

In Linux/Unix, several shell types exist, such as:

* **Bash (Bourne Again Shell):** Default for many Linux distributions.
* **Sh (Bourne Shell):** Simple and older shell.
* **Zsh (Z Shell):** Advanced shell with additional features.
* **Ksh (Korn Shell):** Combines features of earlier shells.
* **Csh (C Shell):** Syntax resembles the C programming language.

**What is Shell Scripting?**

**Shell scripting** involves writing a series of commands in a text file, which the shell executes sequentially. This helps automate repetitive tasks, simplify complex commands, and manage systems efficiently.

A shell script is simply a text file with a .sh extension that contains shell commands.

**Structure of a Shell Script**

1. **Shebang Line:** Indicates the shell interpreter to be used.

bash

**Code**

#!/bin/bash

1. **Commands:** A sequence of shell commands and logic.

**Example Shell Script**

**Script: Greet the User**

bash

**Code**

#!/bin/bash

# Greet the user

echo "Enter your name:"

read name

echo "Hello, $name! Welcome to Shell Scripting."

**Steps to Run the Script:**

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

bash

**code**

chmod +x greet.sh

1. Execute the script:

bash

Copy code

./greet.sh

**Common Shell Commands**

| **Command** | **Description** |
| --- | --- |
| ls | Lists files and directories. |
| pwd | Prints the current working directory. |
| cd | Changes the directory. |
| echo | Displays text or variables. |
| cat | Displays the content of a file. |
| chmod | Changes file permissions. |
| grep | Searches for patterns in files. |
| awk | Processes and analyzes text files. |
| sed | Edits streams of text. |
| ifconfig | Displays network configuration (Linux). |

**Example Script with Logic**

**Script: Check If a File Exists**

bash

**code**

#!/bin/bash

# Check if a file exists

echo "Enter the filename to check:"

read filename

if [ -f "$filename" ]; then

echo "The file '$filename' exists."

else

echo "The file '$filename' does not exist."

fi

**Explanation:**

* [ -f "$filename" ] checks if the given file exists and is a regular file.
* Conditional statements (if, else) control the flow.

**Advantages of Shell Scripting**

1. Automates repetitive tasks.
2. Reduces manual errors.
3. Provides quick prototyping and system administration capabilities.
4. Portable across Unix/Linux systems.

Would you like a script for a specific use case?

* **Operating System GUI & CLI**
* **Introduction to Bash Shell**

1. Operating System GUI and CLI

Linux provides two primary types of user interfaces for interacting with the operating system: **Graphical User Interface (GUI)** and **Command-Line Interface (CLI)**.

**1. Graphical User Interface (GUI)**

**GUI** is a user-friendly interface that uses graphical elements like windows, icons, and buttons. It is designed for users who prefer interacting visually rather than typing commands.

**Examples of Linux GUIs**

1. **GNOME:** A modern desktop environment used by Ubuntu.
2. **KDE Plasma:** A feature-rich and customizable environment.
3. **Xfce:** A lightweight GUI for low-resource systems.
4. **LXDE:** Another lightweight desktop environment.

**Example of GUI Actions**

* **File Management:** Open "File Manager" to browse, copy, or delete files.
* **Network Settings:** Use "Settings" or "Control Panel" to configure Wi-Fi or Ethernet.
* **Web Browsing:** Launch Firefox or Chrome to browse the web.

**2. Command-Line Interface (CLI)**

**CLI** is a text-based interface where users type commands to interact with the system. It is powerful, efficient, and widely used by system administrators and developers.

**Features**

* Lightweight and faster than GUI.
* Allows automation through scripting.
* Provides access to low-level system functionalities.

**Examples of CLI Commands**

| **Command** | **Description** |
| --- | --- |
| ls | Lists files and directories. |
| pwd | Prints the current working directory. |
| cd | Changes the current directory. |
| cp source dest | Copies files or directories. |
| rm file | Removes a file. |
| mkdir dir | Creates a new directory. |
| ping example.com | Checks network connectivity. |

**Example of CLI Usage**

bash

**Code**

# Check system information

uname -a

# Update system packages

sudo apt update && sudo apt upgrade

# Search for a specific text in files

grep "search\_text" filename

**Comparison: GUI vs CLI**

| **Aspect** | **GUI** | **CLI** |
| --- | --- | --- |
| **Ease of Use** | User-friendly and intuitive. | Requires familiarity with commands. |
| **Performance** | Slower due to graphical overhead. | Faster, especially for repetitive tasks. |
| **Customization** | Limited to GUI tools. | Highly customizable with scripts. |
| **Automation** | Limited. | Highly efficient through scripting. |
| **Resource Usage** | Resource-intensive. | Lightweight. |

**Examples of Both GUI and CLI Usage**

**Scenario: Creating a File**

1. **GUI:**
   * Open "File Manager".
   * Right-click in a folder, select **New Document → Text Document**.
   * Name the file and save it.
2. **CLI:**

bash

**Code**

# Create a file named 'example.txt'

touch example.txt

**Scenario: Installing a Package**

1. **GUI:**
   * Open "Software Manager" or "App Store".
   * Search for the package (e.g., VLC Media Player).
   * Click "Install".
2. **CLI:**

bash

**Code**

# Install VLC using apt

sudo apt install vlc

**Conclusion**

* Use **GUI** for tasks that require visualization or simplicity, like media playback or file browsing.
* Use **CLI** for system administration, scripting, or tasks that demand efficiency and automation.

Would you like guidance on a specific GUI or CLI feature in Linux?

4o

2. Introduction to Bash Shell

**Bash** (Bourne Again SHell) is a command-line interpreter and scripting language commonly used in Linux/Unix systems. It is an enhanced version of the original Bourne Shell (sh) and includes additional features, such as command-line editing, job control, and improved scripting capabilities.

Bash is the default shell for many Linux distributions and is powerful for both interactive use and script automation.

**Features of Bash**

1. **Command Execution:** Executes Linux commands like ls, pwd, and mkdir.
2. **Scripting:** Automates tasks using scripts.
3. **Variables:** Supports user-defined and environment variables.
4. **Control Structures:** Provides conditional statements (if, case) and loops (for, while).
5. **History Management:** Tracks and reuses previous commands.
6. **Globbing:** Matches file patterns using wildcards like \* and ?.

**Example: Interactive Bash Usage**

1. Open a terminal.
2. Run basic commands:

bash

**Code**

pwd # Print the current working directory

ls # List files and directories

mkdir test\_dir # Create a new directory named test\_dir

cd test\_dir # Change to test\_dir directory

**Example: Bash Shell Script**

**Script: Basic Greeting Program**

1. **Create the Script File:**

bash

**Code**

nano greet.sh

1. **Write the Script:**

bash

**Code**

#!/bin/bash

# Greeting Script

echo "What is your name?"

read name

echo "Hello, $name! Welcome to Bash Shell scripting."

1. **Make the Script Executable:**

bash

**Code**

chmod +x greet.sh

1. **Run the Script:**

bash

**Code**

./greet.sh

1. **Sample Output:**

vbnet

Copy code

What is your name?

John

Hello, John! Welcome to Bash Shell scripting.

**Key Components of Bash**

**1. Variables**

* **Syntax:** variable\_name=value (No spaces around =).
* **Example:**

bash

**Code**

name="Alice"

echo "Hello, $name!"

**2. Conditional Statements**

* **Example:**

bash

**Code**

#!/bin/bash

echo "Enter a number:"

read num

if [ $num -gt 10 ]; then

echo "The number is greater than 10."

else

echo "The number is 10 or less."

fi

**3. Loops**

* **Example:**

bash

**Code**

#!/bin/bash

for i in {1..5}

do

echo "Number: $i"

done

**4. Functions**

* **Example:**

bash

**Code**

#!/bin/bash

greet() {

echo "Hello, $1!"

}

greet "Alice"

**Advantages of Bash**

1. **Cross-Platform:** Works on most Unix/Linux systems.
2. **Automation:** Simplifies repetitive tasks.
3. **Efficiency:** Lightweight and fast for small tasks.
4. **Integration:** Works seamlessly with Linux utilities.

**Practical Example: System Monitoring Script**

**Script: Display System Information**

bash

**Code**

#!/bin/bash

# Display system information

echo "System Information"

echo "------------------"

echo "Hostname: $(hostname)"

echo "Uptime: $(uptime -p)"

echo "Logged in Users:"

who

1. Save it as sysinfo.sh.
2. Run the script:

bash

**Code**

./sysinfo.sh

1. Sample Output:

yaml

**Code**

System Information

------------------

Hostname: mycomputer

Uptime: up 2 hours, 15 minutes

Logged in Users:

user1 tty7 2024-12-11 08:00

Would you like more advanced examples of Bash scripting?

**Creating a Script File and Building a Command**

* **Variables & User-defined Variables**
* **Status Code**
* **Executable Program**
* **Path**
* **Hashbang or Shebang**
* **Script File**
* **Bashrc File**
* **Arguments**

**Variables & User-defined Variables**

A **variable** is a named memory location used to store data. In Linux, variables are primarily used in shell scripting to manage and manipulate data. Variables can hold strings, numbers, filenames, or commands.

**Types of Variables in Linux**

1. **System Variables:**
   * Predefined variables provided by the shell.
   * Typically in uppercase (e.g., $HOME, $PATH, $USER).
   * Example:

bash

Code

echo "Your home directory is: $HOME"

echo "Your username is: $USER"

1. **User-Defined Variables:**
   * Variables created and assigned by the user.
   * Names are typically in lowercase or camelCase.
   * Example:

bash

Code

name="Alice"

echo "Hello, $name!"

**Rules for Variable Naming**

1. Must start with a letter or underscore (\_).
2. Can contain letters, numbers, and underscores.
3. No spaces around the = sign.
4. Avoid using special characters.
5. Reserved words (e.g., if, while) cannot be used as variable names.

**User-Defined Variables**

**Creating and Using Variables**

1. Assign a value:

bash

Code

variable\_name=value

1. Access the value:

bash

Code

echo $variable\_name

**Example 1: Simple Variable Usage**

bash

Code

#!/bin/bash

# Assigning a value to a variable

name="John"

age=25

# Using the variables

echo "My name is $name and I am $age years old."

**Output:**

csharp

Code

My name is John and I am 25 years old.

**Changing Variable Values**

* Variables are dynamic; you can reassign values.

bash

Code

#!/bin/bash

greeting="Hello"

echo $greeting

# Change the value of the variable

greeting="Hi"

echo $greeting

**Using Variables in Operations**

**Example 2: Arithmetic Operations**

bash

Code

#!/bin/bash

# Assigning numbers to variables

num1=10

num2=20

# Performing arithmetic

sum=$((num1 + num2))

echo "The sum of $num1 and $num2 is $sum"

**Output:**

python

Code

The sum of 10 and 20 is 30

**Read User Input into Variables**

The read command allows input from the user.

**Example 3: Interactive Script**

bash

Code

#!/bin/bash

# Prompting user for input

echo "Enter your name:"

read name

echo "Welcome, $name!"

**Output:**

mathematica

Code

Enter your name:

Alice

Welcome, Alice!

**Exporting Variables**

Variables defined in a shell script are not available in subshells unless exported.

**Example 4: Export Variable**

bash

Code

#!/bin/bash

# Define and export a variable

export greeting="Hello, World!"

# Call another script or subshell

bash -c 'echo $greeting'

**Output:**

Code

Hello, World!

**Example: Combining User-Defined Variables and Commands**

bash

Code

#!/bin/bash

# Assign a variable from a command

current\_date=$(date)

user=$(whoami)

# Display the information

echo "Hello, $user! Today's date is $current\_date."

**Output:**

vbnet

Code

Hello, user1! Today's date is Wed Dec 11 10:00:00 UTC 2024.

**Unsetting Variables**

Use the unset command to remove a variable.

bash

Code

#!/bin/bash

var="Temporary Value"

echo $var

unset var

echo $var # Output will be empty

**Summary**

* Variables in Linux are essential for managing data in shell scripts.
* **System Variables** are predefined, while **User-Defined Variables** are created by the user.
* Variables can store strings, numbers, or command outputs.
* Use export to make variables available in subshells, and unset to remove them.

Would you like more advanced examples or further clarification?

**Status code**

In Linux, a **status code** (or exit code) is a numerical value returned by a command, script, or process to indicate the result of its execution. This code helps determine whether a command succeeded, failed, or encountered specific errors.

* **0:** Indicates success (no error).
* **Non-zero:** Indicates failure or specific error types.

Status codes are essential for automating tasks, as they allow scripts and processes to respond dynamically based on the success or failure of commands.

**How to Check a Status Code**

The special variable $? holds the exit status of the last executed command.

**Example 1: Checking Status Code**

bash

Code

# Run a successful command

ls

echo $? # Output: 0 (Success)

# Run a failing command

ls nonexistent\_file

echo $? # Output: Non-zero (Error)

**Common Status Codes**

| **Code** | **Description** |
| --- | --- |
| **0** | Success: Command executed without errors. |
| **1** | General error: Catch-all for general issues. |
| **2** | Misuse of shell built-ins. |
| **126** | Command invoked but cannot execute. |
| **127** | Command not found. |
| **128** | Invalid exit argument. |
| **130** | Script terminated by Ctrl+C (signal 2). |

**Examples of Status Codes**

**Example 2: Using $? in a Script**

bash

Code

#!/bin/bash

echo "Checking if a file exists..."

ls /path/to/file > /dev/null 2>&1

if [ $? -eq 0 ]; then

echo "File exists."

else

echo "File does not exist."

fi

**Example 3: Using Status Codes for Conditional Execution**

You can chain commands using && (execute if the previous command succeeds) and || (execute if the previous command fails).

bash

Code

# Success case

mkdir new\_directory && echo "Directory created successfully."

# Failure case

cd nonexistent\_directory || echo "Failed to change directory."

**Custom Status Codes in Scripts**

You can define your own status codes in scripts using the exit command.

**Example 4: Custom Exit Codes**

bash

Code

#!/bin/bash

# Custom status code

echo "Processing..."

if [ -d "/nonexistent\_directory" ]; then

echo "Directory exists."

exit 0 # Success

else

echo "Directory does not exist."

exit 1 # Failure

fi

**Running the script:**

bash

Code

./custom\_exit.sh

echo $? # Displays 0 or 1 based on the script's exit code

**Practical Usage of Status Codes**

1. **Error Handling:** Automate responses to errors in scripts.
2. **Debugging:** Identify where and why a command failed.
3. **Conditional Execution:** Build complex workflows based on success/failure of commands.

**Executable Program**

An **executable program** in Linux is a file that contains compiled code or script instructions that the operating system can run. When executed, the file performs a specific task, such as running a software application, processing data, or automating a system operation.

Executable files can be:

1. **Binary Executables:** Compiled from programming languages like C or C++ into machine code (e.g., /bin/ls, /usr/bin/vim).
2. **Scripts:** Text files containing commands interpreted by a shell or language interpreter (e.g., Bash scripts, Python scripts).

**Characteristics of Executable Files**

1. **File Permissions:**
   * Executable files must have the **execute** (x) permission set.
   * Example of file permissions for an executable file:

sql

Copy code

-rwxr-xr-x 1 user group 4096 Dec 11 10:00 my\_program

1. **File Types:**
   * Binary executable: Identified by file headers (e.g., ELF format).
   * Script: Starts with a **shebang** (#!) that specifies the interpreter.
2. **Location:**
   * System-wide executables: Found in directories like /bin, /usr/bin, /sbin.
   * User-created executables: Typically stored in custom directories.

**Examples of Executable Programs**

**Binary Executables**

* Commands like ls, cp, cat, and vim are compiled programs located in /bin or /usr/bin.

**Script Executables**

* Bash script:

bash

Copy code

#!/bin/bash

echo "This is a script."

* Python script:

python

Copy code

#!/usr/bin/python3

print("This is a Python script.")

**How to Identify Executable Files**

1. **Using ls Command:** Files with the x permission are executable:

bash

Copy code

ls -l

Example output:

sql

Copy code

-rwxr-xr-x 1 user group 1234 Dec 11 10:00 my\_program

1. **Using file Command:** Identify the type of a file:

bash

Copy code

file /bin/ls

Example output:

arduino

Copy code

/bin/ls: ELF 64-bit LSB executable, x86-64

**How to Make a File Executable**

1. **Create the File:**

bash

Copy code

nano my\_script.sh

1. **Write the Script:**

bash

Copy code

#!/bin/bash

echo "Hello, world!"

1. **Set Execute Permission:**

bash

Copy code

chmod +x my\_script.sh

1. **Run the Script:**

bash

Copy code

./my\_script.sh

**Running an Executable Program**

1. **System Commands:** Simply type the command (e.g., ls, cat, vim).
2. **Custom Executables:** Specify the relative or absolute path:

bash

Copy code

./my\_program

/path/to/my\_script.sh

1. **Using PATH Environment Variable:** Add custom executables to the PATH variable to run them directly:

bash

Copy code

export PATH=$PATH:/path/to/directory

**Practical Example: Custom Executable Program**

**Create a Bash Script**

bash

Copy code

#!/bin/bash

echo "This is a custom executable program."

**Save and Make It Executable**

bash

Copy code

nano my\_program.sh

chmod +x my\_program.sh

**Run It**

bash

Copy code

./my\_program.sh

**Output:**

vbnet

Copy code

This is a custom executable program.

**Summary**

* An executable program in Linux is any file capable of being run as a command.
* **Binary executables** are compiled machine code, while **scripts** are interpreted by the shell or other interpreters.
* Files must have execute permissions (chmod +x) to be runnable.

**Iteration in Shell Scripting**

* **Useradd & ID Command**
* **Iteration & Loop**
* **For Loop Syntax**
* **Passwd File Manipulation**

**If/Else Conditions & User Backup Scripting**

* **If-Else Conditions**
* **Test Command**
* **Conditional Operators**
* **I/O Redirection**
* **Nested Conditions**
* **User Management Commands (Useradd, User Lock)**
* **Regex, Grep & Cut Command**
* **Home Directory Management**
* **Backup using Tar Command**

**Executing Scripts Over Network**

* **Log Analysis**
* **Launching AWS EC2 Instance**
* **WinSCP Tool**
* **Configuring Web Server (HTTPD)**
* **Access Logs & Error Logs**
* **File Operations (WC, CHMOD, SUDO, AWK, SORT, UNIQ, TAIL, SCP)**

**Advanced Scripting Techniques**

* **Regular Expressions**
* **AWK & GREP Command**
* **SED Command**
* **Creating Options in Scripts**
* **Using GETOPTS & SET Command**
* **Pre-created Variables & Functions**

**Linux OS Fundamentals**

* **Shell Basics**
* **Cisco Switches**
* **GUI vs. CLI**
* **Introduction to Bash Shell**
* **Script File Creation**
* **Test Keyword**
* **Logical Operators**

**Advanced Shell Scripting Concepts**

* **User-defined Variables**
* **Arrays**
* **File Manipulation**
* **Looping Constructs**

**Process & Subprocess Management**

* **Job Control**
* **Daemon Processes**
* **Signal Handling**
* **Looping & Flow Control**

**Advanced Commands and Techniques**

* **Shopt Command**
* **Tree Command**
* **Export Keyword**
* **Parallel Execution**
* **SSH & SCP Commands**
* **AWK Command for Text Processing**