**Shell scripting**

**Linux Architecture**

1. **Hardware** – Physical components (CPU, RAM, Disk, I/O devices).
2. **Kernel** – Core part of OS, manages hardware resources and provides services to processes.
3. **Shell** – Interface between user and kernel (takes input, passes to kernel, displays output).
4. **Application/User** – Programs and users that interact with the OS through the shell.

**What is Shell?**

* A **user interface** that allows access to operating system services.
* Acts as a **bridge between user and kernel**.

**Types of Shell**

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

* Provides icons, menus, windows for interaction.
* Examples:
  + **Windows** → File Explorer
  + **macOS** → Finder
  + **Linux** → GNOME, KDE
  + Command: echo $XDG\_CURRENT\_DESKTOP

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

* Text-based interaction.
* User types commands to communicate with OS.
* Example: Linux terminal, PowerShell in Windows.

**Types of Linux Shell**

1. **sh (Bourne Shell)**
   * Original Unix shell.
   * Lightweight but limited features.
2. **bash (Bourne Again Shell)**
   * Most popular and default on Linux.
   * Supports command history, scripting, job control.
3. **csh (C Shell)**
   * C-like syntax.
   * Useful for programmers familiar with C.
4. **tcsh (TENEX C Shell)**
   * Enhanced version of csh with more features.
5. **ksh (Korn Shell)**
   * Combines features of Bourne and C shell.
   * Supports advanced scripting, functions, arrays.
6. **zsh (Z Shell)**
   * Modern, powerful shell.
   * Features: auto-completion, themes, plugins (oh-my-zsh).

**Commands to Know About Shell**

* echo $0 → Shows current shell in use.
* cat /etc/shells → Lists all shells available on the system.

**What is Shell Scripting?**

* A **file containing multiple shell commands** executed sequentially.
* Extension: .sh
* Purpose: automation of tasks.

**Steps to Create & Run a Shell Script**

1. **Create file**

touch filename.sh

1. **Add commands inside file**

nano filename.sh

1. **Give execute permission**

chmod 700 filename.sh

1. **Execute file**
   * From same location:

./filename.sh

* + From different location (absolute path):

/path/to/filename.sh

**Example Shell Script**

#!/bin/bash

# My first script

echo "Hello, $USER"

echo "Today is $(date)"

echo "Your current directory is $(pwd)"

Run:

./first.sh

Shell Scripting - A Variable Guide

This document provides a detailed overview of variables in shell scripting, based on the provided source, "Shell Scripting: A Variable Guide." It summarises key concepts, definitions, and practical applications, including important syntax rules and examples.

Main Themes and Most Important Ideas/Facts:

**1. What is a Variable?**

• A variable is fundamentally a "name that stores a value." These values can be diverse, including "string, number, filename, command output, etc."

• Crucially, shell scripting does not have explicit data types; "everything is treated as a string by default."

**2. Defining and Accessing Variables:**

• **Definition:** Variables are defined by assigning a value to a name, e.g., name="Sharan" or age=22.

• **Crucial Syntax Rule:** There must be "no spaces around =." This is highlighted as "very important."

• **Accessing:** To retrieve the value stored in a variable, a dollar sign ($) must precede the variable name, e.g., echo $name.

**3. System Variables:**

• The shell predefines a set of variables, known as "System Variables," which provide access to common system information.

• **Key Examples:**

    ◦ $USER: Current username (e.g., echo "Logged in as: $USER")

    ◦ $HOME: Home directory path

    ◦ $PWD: Current working directory

    ◦ $SHELL: Default shell

    ◦ $0: Name of the script itself

    ◦ $$: Process ID of the script

    ◦ $#: Number of arguments passed to the script

    ◦ $@: All arguments passed to the script

**4. Command Substitution:**

• Shell scripting allows the output of a command to be stored directly into a variable.

• This is achieved using the $(command) syntax.

• **Examples:**

    ◦ today=$(date) stores the current date.

    ◦ files=$(ls) stores a list of files.

    ◦ "Today is $today" and "Files: $files" demonstrate accessing these command outputs.

**5. Reading User Input:**

• The read command is used to capture input from the user and store it in a variable.

• **Example:**

    ◦ echo "Enter your name:" prompts the user.

    ◦ read username stores the user's input into the username variable.

    ◦ echo "Welcome, $username" then uses this input.

**6. Constant Variables (Read-only):**

• Variables can be made immutable (read-only) using the readonly command.

• **Example:**

    ◦ PI=3.14 defines the variable.

    ◦ readonly PI makes it read-only.

    ◦ Attempting to reassign (PI=22) will result in an "error: cannot change."

**7. Unsetting Variables:**

• Variables can be removed from memory using the unset command.

• **Example:**

    ◦ name="Linux" defines the variable.

    ◦ unset name removes it.

    ◦ echo $name will then produce "(blank output)."

**8. Practical Scripting Example:**

The provided example script consolidates these concepts:

#!/bin/bash

# Defining variables

name="Bro"

city="Mangalore"

echo "Hello $name, welcome to $city!"

# System variables

echo "Your current shell is $SHELL"

echo "You are logged in as $USER"

echo "Your current directory is $PWD"

# User input

echo "Enter your favorite language:"

read lang

echo "Nice! You like $lang"