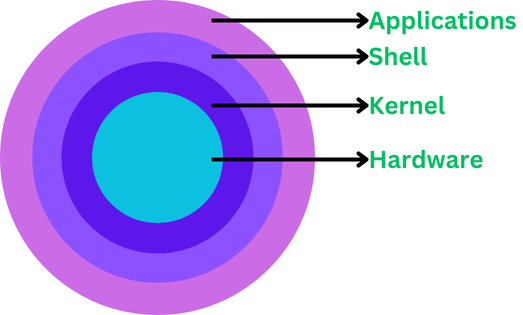
**Shell Scripting**

**Linux architecture**



**Types of User Interfaces**

In the context of using shells and interacting with operating systems, GUI and CLI are two different types of user interfaces.

**1) Graphical user interface (GUI)**

A graphical user interface allows users to interact with the system through graphical elements like windows, icons, menus, and buttons.

**Examples**: Desktop environments like GNOME, KDE, macOS, and Windows, as well as specific applications like web browsers and graphical file managers.

**2) Command line interface (CLI)**

A command-line interface allows users to interact with the system by typing text commands into a terminal or console.

**Examples**: Shells like bash, zsh, ksh, and dash. Terminals and consoles used in Linux, macOS, and Windows (Command Prompt, PowerShell, Windows Subsystem for Linux).

**Shell?**

Shell is a command line interface between kernel and Application/user.

**Types of Shell**

SH - Bourne shell

BASH - Bourne shell

KSH - Korn shell

CSH - C shell

ZSH - Z shell

DASH - Debian Almquist Shell

**Shell Scripting**

A shell script is an executable file containing multiple shell commands that are executed sequentially.

**commands**

→ To see which shell you are using

**echo $0**

→ To see the available shells

**cat /etc/shells**

**Steps to create Shell script**

1) Create a .sh file.

ex:- **touch demo.sh**

2) Give executable permission to the file.

ex:- **chmod a+x demo.sh**

3) Edit the file and write the script.

ex:- **nano demo.sh**

**#! /bin/bash** → selecting the bash terminal (#! is called as shebang operator)

**echo “Hi”** → prints Hi message

4) Run your file using the absolute path

ex:- /home/username/demo.sh **or** ./demo.txt (./ represents current working directory)

**Taking Input from the user**

→ read variable-name

**ex:- read age**

(This will take the input from the user and store it in a variable called age. ex: age = 20)

**Printing the output**

→ echo “write a statement here”

**ex:- echo “Hi Sachin, How are you?”**

**Declaring the variables**

**→ for strings**

variable\_name = “statement”

**ex:- name = “Sachin”**

**→ for numbers**

variable\_name = number

**ex:- age = 20**

**Operators**

**1. Arithmetic Operators**

Arithmetic operators are special symbols that perform mathematical operations such as addition, subtraction, multiplication, etc.

* + - + : Addition
    - - : Subtraction
    - \* : Multiplication
    - / : Division
    - % : Modulus

**ex:-**

**#!/bin/bash**

**a=10**

**b=20**

**echo $((a + b)) # Output: 30**

**echo $((a - b)) # Output: -10**

**echo $((a \* b)) # Output: 200**

**echo $((b / a)) # Output: 2**

**echo $((b % a)) # Output: 0**

**2. Relational Operators**

These operators check the relationship between two operands and returns the true or false.

* + - -eq : Equal to
    - -ne : Not equal to
    - -gt : Greater than
    - -lt : Less than
    - -ge : Greater than or equal to
    - -le : Less than or equal to

**ex:-**

**#!/bin/bash**

**a=10**

**b=20**

**if [ $a -eq $b ]; then echo "a is equal to b"; fi**

**if [ $a -ne $b ]; then echo "a is not equal to b"; fi**

**if [ $a -gt $b ]; then echo "a is greater than b"; fi**

**if [ $a -lt $b ]; then echo "a is less than b"; fi**

**if [ $a -ge $b ]; then echo "a is greater than or equal to b"; fi**

**if [ $a -le $b ]; then echo "a is less than or equal to b"; fi**

**3. Logical Operators**

These operators compare two values and return true or false.

* + - ! : NOT
    - -o : OR
    - -a : AND

**ex:-**

**#!/bin/bash**

**a=10**

**b=20**

**if [ $a -lt 15 -a $b -gt 15 ]; then echo "Both conditions are true"; fi**

**if [ $a -lt 15 -o $b -lt 15 ]; then echo "At least one condition is true"; fi**

**if [ ! $a -ge 15 ]; then echo "Condition is true"; fi**

**4. String Operators**

These operators are used to compare strings.

* + - = : Equal to
    - != : Not equal to
    - -z : String is null, that is, has zero length
    - -n : String is not null

**ex:-**

**#!/bin/bash**

**str1="Hello"**

**str2="World"**

**if [ "$str1" = "$str2" ]; then echo "Strings are equal"; else echo "Strings are not equal"; fi**

**if [ "$str1" != "$str2" ]; then echo "Strings are not equal"; fi**

**if [ -z "$str1" ]; then echo "String is empty"; else echo "String is not empty"; fi**

**if [ -n "$str1" ]; then echo "String is not empty"; fi**

**5. File Test Operators**

These operators are used to test various properties of files.

* + - -e : File exists
    - -r : File is readable
    - -w : File is writable
    - -x : File is executable
    - -f : File is a regular file
    - -d : Directory exists
    - -s : File is not empty

**ex:-**

**#!/bin/bash**

**file="example.txt"**

**if [ -e "$file" ]; then echo "File exists"; fi**

**if [ -r "$file" ]; then echo "File is readable"; fi**

**if [ -w "$file" ]; then echo "File is writable"; fi**

**if [ -x "$file" ]; then echo "File is executable"; fi**

**if [ -f "$file" ]; then echo "File is a regular file"; fi**

**if [ -d "$file" ]; then echo "It is a directory"; fi**

**if [ -s "$file" ]; then echo "File is not empty"; fi**

**Conditional statements**

**1. if Statement**

The if statement executes a block of code if a specified condition is true.

**ex:-**

**#!/bin/bash**

**a=10**

**if [ $a -gt 5 ]; then**

**echo "a is greater than 5"**

**fi**

**2. if-else Statement**

The if-else statement executes one block of code if a specified condition is true and another block of code if the condition is false.

**ex:-**

**#!/bin/bash**

**a=10**

**if [ $a -gt 5 ]; then**

**echo "a is greater than 5"**

**else**

**echo "a is not greater than 5"**

**fi**

**3. if-elif-else Statement**

The if-elif-else statement executes different blocks of code based on multiple conditions.

**ex:-**

**#!/bin/bash**

**a=10**

**if [ $a -gt 15 ]; then**

**echo "a is greater than 15"**

**elif [ $a -gt 5 ]; then**

**echo "a is greater than 5 but less than or equal to 15"**

**else**

**echo "a is 5 or less"**

**fi**

**4. Nested if Statements**

You can also nest if statements within each other to perform more complex checks.

**ex:-**

**#!/bin/bash**

**a=10**

**b=20**

**if [ $a -gt 5 ]; then**

**if [ $b -gt 15 ]; then**

**echo "a is greater than 5 and b is greater than 15"**

**else**

**echo "a is greater than 5 but b is not greater than 15"**

**fi**

**else**

**echo "a is not greater than 5"**

**fi**

**Logical Operators in Conditional Statements**

Logical operators are used to combine multiple conditions in a single if statement.

* + - && : Logical AND
    - || : Logical OR
    - ! : Logical NOT

**ex:-**

**#!/bin/bash**

**a=10**

**b=20**

**if [ $a -gt 5 ] && [ $b -gt 15 ]; then**

**echo "Both conditions are true"**

**fi**

**if [ $a -gt 15 ] || [ $b -gt 15 ]; then**

**echo "At least one condition is true"**

**fi**

**Loops**

Loops are used to repeatedly execute a block of commands as long as a certain condition is true.

**1. for loop**

The for loop iterates over a list of items and executes the specified commands for each item.

**i) Basic for Loop**

**#!/bin/bash**

**for i in 1 2 3 4 5; do**

**echo "Number: $i"**

**done**

**ii) for Loop with a Range**

**#!/bin/bash**

**for i in {1..5}; do**

**echo "Number: $i"**

**done**

**iii) for Loop with Step Value**

**#!/bin/bash**

**for i in {1..10..2}; do**

**echo "Number: $i"**

**done**

**iv) for Loop Iterating Over Files**

**#!/bin/bash**

**for file in \*.txt; do**

**echo "File: $file"**

**done**

**2. while Loop**

The while loop executes the specified commands as long as the condition is true.

**#!/bin/bash**

**counter=1**

**while [ $counter -le 5 ]; do**

**echo "Counter: $counter"**

**counter=$((counter+1))**

**done**

**3. until Loop**

The until loop executes the specified commands until the condition becomes true.

**#! /bin/bash**

**counter=1**

**until [ $counter -gt 5 ]; do**

**echo "Counter: $counter"**

**counter=$((counter+1))**

**done**

**Functions**

Functions allow you to encapsulate a set of commands, making your scripts reusable. There are two ways to create functions

**1) first way**

**function\_name () {**

**# commands**

**}**

**2) second way**

**function function\_name {**

**# commands**

**}**

**Example:-**

**#!/bin/bash**

**# Define a function**

**greet() {**

**echo "Hello, $1"**

**}**

**# Call the function**

**greet "World"**

**Function with multiple arguments:**

**#!/bin/bash**

**add\_numbers() {**

**local sum=$(($1 + $2))**

**echo "Sum: $sum"**

**}**

**add\_numbers 5 10**

**Function with a return value:**

**#!/bin/bash**

**multiply\_numbers() {**

**local product=$(($1 \* $2))**

**echo $product**

**}**

**result=$(multiply\_numbers 4 5)**

**echo "Product: $result"**

**To check the user existence**

**#! /bin/bash**

**if id username; then**

**echo “user exists”**

**else**

**echo “user doesn't exist”**

**fi**

**To read a file line by line using a while loop**

**while read line; do**

**echo "$line"**

**done < "$filename"**