**1. What is Shell Scripting?**

Shell scripting is the process of writing a file with Linux/Unix commands that the shell can execute in order.

**Use Cases:**

* Automating backups
* Starting/stopping services
* User creation
* Deployment scripts

**Example:**

#!/bin/bash

echo "System update started"

sudo apt update && sudo apt upgrade -y

### ****2. What is a Variable?****

A variable in shell scripting is used to **store data** (like text, numbers, or paths) so you can reuse it in your script.

### ****Syntax:****

### variable\_name=value

### Types of Variables in Shell Scripting:

|  |  |  |
| --- | --- | --- |
| 1️. **User-defined** | * Created by the user in a script | name="Shan" |

|  |  |  |
| --- | --- | --- |
| 2️. **Environment** | * Predefined by the system, available globally | $HOME, $USER, $PATH |

|  |  |  |
| --- | --- | --- |
| 3️. **Positional** | * Used to access script arguments ($1, $2, etc.) | echo "First arg: $1" |

|  |  |  |
| --- | --- | --- |
| 4️. **Special** | * Provide info about execution ($?, $$, $#) | echo "PID is $$" |

## **1. Environment Variables**

These are predefined by the system and available to all processes. You can also create your own.

| **Variable** | **Description** | **Example Output (depends on system)** |
| --- | --- | --- |
| $USER | Current logged-in username | shanthakumar |
| $HOME | Path to the home directory | /home/shanthakumar |
| $PATH | System path for executable files | /usr/bin:/bin:/usr/sbin:... |
| $SHELL | Current shell in use | /bin/bash |
| $HOSTNAME | Hostname of the system | devops-machine |
| $PWD | Current working directory | /home/user/scripts |

## **2. Positional Parameters**

These are used to access command-line arguments passed to a script.

| **Variable** | **Description** | **Example** |
| --- | --- | --- |
| $0 | Script name | ./myscript.sh |
| $1 | First argument | arg1 |
| $2 | Second argument | arg2 |
| $# | Total number of arguments | 2 |
| $\* | All arguments as a single string | "arg1 arg2" |
| $@ | All arguments as individual items | "arg1" "arg2" |

EX1: bash args.sh Hello World

### 🔸 ****$0**** – Script name

echo "Script name: $0"

**Output:**

Script name: args.sh

### 🔸 ****$1, $2**** – First and second arguments

echo "First Arg: $1"

echo "Second Arg: $2"

**Output:**

First Arg: Hello

Second Arg: World

### 🔸 ****$#**** – Number of arguments

echo "Total arguments: $#"

**Output:**

Total arguments: 2

### 🔸 ****$@**** – All arguments (individually)

echo "All args using \$@: $@"

**Output:**

All args using $@: Hello World

### 🔸 ****$****\* – All arguments (as a single string)

echo "All args using \$\*: $\*"

**Output:**

All args using $\*: Hello World

## **3. Special Variables**

These provide metadata or special status during execution.

| **Variable** | **Description** | **Example Use** |
| --- | --- | --- |
| $$ | Current script’s process ID (PID) | echo $$ |
| $? | Exit status of last command (0 = success) | ls xyz; echo $? |
| $! | PID of the last background command | sleep 30 & echo $! |

EX1:

### ****$$**** – Current script process ID

echo "Current Script PID: $$"

**Output:**

Current Script PID: 34567

### 🔸 ****$?**** – Exit status of last command (0 = success, non-zero = error)

ls

echo "Exit code: $?"

ls invalid\_dir

echo "Exit code: $?"

**Output:**

Exit code: 0

ls: cannot access 'invalid\_dir': No such file or directory

Exit code: 2

### 🔸 ****$!**** – PID of last background process

sleep 30 &

echo "Background PID: $!"

**Output:**

Background PID: 34600

**3. Taking Input and Showing Output in Shell Scripting**

#### **1.** read **Command**

* Used to take **user input** from the keyboard/commandline.
* Syntax:

read variable\_name

#### **2.** echo **Command**

* Displays output on the screen.
* Syntax:

echo "Message"

Ex1:

#!/bin/bash

echo "What is your name?"

read name # taking input from user

echo "How old are you?"

read age # taking another input

echo "Hello $name, you are $age years old."

**What is an Operator in Shell Scripting?**

An **operator** is a **symbol or keyword** used to perform specific operations on **variables** or **values**, such as:

* Arithmetic calculations
* Comparisons
* Logical decisions
* String handling
* File testing

Operators help you **control the logic and flow** of your shell script.

## ✅ 1. **Arithmetic Operators**

| **Operator** | **Description** |
| --- | --- |
| + | Addition |
| - | Subtraction |
| \* | Multiplication |
| / | Division |
| % | Modulus |

#!/bin/bash

a=20

b=7

echo "Add: $((a + b))"

echo "Subtract: $((a - b))"

echo "Multiply: $((a \* b))"

echo "Divide: $((a / b))"

echo "Modulus: $((a % b))"

## ✅ 2. **Relational Operators** (Integer comparison)

| **Operator** | **Description** |
| --- | --- |
| -eq | Equal to |
| -ne | Not equal to |
| -gt | Greater than |
| -lt | Less than |
| -ge | Greater or equal |
| -le | Less or equal |

#!/bin/bash

a=15

b=20

if [ $a -eq $b ]; then echo "Equal"; fi

if [ $a -ne $b ]; then echo "Not Equal"; fi

if [ $a -gt $b ]; then echo "Greater"; fi

if [ $a -lt $b ]; then echo "Less"; fi

if [ $a -ge $b ]; then echo "Greater or Equal"; fi

if [ $a -le $b ]; then echo "Less or Equal"; fi

## ✅ 3. **Logical Operators**

| **Operator** | **Description** |
| --- | --- |
| && | Logical AND |
| ! | Logical NOT |

#!/bin/bash

a=5

b=10

# AND

if [[ $a -lt 10 && $b -gt 5 ]]; then

echo "AND condition is true"

fi

# OR

if [[ $a -lt 5 || $b -gt 5 ]]; then

echo "OR condition is true"

fi

# NOT

if ! [ $a -gt 10 ]; then

echo "NOT condition is true"

fi

## ✅ 4. **String Operators**

| **Operator** | **Description** |
| --- | --- |
| = | Equal |
| != | Not equal |
| < | Less than (ASCII) |
| > | Greater than (ASCII) |
| -z | String is null (zero length) |
| -n | String is not null (non-zero len) |

#!/bin/bash

str1="apple"

str2="banana"

if [ "$str1" = "$str2" ]; then echo "Equal"; else echo "Not equal"; fi

if [ "$str1" != "$str2" ]; then echo "Not Equal"; fi

if [[ "$str1" < "$str2" ]]; then echo "$str1 is less than $str2"; fi

if [[ "$str1" > "$str2" ]]; then echo "$str1 is greater than $str2"; fi

if [ -z "$str1" ]; then echo "Empty String"; fi

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

## ✅ 5. **File Test Operators**

| **Operator** | **Description** |
| --- | --- |
| -e | File exists |
| -f | Regular file |
| -d | Directory |
| -s | File is not empty |
| -r | Readable |
| -w | Writable |
| -x | Executable |

#!/bin/bash

file="/etc/passwd"

dir="/etc"

[ -e "$file" ] && echo "File exists"

[ -f "$file" ] && echo "Regular file"

[ -d "$dir" ] && echo "It's a directory"

[ -s "$file" ] && echo "File is not empty"

[ -r "$file" ] && echo "File is readable"

[ -w "$file" ] && echo "File is writable"

[ -x "$file" ] && echo "File is executable"

## ✅ 6. **Assignment Operators**

| **Operator** | **Description** |
| --- | --- |
| = | Assign value |
| += | Add and assign |
| -= | Subtract and assign |
| \*= | Multiply and assign |
| /= | Divide and assign |
| %= | Modulus and assign |

#!/bin/bash

a=10

((a += 5)) # a = a + 5

echo "After += : $a"

((a -= 3)) # a = a - 3

echo "After -= : $a"

((a \*= 2)) # a = a \* 2

echo "After \*= : $a"

((a /= 4)) # a = a / 4

echo "After /= : $a"

((a %= 3)) # a = a % 3

echo "After %= : $a"