**Shell Scripting**

**🔰Fundamentals:**

**1. What is a Shell?**

* A shell is a program that takes commands from the user and gives them to the operating system to execute.
* Examples: bash (Bourne Again Shell), zsh, sh, ksh.
* Use: It’s the interface between user and OS.

User (Commands)

Shell (bash/sh/zsh)

- Parses commands

- Executes scripts

Operating System Kernel

- Manages CPU, Memory, I/O

Hardware (CPU, RAM, Disk)

**2. What is Shell Scripting?**

* Shell scripting is writing a sequence of commands in a file (script) that the shell can execute automatically.
* Use in DevOps: Automates repetitive tasks like deployments, monitoring, backups, and server setup.

**3. Creating & Running a Script**

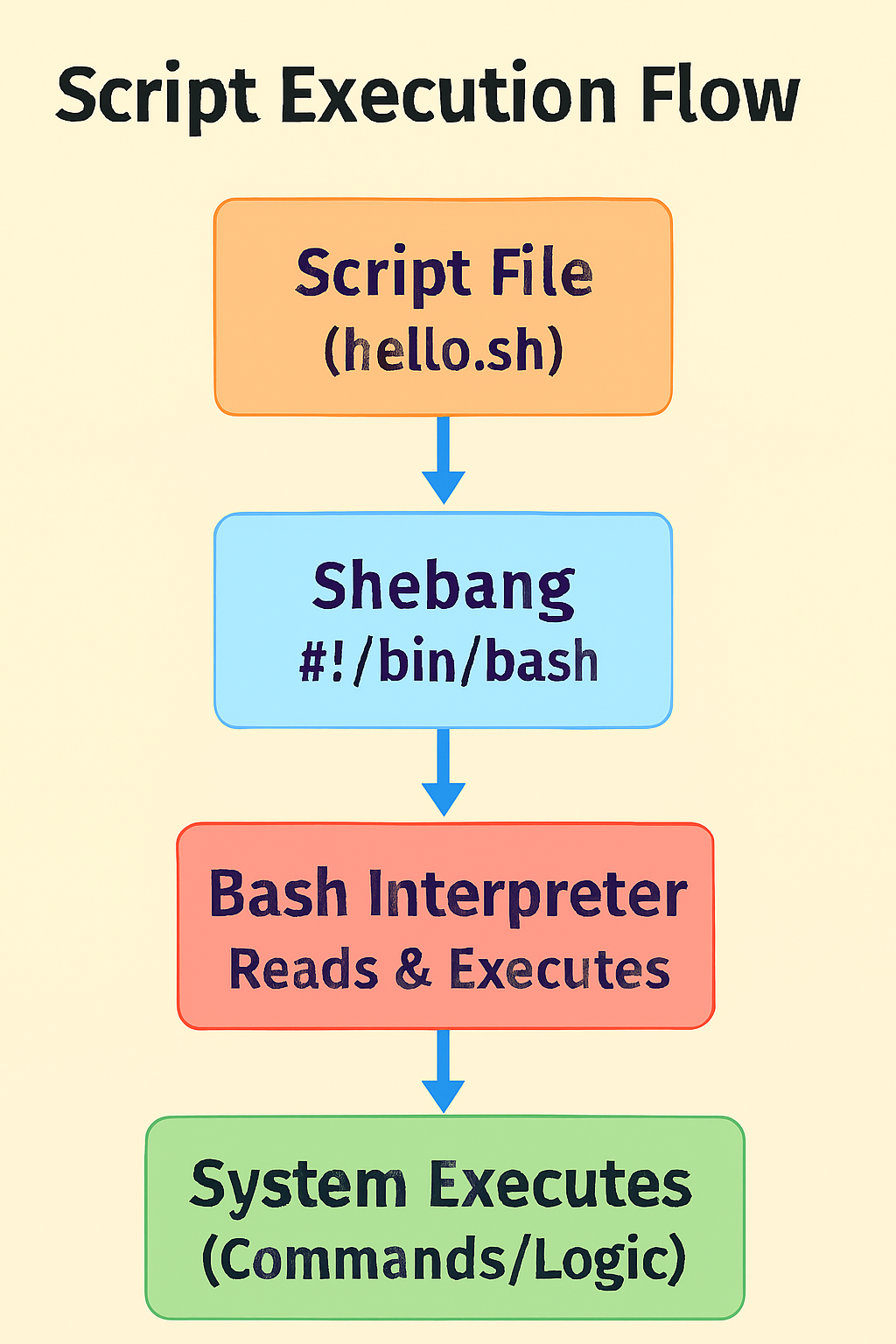
Steps (works on Linux / WSL / Git Bash on Windows):

* Create a new file → nano hello.sh
* Add: #!/bin/bash

echo "Hello, DevOps!"

* #!/bin/bash → Shebang → tells the system to use Bash shell.
* Save and exit.
* Make executable → chmod +x hello.sh
* Run → ./hello.sh

**Output:** Hello, DevOps!



**4. Comments**

* Text in a script ignored by the shell, used for documentation.
* Symbol: # (except in shebang).

**Example:**

#!/bin/bash

# This script prints hello

echo "Hello World"

**5. Printing Output**

* Displays text or values.
* Commands: echo, printf.

**Example:**

echo "Welcome to DevOps"

printf "User: %s\n" "Chinni"

**Output:**

Welcome to DevOps

User: Chinni

**6. Variables**

Variables store values (strings, numbers, paths).

Types:

* User-defined → created by you
* Environment → predefined (e.g., $HOME, $USER)

**Example:**

#!/bin/bash

name="Chinni"

echo "Hello $name"

echo "Your home directory is $HOME"

**Output:**

Hello Chinni

Your home directory is /home/chinni

**7. User Input (read)**

Allows users to provide input during script execution.

**Example:**

#!/bin/bash

echo "Enter your name: "

read username

echo "Welcome $username!"

Run → enter DevOps **Output:**

Enter your name:

DevOps

Welcome DevOps!

**8. Command Substitution**

Runs a command and stores/prints its result.

Syntax: $(command) or `command`

**Example:**

#!/bin/bash

today=$(date)

echo "Today is $today"

**Output:**

Today is Fri Sep 12 21:10:15 IST 2025

**9. Exit Status ($?)**

* Every command returns an exit code → 0 (success), non-zero (error).
* Use: Check if a command worked before moving on.

**Example:**

#!/bin/bash

ls /notexist

echo "Exit status: $?"

**Output:**

ls: cannot access '/notexist': No such file or directory

Exit status: 2

**10. Quotes in Shell**

* Single quotes (' ') → literal (no variable expansion).
* Double quotes (" ") → expands variables.
* Backslash () → escapes characters.

**Example:**

#!/bin/bash

name="DevOps"

echo "Hello $name" # variable expands

echo 'Hello $name' # literal

echo "Path: \$HOME" # escape

**Output:**

Hello DevOps

Hello $name

Path: $HOME

**11. Arithmetic Operations**

Perform calculations inside scripts.

Syntax: $((expression)) or expr.

**Example:**

#!/bin/bash

a=5

b=3

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

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

**Output:**

Sum: 8

Product: 15

**12. Script Arguments**

Pass values when running a script.

Special variables:

* $0 = script name
* $1, $2 = first, second argument
* $# = number of arguments
* $@ = all arguments

**Example (args.sh):**

#!/bin/bash

echo "Script name: $0"

echo "First arg: $1"

echo "Second arg: $2"

echo "Number of args: $#"

Run: ./args.sh AWS DevOps

**Output:**

Script name: ./args.sh

First arg: AWS

Second arg: DevOps

Number of args: 2

**🧠Intermediate Topics**

**1. Conditional Statements (if, if-else, elif)**

Used to make decisions in scripts based on conditions.

Syntax:

if [ condition ]; then

commands

elif [ condition ]; then

commands

else

commands

fi

**Example (ifelse.sh)**

#!/bin/bash

echo "Enter a number: "

read num

if [ $num -gt 10 ]; then

echo "Number is greater than 10"

elif [ $num -eq 10 ]; then

echo "Number is equal to 10"

else

echo "Number is less than 10"

fi

**Output**

Enter a number:

7

Number is less than 10

**2. Loops**

**For Loop:** Repeats commands for a list of items.

**Example (for.sh):**

#!/bin/bash

for i in 1 2 3 4 5

do

echo "Number: $i"

done

**Output:**

Number: 1

Number: 2

Number: 3

Number: 4

Number: 5

**While Loop:** Runs while condition is true.

**Example (while.sh):**

#!/bin/bash

count=1

while [ $count -le 3 ]

do

echo "Count: $count"

((count++))

done

**Output:**

Count: 1

Count: 2

Count: 3

**Until Loop:** Runs until condition becomes true.

**Example (until.sh):**

#!/bin/bash

count=1

until [ $count -gt 3 ]

do

echo "Count: $count"

((count++))

done

**Output:**

Count: 1

Count: 2

Count: 3

**3. Case Statement:** Alternative to multiple if-else.

**Syntax:**

case $var in

pattern1) commands ;;

pattern2) commands ;;

\*) default ;;

esac

**Example (case.sh)**

#!/bin/bash

echo "Enter a choice: start/stop/restart"

read action

case $action in

start) echo "Starting service..." ;;

stop) echo "Stopping service..." ;;

restart) echo "Restarting service..." ;;

\*) echo "Invalid choice" ;;

esac

**Output**

Enter a choice: start/stop/restart

restart

Restarting service...

**4. Functions**

A function is a reusable block of code inside a script.

Makes scripts modular & avoids repetition.

**Example (function.sh)**

#!/bin/bash

greet() {

echo "Hello $1, welcome to DevOps scripting!"

}

greet "Chinni"

greet "Ravi"

**Output**

Hello Chinni, welcome to DevOps scripting!

Hello Ravi, welcome to DevOps scripting!

**5. Arrays**

An array stores multiple values in one variable.

Access: ${array[index]}, loop with "${array[@]}".

**Example (array.sh)**

#!/bin/bash

clouds=("AWS" "Azure" "GCP")

echo "First cloud: ${clouds[0]}"

for c in "${clouds[@]}"

do

echo "Cloud: $c"

done

**Output**

First cloud: AWS

Cloud: AWS

Cloud: Azure

Cloud: GCP

**6. String Operations**

Strings can be checked, compared, and manipulated.

**Example (string.sh):**

#!/bin/bash

str="DevOps"

echo "Length: ${#str}"

echo "Substring (first 3 chars): ${str:0:3}"

if [ "$str" == "DevOps" ]; then

echo "String matched"

fi

**Output**

Length: 6

Substring (first 3 chars): Dev

String matched

**7. File Test Operators**

Used to check properties of files and directories.

| Operator | Meaning |
| --- | --- |
| -f file | True if file exists and is a regular file |
| -d dir | True if directory exists |
| -r file | True if readable |
| -w file | True if writable |
| -x file | True if executable |

**Example (filetest.sh)**

#!/bin/bash

if [ -f "/etc/passwd" ]; then

echo "File exists"

else

echo "File not found"

fi

**Output**

File exists

**8. Numeric & String Comparisons**

Used in conditions inside scripts.

| Operator | Meaning |
| --- | --- |
| -eq | equal |
| -ne | not equal |
| -lt | less than |
| -gt | greater than |
| == | strings equal |
| != | strings not equal |

**Example (compare.sh)**

#!/bin/bash

a=5

b=10

if [ $a -lt $b ]; then

echo "$a is less than $b"

fi

**Output:** 5 is less than 10

**9. Input/Output Redirection**

Redirects command output/input.

| Symbol | Meaning |
| --- | --- |
| > | overwrite output to file |
| >> | append output to file |
| < | take input from file |
| 2> | redirect errors |
| &> | redirect both stdout & stderr |

**Example (redirect.sh)**

#!/bin/bash

echo "Hello World" > output.txt

cat output.txt

**Output:** Hello World

**10. Pipes**

A pipe (|) passes output of one command as input to another.

**Example (pipe.sh):**

#!/bin/bash

ps aux | grep bash

**Output:**(Shows all running processes with "bash" in them)

**11. Debugging**

Run a script in debug mode with bash -x script.sh

Shows each command before execution.

**Example:** bash -x hello.sh

**Output (with debug trace):**

+ echo 'Hello, DevOps!'

Hello, DevOps!

**12. Break**

break is used inside loops (for, while, until) to terminate the loop immediately, regardless of the condition.

After break, the control jumps out of the loop and continues with the next command after the loop.

**Example with break:**

#!/bin/bash

# Break Example

for num in 1 2 3 4 5

do

if [ $num -eq 3 ]; then

echo "Found 3! Exiting loop..."

break

fi

echo "Number: $num"

done

echo "Loop ended."

**Output:**

Number: 1

Number: 2

Found 3! Exiting loop...

Loop ended.

**13. Continue**

continue is used inside loops to skip the rest of the current iteration and move to the next iteration of the loop.

The loop itself is not terminated.

**Example with continue:**

#!/bin/bash

# Continue Example

for num in 1 2 3 4 5

do

if [ $num -eq 3 ]; then

echo "Skipping number 3..."

continue

fi

echo "Number: $num"

done

echo "Loop completed."

**Output:**

Number: 1

Number: 2

Skipping number 3...

Number: 4

Number: 5

Loop completed.

**Use case:** Skip processing unwanted data but continue the loop.

**Real-world DevOps Example:**

Script: Stop at first failed server check

#!/bin/bash

servers=("google.com" "invalidsite.com" "yahoo.com")

for server in "${servers[@]}"

do

ping -c 1 $server > /dev/null 2>&1

if [ $? -ne 0 ]; then

echo "Server $server is down! Stopping checks."

break

fi

echo "Server $server is UP."

done

**Output:**

Server google.com is UP.

Server invalidsite.com is down! Stopping checks.

**🚀Advanced Shell Scripting Topics**

**1. Error Handling (set -e, trap, ||, &&)**

Error handling ensures that your script **doesn’t silently fail** when a command breaks.

* set -e: Exit immediately if a command fails.
* trap: Run specific commands when a signal/error occurs.
* ||: Run next command only if the previous fails.
* &&: Run next command only if the previous succeeds.

**Example**

#!/bin/bash

set -e # Exit on error

echo "Starting script..."

ls /tmp/testfile || echo "File not found!"

echo "This line won’t run if an error occurs."

**Output**

Starting script...

ls: cannot access '/tmp/testfile': No such file or directory

File not found!

**2. Logging (redirecting to log files, timestamps)**

**Definition**

Logging records script activities for debugging & auditing.

> overwrite log

>> append log

Use date for timestamps

**Example**

#!/bin/bash

logfile="script.log"

echo "$(date): Script started" >> $logfile

echo "Running a task..." >> $logfile

**Output in script.log:**

Fri Sep 12 20:00:00 IST 2025: Script started

Running a task...

**3. File Handling (cat, grep, awk, sed)**

Scripts often process files (create, read, update, delete).

* cat: display file
* grep: search text
* awk: column/row processing
* sed: text replacement

**Example**

#!/bin/bash

echo "user1,user2,user3" > users.txt

cat users.txt

grep "user1" users.txt

awk -F',' '{print $2}' users.txt

sed 's/user2/admin/' users.txt

**Output**

user1,user2,user3

user1

user2

user1,admin,user3

**4. Signals & Traps**

trap lets you handle **signals** like SIGINT (Ctrl+C).

**Example**

#!/bin/bash

trap "echo 'Script interrupted! Cleaning up...'; exit" SIGINT

echo "Running... Press Ctrl+C to stop."

while true

do

sleep 2

done

**Output**

Running... Press Ctrl+C to stop.

^C Script interrupted! Cleaning up...

**5. Regular Expressions**

Regex helps **match patterns** in text.

* grep -E "pattern" → extended regex
* [[ string =~ regex ]] → bash regex

**Example**

#!/bin/bash

text="devops2025"

if [[ $text =~ [a-z]+[0-9]{4} ]]; then

echo "Matched: $text"

fi

**Output:** Matched: devops2025

**6. Scheduling with Cron Jobs**

Cron runs scripts at scheduled times.

Edit cron: crontab -e

Syntax: \* \* \* \* \* command

min hour day month weekday

**Example (run every 5 minutes)**

\*/5 \* \* \* \* /home/user/backup.sh >> backup.log 2>&1

**7. Process Management**

Manage processes inside scripts.

* & run in background
* ps list processes
* kill stop process

**Example**

#!/bin/bash

sleep 100 &

pid=$!

echo "Started process with PID $pid"

kill $pid

**Output:** Started process with PID 1234

**8. Working with JSON/YAML (jq, yq)**

Use tools to parse structured data in scripts.

**Example (JSON with jq)**

echo '{"name":"devops","year":2025}' | jq '.name'

**Output:** "devops"

**9. Command-line Arguments Parsing (getopts)**

getopts helps handle **flags/options** like -u username -p password.

**Example**

#!/bin/bash

while getopts u:p: flag

do

case "${flag}" in

u) user=${OPTARG};;

p) pass=${OPTARG};;

esac

done

echo "User: $user, Password: $pass"

**Run:** ./script.sh -u admin -p 1234

**Output:** User: admin, Password: 1234

**10. Automation Scripts**

**Examples**

* Backup: tar -czf backup.tar.gz /home/user/
* Monitoring: df -h | awk '$5 > 80 {print "Disk usage alert!"}'
* Deployment (copy files): scp app.tar.gz server:/opt/apps/

**11. Shell Script Optimization & Best Practices**

* Use set -euo pipefail (strict mode)
* Modularize with functions
* Log everything
* Avoid hardcoding paths (use variables)
* Test with shellcheck

**12. Integration in DevOps Pipelines**

* **Jenkins** → use shell scripts in pipeline stages
* **GitHub Actions** → define run: ./deploy.sh
* **Docker** → entrypoint scripts for containers
* **Kubernetes** → init scripts for pods

