**Shell Scripting**

**28/01/2025**

1.What is Shell Scripting?

Shell scripting is the process of writing programs called scripts,

leverage the power of the command-line interface (CLI)

It is used to automate task on Unix-based OS like Linux.

2.Why Use Shell Scripting?

Example: Manually copying and renaming dozens of files every day. A shell script can automate this process, saving you time and effort.

* Automation
* Efficiency
* Customization
* Portability
* Resource Management

3. Common Shells for Scripting.

* Bash
* Sh
* Zsh
* Ksh

**Executing file :**

* We can execute shell file in two different ways.

Sh filename or ./Filename

**Practical:**

Creating a new file with .sh extensioin and running the file.







**Popular shells:**

* **Bash (Bourne Again Shell)** – Default on most Linux distributions.
* **Sh (Bourne Shell)** – Older but still used in scripts.
* **Ksh (Korn Shell)** – More advanced features than Bash.
* **Csh (C Shell)** – Uses a syntax similar to C programming.
* **Zsh (Z Shell)** – An improved version of Bash.

**Add a shebang (first line) to specify the shell:**

#1#!/bin/bash echo

#2 "Hello, World**!"**

**Shell User Input**

**Reading user input using read**

echo "Enter your name:"

read name

echo "Hello, $name!"

**-p flag allows inline prompts:**

read -p "Enter your age: " age

**-s flag hides input (useful for passwords):**

read -s -p "Enter password: " password

**Shell Using Variables**

**Defining Variables**

name="John Doe"

age=25

**Accessing Variables**

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

**Read-only Variables**

readonly pi=3.14

pi=3.141 # This will give an error

**Unsetting Variables**

unset name

**Shell Operators**

**Arithmetic Operators**

a=10

b=5

echo $((a + b)) # Addition

echo $((a - b)) # Subtraction

echo $((a \* b)) # Multiplication

echo $((a / b)) # Division

**Comparison Operators**

[ $a -eq $b ] # Equal

[ $a -ne $b ] # Not equal

[ $a -lt $b ] # Less than

[ $a -gt $b ] # Greater than

**Logical Operators**

[ $a -gt 0 -a $b -gt 0 ] # AND (&&)

[ $a -gt 0 -o $b -lt 0 ] # OR (||)

**String Operators**

[ -z "$str" ] # String is empty

[ -n "$str" ] # String is not empty

**6. Shell Loops**

**For Loop**

for i in 1 2 3 4 5; do

echo "Number: $i"

done

**While Loop**

count=1

while [ $count -le 5 ]; do

echo "Count: $count"

((count++))

done

**Until Loop**

until [ $count -gt 5 ]; do

echo "Count: $count"

((count++))

done

**7. Shell Arguments & Passing Arguments**

**Accessing Arguments**

echo "Script Name: $0"

echo "First Argument: $1"

echo "Second Argument: $2"

**Passing Multiple Arguments**

./myscript.sh arg1 arg2 arg3

**Accessing All Arguments**

echo "All arguments: $@"

echo "Total number of arguments: $#"

**8. Shell Conditions**

**If Statement**

if [ $age -gt 18 ]; then

echo "You are an adult."

fi

**If-Else**

if [ $age -gt 18 ]; then

echo "Adult"

else

echo "Minor"

fi

**Else If (elif)**

if [ $age -lt 13 ]; then

echo "Child"

elif [ $age -lt 18 ]; then

echo "Teen"

else

echo "Adult"

fi

**9. Shell File Test Operators**

[ -e file.txt ] # Exists

[ -f file.txt ] # Regular file

[ -d dir ] # Directory

[ -r file.txt ] # Readable

[ -w file.txt ] # Writable

[ -x file.txt ] # Executable

**10. Shell Advanced Operators**

* **Ternary Operator** (Using && and ||)

[ $a -gt $b ] && echo "A is greater" || echo "B is greater"

* **Case Statement**

case $1 in

start) echo "Starting";;

stop) echo "Stopping";;

\*) echo "Invalid option";;

esac

**11. Shell Arrays**

arr=(one two three)

echo ${arr[0]} # First element

echo ${arr[@]} # All elements

**12. Foreground & Background Modes**

./script.sh & # Run in background

jobs # List background jobs

fg %1 # Bring job 1 to foreground

**13. Exit Status**

echo $? # Prints exit status of last command

exit 0 # Exit with status 0

**14. Standard Input and Output**

echo "Hello" > file.txt # Write to file

cat < file.txt # Read from file

ls > output.txt # Redirect output

**15. Special Files**

* /dev/null – Discards output.
* /etc/passwd – User accounts.
* /proc/cpuinfo – CPU information.

**16. Piping**

ls | grep "test"

ps aux | grep "bash"

**17. Advanced Commands**

* awk – Pattern scanning and processing.
* sed – Stream editing.
* cut – Extract columns.
* find – Search files.

**Square Brackets [ ] (Test Command)**

* Used for conditional expressions and file tests.
* Equivalent to the test command in shell scripting.
* Requires spaces around brackets.

**Usage Examples**

**String Comparisons**

str="hello"

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

echo "Strings match"

fi

**Numerical Comparisons**

num=10

if [ $num -eq 10 ]; then

echo "Number is 10"

fi

**File Tests**

if [ -f myfile.txt ]; then

echo "File exists"

fi

**Logical AND/OR**

if [ $num -gt 5 ] && [ $num -lt 15 ]; then

echo "Number is between 5 and 15"

fi

**2. Double Square Brackets [[ ]] (Enhanced Test Command)**

* More powerful than [], specific to **Bash/Ksh/Zsh**.
* Supports regex matching (=~).
* No need to escape &&, ||, <, or >.

**Usage Examples**

**String Comparisons**

if [[ "hello" == "hello" ]]; then

echo "Match"

fi

**Regex Matching**

if [[ "hello123" =~ ^hello[0-9]+$ ]]; then

echo "Regex matched"

fi

**Logical Operators without Escape**

if [[ $num -gt 5 && $num -lt 15 ]]; then

echo "Number is between 5 and 15"

fi

**3. Curly Braces {} (Variable Expansion & Code Blocks)**

* Used for variable expansion, grouping commands, and brace expansion.

**Usage Examples**

**Variable Expansion**

name="World"

echo "Hello, ${name}!"

* {} is needed when appending strings to variables:

var="file"

echo "${var}name" # Correct

echo "$varname" # Incorrect, shell looks for $varname

**Command Grouping**

* {} groups multiple commands.
* Commands inside {} run in the **current shell**.

{ echo "Hello"; echo "World"; }

**Brace Expansion**

echo {a,b,c} # Output: a b c

echo {1..5} # Output: 1 2 3 4 5

**4. Parentheses () (Subshell Execution & Command Grouping)**

* Used to execute commands in a **subshell**.

**Usage Examples**

**Subshell Execution**

* Commands inside () run in a separate shell process.

(current\_dir=$(pwd); echo "Inside subshell: $current\_dir")

echo "Outside subshell: $current\_dir"

* Variables defined inside () are **not accessible outside**.

**Command Grouping**

* Similar to {}, but runs in a **subshell**.

(echo "Line 1"; echo "Line 2")

**5. Double Parentheses (()) (Arithmetic Operations)**

* Used for arithmetic evaluations.
* Supports operators like +, -, \*, /, %, ++, --.

**Usage Examples**

**Arithmetic Operations**

num=10

((num++)) # Increment num

echo $num # Output: 11

**Arithmetic Comparisons**

if (( num > 5 )); then

echo "Number is greater than 5"

fi

* More readable than [ $num -gt 5 ].

**Inline Arithmetic**

echo $((10 + 5)) # Output: 15

**6. Angle Brackets < > (Redirection)**

* Used for input/output redirection.

**Usage Examples**

**Input Redirection**

wc -l < myfile.txt # Count lines in myfile.txt

**Output Redirection**

echo "Hello" > file.txt # Write to file

echo "World" >> file.txt # Append to file

**Comparison Table**

|  |  |  |
| --- | --- | --- |
| **Bracket Type** | **Purpose** | **Example** |
| [ ] | Basic conditions | [ $a -gt 5 ] |
| [[ ]] | Advanced conditions | [[ $str == "test" ]] |
| { } | Variable expansion, grouping | echo ${var}name |
| ( ) | Subshell execution | (echo "Inside subshell") |
| (( )) | Arithmetic operations | ((num++)) |
| < > | Redirection | echo "Hello" > file.txt |

**Summary**

* Use **[ ]** for simple conditions and file checks.
* Use **[[ ]]** for advanced conditions, regex, and better syntax.
* Use **{}** for variable expansion, grouping commands, and brace expansion.
* Use \*\*( )\*\* for subshell execution.
* Use **(())** for arithmetic operations.
* Use **< >** for input/output redirection.