Linux (Continue)

Shell Scripting I: Basics & System Info Script

Overview & Learning Outcomes

This lab you will:

- Understand what a **shell script** is and why we use it
- **Create** and **run** simple bash scripts
- Use **variables** and **echo** to show information
- **Read user input** using read
- Write simple **if-statements** for decision-making

What Is a Shell Script?

A shell script is:

- A text file containing a list of **shell commands**
- Executed by a shell (e.g., **bash**) line by line

Why use scripts?

- **Automate** repetitive tasks
- Save and **reuse** command sequences
- Create simple **tools** for system administration

How Shell Executes Scripts

Execution model:

- Read one line from the script
- Parse and execute the command
- Move to the next line

Two common ways to run:

- `bash script.sh` (run with bash explicitly)
- `./script.sh` (run as executable file)

NOTE: Scripts often start with a shebang line

Shebang & Running Scripts

Shebang line: tells system which **interpreter** to use

- For example (first line of file):
  - `#!/bin/bash`

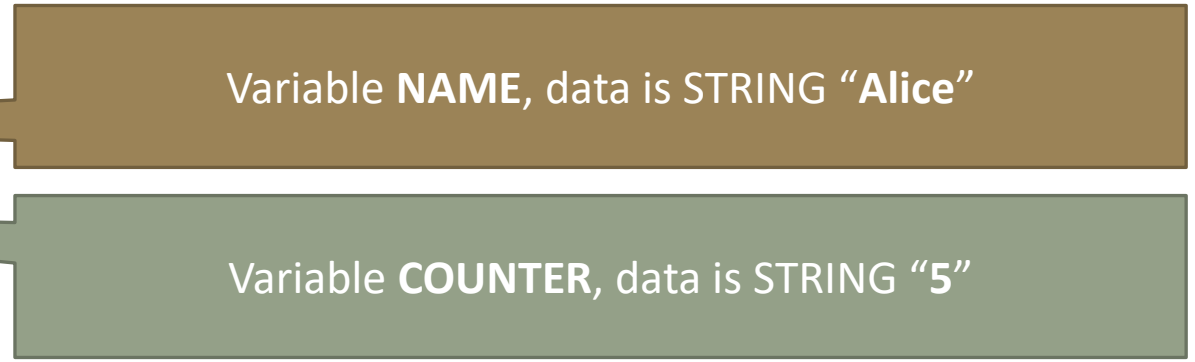
Creating & running a script:

1. Create file: **`vim myscript.sh`** (or another editor)
2. Add **shebang** and commands
3. Make executable: **`chmod +x myscript.sh`**
4. Run: **`./myscript.sh`**

Variables & echo

Creating variables:

- `NAME="Alice"`
- `COUNTER=5`



Using variables:

```
echo "Hello $NAME"                    # output is "Hello Alice"
echo "The counter is $COUNTER"        # output is "The counter is 5"
```

NOTE: **NO SPACE** around "**=**" in variable assignment

Reading User Input with read

"**read**" command: get input from user

- For example:

```
read NAME
echo "Hello, $NAME"
```

- With prompt:

```
read -p "Enter your name: " NAME
```

NOTE: Input is stored in the **variable** for later use

Basic Conditions – if Statements

Simple if-statement:

```
if [ CONDITION ]; then
    commands
fi
```

Make sure to keep a space between each word/symbol

For example:

```
read -p "Enter a number: " NB
if [ "$NB" -gt 10 ]; then
    echo "NB is greater than 10"
fi
```

NOTE: In bash/shell script, **“if”** conditions are built from **test expressions**.

`test expressions` or `[ … ]` or `[[ … ]]`

Practically, you use either:

- `[ ... ]` : POSIX test (`test expressions`)
- `[[ ... ]]` : bash/shell/ksh enhanced test
- `(( ... ))` : Arithmetic test

NOTE: Recommended to use `[[ … ]]` in bash

Numeric (integer) Comparisons in `[ ... ]` or **`test`**

No.	Comparison	Description
1	`-eq`	Equal
2	`-ne`	Not equal
3	`-lt`	Less than
4	`-le`	Less than or equal
5	`-gt`	Greater than
6	`-ge`	Greater than or equal

```
if [ "$n" -ge 60 ]; then echo "pass"; fi

if (( n >= 60 )); then echo "pass"; fi
```

String Comparisons in **[…]** or **[[…]]**

No.	Comparison	Description
1	= , ==	Equal, **=** use in […] or **==** use in [[…]]
2	!=	Not equal
3	< , >	Exicographic compare in [[…]]
4	\< , \>	Exicographic compare in […]
5	-z s	String length is zero (empty)
6	-n s	String length is non-zero

```
if [[ "$a" == "$b" ]]; then echo "same"; fi
if [[ -z "$name" ]]; then echo "missing"; fi
if [[ "$x" < "$y" ]]; then echo "x comes first"; fi
```

File Tests in […] or [[…]] (Existence / Type)

No.	Check Type	Description
1	`-e path`	Exists
2	`-f path`	Regular file
3	`-d path`	Directory
4	`-L path`	Symlink
5	`-b path`	Block device
6	`-c path`	Char device
7	`-p path`	Named pipe
8	`-S path`	Socket

File Tests in […] or [[…]] (Permissions)

No.	Check Permission	Description
1	-r path	Readable
2	-w path	Writable
3	-x path	Executable
4	-s path	Size > 0 (non-empty)

```
if [[ -d "$dir" && -w "$dir" ]]; then
  echo "can write into directory"
fi
```

File Tests in […] or [[…]] (Comparisons)

No.	Comparison	Description
1	`file1 -nt file2`	Newer than
2	`file1 -ot file2`	Older than
3	`file1 -ef file2`	Same file (same inode)

```
if [[ "hello.txt" -nt "lab04/output/myaccess.log" ]]; then
  echo "myaccess.log is newer than hello.txt"
fi
```

Logical Operators […] or [[…]]

No.	Logical operator	Description
1	! expr	NOT
2	expr1 && expr2	AND
3	expr1 || expr2	OR
Parentheses for grouping: (expr) inside [[…]]		

```
if [[ -f "$f" && ! -s "$f" ]]; then
  echo "file exists but is empty"
fi
```

```
if [ -r "$f" ] && [ -w "$f" ]; then
  echo "readable and writable"
fi
```

Good Practices in Shell Scripts

- Start with a clear **shebang** line
- Use meaningful **variable** names
- Add **comments** with # to explain sections
- **Print** clear messages for the user
- Keep scripts small and **focused on one task**
- **Test scripts** on non-critical data first

Thank You