🐚Lab 2 - Script Execution & Explanation

Shell scripting allows you to **automate tasks** in Linux/Unix by writing commands inside a file that the shell executes line by line.

---

1. 🔹 What is a Shell Script?

- A **shell** is a command-line interpreter (e.g., `bash`, `zsh`, `sh`).
- A **shell script** is a text file with a series of commands.
- File usually has `.sh` extension, though not mandatory.

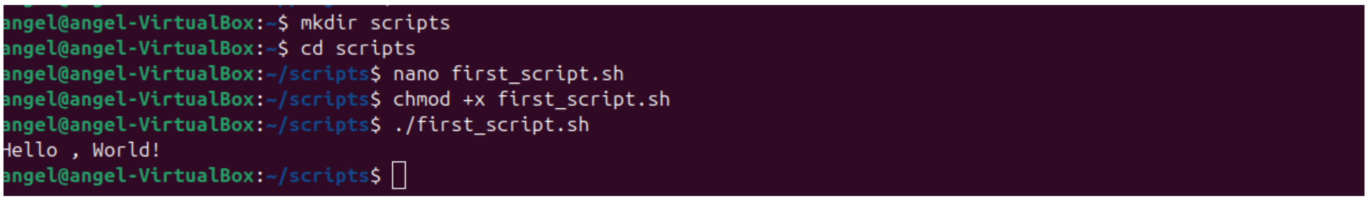
Example: `hello.sh`

```
#!/bin/bash
echo "Hello, World!"
```

Run it:

```
chmod +x hello.sh   # make it executable
./hello.sh
```

📸Output image of the commands :



```
angel@angel-VirtualBox:~$ mkdir scripts
angel@angel-VirtualBox:~$ cd scripts
angel@angel-VirtualBox:~/scripts$ nano first_script.sh
angel@angel-VirtualBox:~/scripts$ chmod +x first_script.sh
angel@angel-VirtualBox:~/scripts$ ./first_script.sh
Hello , World!
angel@angel-VirtualBox:~/scripts$ 
```

2. 🔹 Variables

Variables store data (text, numbers, paths, etc.).

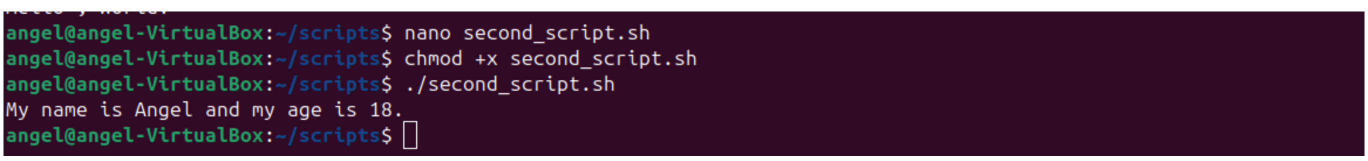
Defining variables

```
name="Angel"
age=18
```

Environment variables

```
echo $HOME    # home directory
echo $USER    # current user
echo $PWD    # present working directory
```

📸Output image of the commands :



```
angel@angel-VirtualBox:~/scripts$ nano second_script.sh
angel@angel-VirtualBox:~/scripts$ chmod +x second_script.sh
angel@angel-VirtualBox:~/scripts$ ./second_script.sh
My name is Angel and my age is 18.
angel@angel-VirtualBox:~/scripts$ 
```

Operators:

- `-eq` (equal)
- `-ne` (not equal)
- `-gt` (greater than)
- `-lt` (less than)
- `-ge` (greater or equal)
- `-le` (less or equal)

---

3. 🔹 Loops

For loop

```
for i in 1 2 3 4 5
do
    echo "Number: $i"
done
```

Or use a range:

```
for i in {1..5}
do
    echo "Iteration $i"
done
```

While loop

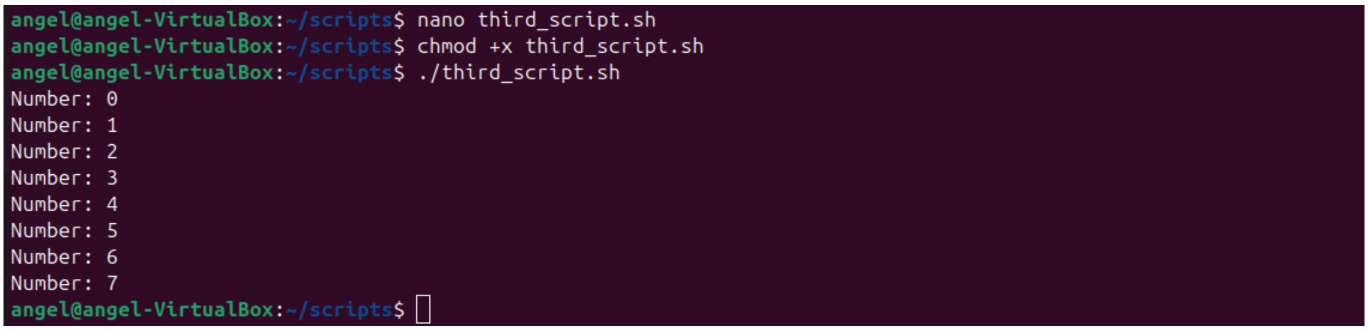
```
count=1
while [ $count -le 5 ]
do
    echo "Count: $count"
    ((count++))   # increment
done
```

Until loop

Runs until condition becomes true.

```
x=1
until [ $x -gt 5 ]
do
    echo "Value: $x"
    ((x++))
done
```

📸Output image of the commands :



```
angel@angel-VirtualBox:~/scripts$ nano third_script.sh
angel@angel-VirtualBox:~/scripts$ chmod +x third_script.sh
angel@angel-VirtualBox:~/scripts$ ./third_script.sh
Number: 0
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
Number: 6
Number: 7
angel@angel-VirtualBox:~/scripts$
```

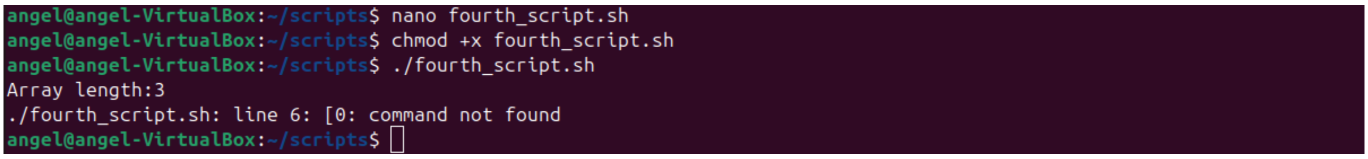
4. 🔹 Arrays

```
fruits=("apple" "banana" "cherry")

echo "First fruit: ${fruits[0]}"

for fruit in "${fruits[@]}"; do
    echo "Fruit: $fruit"
done
```

📸Output image of the commands :



```
angel@angel-VirtualBox:~/scripts$ nano fourth_script.sh
angel@angel-VirtualBox:~/scripts$ chmod +x fourth_script.sh
angel@angel-VirtualBox:~/scripts$ ./fourth_script.sh
Array length:3
./fourth_script.sh: line 6: [0: command not found
angel@angel-VirtualBox:~/scripts$
```

---

9. 🔹 Useful Commands in Scripts

- `date` → show current date/time
- `whoami` → show current user
- `ls` → list files
- `pwd` → print working directory
- `cat` → read file contents

10. 🔹 A Practical Example

Backup script (`backup.sh`):

```
#!/bin/bash
# Backup home directory to /tmp

backup_file="/tmp/home_backup_$(date +%Y%m%d%H%M%S).tar.gz"

tar -czf $backup_file $HOME

echo "Backup saved to $backup_file"
```

Run:

```
./backup.sh
```

📌Extra Questions

Q1. What is the purpose of `#!/bin/bash` at the top of a script?

The line `#!/bin/bash` is called a **shebang** (or **hashbang**), and it serves an important role in shell scripting.

Explanation:

- **`#!/bin/bash`** specifies the path to the **Bash shell** that will be used to interpret and execute the script.
- This allows the system to know which interpreter (in this case, `bash`) to use when running the script, regardless of the user's current shell environment.

Why it's important:

- It makes the script **portable**. Even if the script is executed in a different environment, it will always use `/bin/bash` as the interpreter.
- Without this line, the script might not run correctly, or it may run with an incorrect interpreter, depending on the environment.

Example:

```
#!/bin/bash
echo "Hello, World!"
```

Q2. How do you make a script executable?

To make a script executable, you need to change its file permissions to allow execution.

Steps to Make a Script Executable:

1. **Use `chmod` to add execute permissions**:

```
```

chmod +x script_name.sh Run the script:

After making the script executable, you can run it directly: ./script_name.sh