3.1 Creating and Executing Shell Scripts

A shell script is simply a text file containing a series of commands that the Linux shell can execute. Think of it like a recipe for your computer.

How to Create a Script (using vi):

You will use the vi text editor to write your scripts.

1. Open your terminal.
2. Type vi my_script.sh and press Enter. This will open the vi editor.

   Note: Defult is command mode need to go in insert mode

3. **Press i** (for "insert" mode) to start typing.
4. Type your commands inside the editor.
5. To save and exit: **Press Esc** (to exit insert mode), then type **:wq** (write and quit), and press Enter.

Example Script (hello.sh):

Let's create a simple script that prints "Hello, Shell Scripting!"

1. Open vi hello.sh.
2. **Press i** to enter insert mode.
3. Add these two lines:

   Bash

   #!/bin/bash

   echo "Hello, Shell Scripting!"

   - #!/bin/bash: This is called a "shebang." It tells your system which shell interpreter to use to run the script (in this case, bash). Always put this at the very top of your script.
   - echo "Hello, Shell Scripting!": This command simply prints the text inside the quotes to your terminal.

4. **Press Esc** to exit insert mode.
5. Type **:wq** and press Enter to save and quit.

How to Execute a Script:

Used sh command to run the script

Ex : sh hello.sh

What you'll see:

Hello, Shell Scripting!

3.2 Shell Metacharacters and Operators

These are special characters and symbols that have specific meanings in the shell, helping you do powerful things.

3.2.1 Filename Expansion (Wildcards)

Wildcards are special characters you use to match patterns in filenames. They're super useful for selecting multiple files at once.

- * (Asterisk): Matches *any* sequence of characters (including no characters).
  - **Example:** ls *.txt will list all files ending with .txt.
  - **Example:** rm photo*.jpg will delete all JPEG files starting with "photo".
- **?** (Question Mark): Matches *any single* character.
  - **Example:** ls file?.txt will match file1.txt, fileA.txt, but not file10.txt.
- **[ ]** (Square Brackets): Matches *any one* of the characters inside the brackets. You can also specify a range.
  - **Example:** ls [abc]file.txt will match afile.txt, bfile.txt, cfile.txt.
  - **Example:** ls [0-9]report.pdf will match 1report.pdf, 2report.pdf, etc.

Note: if we want to use meta characters as normal characters, we should use '\' before the character.

3.2.2 Input/Output Redirection (I/O Redirection)

Normally, commands take input from your keyboard (standard input) and display output on your screen (standard output). Redirection lets you change where the input comes from or where the output goes.

- > (Redirect Standard Output/Output Redirection): Sends the command's output to a file. If the file exists, it will be overwritten.

Example: ls -l > file_list.txt

This will save the detailed list of files (ls -l) into a new file called file_list.txt. If file_list.txt already exists, its contents will be replaced.

- **>>** (Append Standard Output/ Append Redirection): Sends the command's output to a file, *adding it to the end* if the file already exists. If the file doesn't exist, it creates it.

Example: echo "Another line" >> file_list.txt

- This will add "Another line" to the end of file_list.txt without deleting its existing content.
- **<** (Redirect Standard Input/ Input Redirection): Takes input for a command from a file instead of the keyboard.

  Example: wc -l < file_list.txt

- wc -l counts lines. This command will count the lines in file_list.txt as if you typed its content directly.

3.2.3 Pipes (|)

Pipes let you connect commands by sending the output of one command as the input to another command. Think of it as a conveyor belt passing goods from one machine to the next.

- **Example:** ls -l | grep "txt"
  - ls -l lists files in detail.
  - | takes the output of ls -l and feeds it as input to grep.
  - grep "txt" searches for lines containing "txt".
  - **Result:** This command will show you only the lines from ls -l that contain "txt", effectively listing only .txt files from the detailed output.

3.2.4 Command Substitution ($())

Command substitution allows you to use the output of a command as part of another command. Which means execute one command inside another command.

- **Example:** echo "Today's date is: $(date)"
  - $(date) will run the date command.

- The output of the date command (e.g., "Fri Jul 18 08:05:07 AM IST 2025") will then replace $(date) in the echo command.
- **Result:** Today's date is: Fri Jul 18 08:05:07 AM IST 2025 (the date will be your current date and time).

3.3 Control Flow Structures

These structures allow your script to make decisions and repeat actions, making them much more powerful and dynamic.

if-else :

The if-else statement allows your script to execute different blocks of code based on whether a condition is true or false.

- **Syntax:**

  Bash

  ```
  if [ condition ]; then
     # Commands to execute if condition is true
  elif [ condition ]; then
    # Commands to execute if condition is true
  else
     # Commands to execute if condition is false
  fi
  ```

- **Example:** Check if a file exists.
  1. Open vi check_file.sh.
  2. **Press i** to enter insert mode.
  3. Add these lines:

     Bash

     ```
     #!/bin/bash
     FILE="my_file.txt"

     if [ -f "$FILE" ]; then
        echo "$FILE exists."
     else
        echo "$FILE does not exist."
     ```

```
fi
```

- [-f "$FILE"]: This is a condition that checks if $FILE is a regular file.

4. **Press Esc**, then type **:wq** and Enter.
5. chmod +x check_file.sh
6. ./check_file.sh

- Try creating my_file.txt with touch my_file.txt and running it again to see the different output.

Case:

The case statement is useful when you have multiple possible values for a variable and want to run different commands for each value. It's like a cleaner if-else if-else chain.

- **Syntax:**

  Bash

```
case $variable in
  pattern1)
    # Commands for pattern1
    ;;
  pattern2)
    # Commands for pattern2
    ;;
  *)
    # Default commands (if no pattern matches)
    ;;
esac
```

- **Example:** Greet based on a fruit choice.
  1. Open vi fruit_greeter.sh.
  2. **Press i** to enter insert mode.
  3. Add these lines:

     Bash

```
#!/bin/bash
read -p "Enter your favorite fruit (apple/banana/orange): " FRUIT

case $FRUIT in
  apple)
    echo "Apples are crisp and delicious!"
    ;;
  banana)
    echo "Bananas are great for energy!"
    ;;
  orange)
    echo "Oranges are full of Vitamin C!"
    ;;
  *)
    echo "That's an interesting choice!"
    ;;
esac
```

- read -p "..." FRUIT: This prompts the user and stores their input in the FRUIT variable.

4. **Press Esc**, then type **:wq** and Enter.
5. chmod +x fruit_greeter.sh
6. ./fruit_greeter.sh (try entering "apple", "banana", "orange", or something else).

for loop :

The for loop allows you to repeat a set of commands for each item in a list.

- **Syntax:**

  Bash

  for item in list_of_items;

```
do
   # Commands to execute for each item
done
```

- **Example:** Process multiple files.
  1. Open vi file_processor.sh.
  2. **Press i** to enter insert mode.
  3. Add these lines:

     Bash

     ```
     #!/bin/bash
     for FILE in doc1.txt doc2.pdf image.jpg;
      do
        echo "Processing $FILE..."
        # You could add commands here like:
        # cp "$FILE" /backup/
     done
     ```

  4. **Press Esc**, then type **:wq** and Enter.
  5. chmod +x file_processor.sh
  6. ./file_processor.sh

- **Example (numeric range):**

  1. Open vi counter.sh.
  2. **Press i** to enter insert mode.
  3. Add these lines:

     Bash

     ```
     #!/bin/bash
     for i in {1..5};
     do
        echo "Counting: $i"
     ```

done

- {1..5} expands to 1 2 3 4 5.

4. **Press Esc**, then type **:wq** and Enter.
5. chmod +x counter.sh
6. ./counter.sh

while loop :

The while loop repeatedly executes commands as long as a condition remains true.

- **Syntax:**

  Bash

  ```
  while [ condition ];
  do
     # Commands to execute as long as condition is true
  done
  ```

- **Example:** Countdown timer.
  1. Open vi countdown.sh.
  2. **Press i** to enter insert mode.
  3. Add these lines:

  ```
  #!/bin/bash
  COUNT=5
  while [ $COUNT -gt 0 ]; do
     echo "Countdown: $COUNT"
     sleep 1 # Wait for 1 second
     COUNT=$((COUNT - 1)) # Decrease COUNT by 1
  done
  echo "Blast off!"
  ```

  - [$COUNT -gt 0]: Checks if the value of COUNT is greater than 0.
  - sleep 1: Pauses the script for 1 second.
  - COUNT=$((COUNT - 1)): This is how you do arithmetic in shell scripts.

4. **Press Esc**, then type **:wq** and Enter.
5. chmod +x countdown.sh
6. ./countdown.sh

until loop :

The until loop is the opposite of while. It repeatedly executes commands as long as a condition remains *false*. It stops when the condition becomes true.

- **Syntax:**

  until [condition];

  do

  # Commands to execute as long as condition is false

  done

- **Example:** Wait for a file to appear.
  1. Open vi wait_for_file.sh.
  2. **Press i** to enter insert mode.
  3. Add these lines:

     Bash

     #!/bin/bash

     FILE="report.log"

     echo "Waiting for $FILE to appear..."

     until [-f "$FILE"]; do

     sleep 5 # Check every 5 seconds

     done

     echo "$FILE has appeared! Processing..."

  4. **Press Esc**, then type **:wq** and Enter.
  5. chmod +x wait_for_file.sh
  6. ./wait_for_file.sh (In another terminal, create report.log using touch report.log and observe wait_for_file.sh).

3.4 Logical Operators

These operators combine multiple conditions to create more complex decision-making in your scripts.

- **&&** (AND operator): Both conditions must be true for the whole expression to be true.
  - **Example:** if [-f "file.txt"] && [-w "file.txt"]; then echo "File exists and is writable."; fi
    - This checks if file.txt exists AND if it's writable.
- || (OR operator): At least one of the conditions must be true for the whole expression to be true.
  - **Example:** if [-f "file1.txt"] || [-f "file2.txt"]; then echo "At least one file exists."; fi
    - This checks if file1.txt exists OR file2.txt exists (or both).
- **!** (NOT operator): Reverses the truth value of a condition. If a condition is true,

  ! makes it false, and vice-versa.

  Example: if ! [-d "my_directory"]; then

  echo "my_directory does not exist.";

  fi

  - [-d "my_directory"] is true if the directory exists and ! makes it true if the directory does not exist.

3.5 test and [ ] command for Condition Testing

The test command (and its shorter alias [ ]) is used to evaluate conditions. We've seen it already in

if and while statements.

Common Tests:

- **File Tests:**
  - -f file: True if file is a regular file.
  - -d directory: True if directory is a directory.
  - -e file/directory: True if file or directory exists.
  - -r file: True if file is readable.
  - -w file: True if file is writable.
  - -x file: True if file is executable.
  - -nt file : check for new file (newer than).
  - -ot file: check for old file (older than).
- **Numeric Tests:**
  - num1 -eq num2: True if num1 is equal to num2.
  - num1 -ne num2: True if num1 is not equal to num2.
  - num1 -gt num2: True if num1 is greater than num2.
  - num1 -ge num2: True if num1 is greater than or equal to num2.
  - num1 -lt num2: True if num1 is less than num2.
  - num1 -le num2: True if num1 is less than or equal to num2.
- **String Tests:**
  - "string1" = "string2": True if string1 is equal to string2.
  - "string1" != "string2": True if string1 is not equal to string2.
  - -z "string": True if string is empty (zero length).
  - -n "string": True if string is not empty.

Example:

1. Open vi conditions.sh.
2. **Press i** to enter insert mode.
3. Add these lines:

   Bash

   ```
   #!/bin/bash
   NUM1=10
   ```

```
NUM2=20

if [ $NUM1 -lt $NUM2 ]; then
   echo "$NUM1 is less than $NUM2"
fi
STRING1="hello"
STRING2="world"

if [ "$STRING1" != "$STRING2" ]; then
   echo "$STRING1 and $STRING2 are different."
fi
```

4. **Press Esc**, then type **:wq** and Enter.
5. chmod +x conditions.sh
6. ./conditions.sh

Important Note on [ ]:

- Always put spaces around the brackets [condition].
- Always quote your variables inside [ ] (e.g., "$FILE", "$STRING1") to prevent issues if they contain spaces or are empty.

3.6 Arithmetic Operations (expr, $(()))

Shell scripts can also perform basic math!

expr command

The expr command is older but still works for simple arithmetic.

- **Example:**
  1. Open vi arithmetic_expr.sh.
  2. **Press i** to enter insert mode.
  3. Add these lines:

```
Bash

#!/bin/bash

RESULT=$(expr 5 + 3)

echo "5 + 3 = $RESULT"


# Note: For multiplication, you need to escape the asterisk or quote it

PRODUCT=$(expr 4 \* 2)

echo "4 * 2 = $PRODUCT"
```

  Notice the backslash before * for multiplication (*). This is because * is a wildcard in the shell and needs to be escaped so expr sees it as a multiplication operator.

  4. **Press Esc**, then type **:wq** and Enter.
  5. chmod +x arithmetic_expr.sh
  6. ./arithmetic_expr.sh

$(()) (Arithmetic Expansion)

This is the more modern and preferred way to do arithmetic in Bash. It's simpler and more flexible.

- **Syntax:** $((expression))
- **Example:**
  1. Open vi arithmetic_bash.sh.
  2. **Press i** to enter insert mode.
  3. Add these lines:

Bash

```
#!/bin/bash
A=10
B=5

SUM=$((A + B))
echo "Sum: $SUM"

DIFFERENCE=$((A - B))
echo "Difference: $DIFFERENCE"

PRODUCT=$((A * B))
echo "Product: $PRODUCT"

DIVISION=$((A / B))
echo "Division: $DIVISION"

MODULO=$((A % B)) # Remainder after division
echo "Modulo: $MODULO"
```

- You don't need expr or special escaping for operators like * inside $(()).
- You can directly use variable names without the $ inside the $(()) for readability, but using $ (e.g., $A) is also correct.

4. **Press Esc**, then type **:wq** and Enter.
5. chmod +x arithmetic_bash.sh
6. ./arithmetic_bash.sh