Assignment 9

Linux Programming

Name : Yathish N R

Class: 3B CY

Roll No: 66

USN : ENG25CY1007

1. **Write a shell script using if…else to check if a number is even or odd.**
   You can use the **modulo operator (\%)** inside an if...else structure. If a number divided by 2 has a remainder of 0, it's even; otherwise, it's odd.

```
#!/bin/bash

# Prompt the user for a number
read -p "Enter a number: "
number
# Use the arithmetic evaluation command '((...))' and the
modulo operator if (( number % 2 == 0 )); then     echo
"$number is **Even**." else     echo "$number is **Odd**."
fi
```

2. **Explain the difference between if and case statements in bash.**

The difference comes down to the kind of condition they check:

*if Statement*

- **Focus**: Evaluates a **series of logical conditions** (like greater than, file exists, or combinations using && and ||).
- **Structure**: Sequential checking (if, elif, else).
- **Best For**: **Complex decision-making** or checking non-equality and numerical ranges.

*case Statement*

1. **Focus**: Evaluates a **single variable** against multiple possible **patterns** (literal strings, numbers, or wildcards).
2. **Structure**: Direct matching. Executes the code block for the **first exact pattern match** it finds, then stops.

**Best For: Simplified menu handling** and replacing long, nested if-elif-else chains for better readability.

3. **Write a script to find the largest of three numbers entered by the user.**

This script takes three numbers from the user and uses nested if statements to compare them and find the largest.

```
#!/bin/bash

# Read the three numbers read -p "Enter the
first number (num1): " num1 read -p "Enter
the second number (num2): " num2 read -p
"Enter the third number (num3): " num3
# Start the comparison if (( num1 > num2 ))
&& (( num1 > num3 )); then     # num1 is
larger than both num2 and num3
largest=$num1
elif (( num2 > num3 )); then
    # If we are here, num1 wasn't the largest. Now check if num2
is larger than num3.     largest=$num2 else
    # If we are here, num1 wasn't the largest, and num2 wasn't
larger than num3.
    # Therefore, num3 must be the largest (or equal to others).
    largest=$num3 fi  echo "The largest number among
$num1, $num2, and $num3 is:
**$largest**."
```

4. **How do you use a for loop to traverse an array in bash? Give an example. The**

array is defined as arr=(123, "Abs", -2.3, 'A', 23.56, 0).
The most common way to iterate through all elements of a bash array is to use the special **@ or * array indices** inside the in clause of a for loop.
**Array Definition:** arr=(123, "Abs", -2.3, 'A', 23.56, 0)

```
#!/bin/bash

# Array definition
arr=(123 "Abs" -2.3 'A' 23.56 0)

echo "--- Array Elements ---"
# Loop through the array elements
# The construct "${arr[@]}" expands to *all* elements of the array.
```

```
for element in "${arr[@]}"; do
echo "Element: $element" done
echo "--- Traversal Complete ---
"
```

**Output:**

```
--- Array Elements ---
Element: 123
Element: Abs
Element: -2.3
Element: A
Element: 23.56
Element: 0
--- Traversal Complete ---
```

5. **Write a shell script to loop through all files in the current directory and display**

   **Their names.**

   Bash is very efficient at this. You can simply use a for loop with the **wildcard character (*)**, which the shell automatically expands to all files and directories in the current path.

   ```
   #!/bin/bash
    echo "Files and Directories in the current location
   ($(pwd)): " echo "-----------------------------------------
   --------------"
   # '*' is a wildcard that matches all file and directory names.
   for item in *; do
       # You can add a check if you only want
   *files*     if [ -f "$item" ]; then
   echo "File: **$item**"     # Or if you want
   directories     elif [ -d "$item" ]; then
   echo "Directory: $item/"     fi done
   ```

6. **What is the difference between while and until loops in bash?**

The difference is simple: **when they stop and when they continue.**

- **while Loop (The "As Long As" Loop)**
  - **Continues** to execute the loop body **AS LONG AS** the test condition is **TRUE** (returns an exit status of 0).
  - It stops when the condition becomes **FALSE** (returns a non-zero exit status).
- **until Loop (The "Keep Going Until" Loop)**
  - **Continues** to execute the loop body **AS LONG AS** the test condition is **FALSE** (returns a non-zero exit status).

○ It stops when the condition becomes **TRUE** (returns an exit status of 0).

**7. Write a countdown timer script using a while loop.**

This script starts at a user-defined number and counts down to 0, pausing for one second between each number using the sleep command.

```
#!/bin/bash
 read -p "Enter the number of seconds to countdown from: "
counter
# Input validation (optional but good practice)
if ! [[ "$counter" =~ ^[0-9]+$ ]] || [ "$counter" -lt 0 ];
then     echo "Error: Please enter a non-negative integer."
exit 1 fi  echo "--- Starting Countdown ---"
# The while loop continues as long as 'counter' is greater than 0
while [ $counter -gt 0 ]; do
    echo "Time remaining: **$counter**"

    # Pause for 1 second
sleep 1
    # Decrement the counter (using the arithmetic expression
'let')     let counter=counter-1 done  echo "--------------------
------" echo "**Time's up!**"
```

**8. How do you use break and continue statements in loops? Give examples.**

These are control statements used to alter the normal execution flow inside any loop (for, while, until).

**1. break Statement**

- **Purpose**: Immediately **exits** the innermost loop entirely.
- **Effect:** Execution jumps to the command immediately following the loop's termination. **Example: Stopping a search once an item is found.**

```
for i in {1..10}; do
if [ $i -eq 5 ]; then
       echo "Found the number 5. **Breaking** the
loop."         break # Exits the loop completely     fi
echo "Current number: $i" done
echo "Loop finished (or broken)."
# Output: Current number: 1, 2, 3, 4, Found the number 5. Breaking
the loop. Loop finished (or broken).
```

**2. continue Statement**

- **Purpose:** Skips the **rest of the current iteration** of the loop and moves immediately to the next iteration.
- **Effect:** The loop's condition is re-evaluated, and the loop starts again from the top of the body (if the condition allows).

**Example: Skipping even numbers (only processing odd numbers).**

```
for i in {1..7}; do     if
(( i % 2 == 0 )); then
        echo "Number $i is even. **Continuing** to the next
iteration."
        continue # Skips 'Processing...' for this
iteration     fi     echo "Processing odd number: $i"
done
echo "Loop completed."
# Output: Processing odd number: 1, Number 2 is even.
Continuing..., Processing odd number: 3, ...
```

**9. Write a script to check if a file exists or not using the if and else loop.**

You use the **file test operator -f** inside the if statement's conditional block ([ ... ] or [[ ... ]]).

```
#!/bin/bash

read -p "Enter the file name to check: "
filename
# The '-f' operator returns true if the file exists AND is a regular
file.
if [ -f "$filename" ]; then     echo
"File '$filename' **exists**."
echo "It is a regular file." else
    # This includes cases where the path doesn't exist, or it's a
directory/link.
    echo "File '$filename' **does NOT exist** (or is not a regular
file)." fi
```

**10. Write a script to calculate factorial of a number using for loop.**

The factorial of a non-negative integer n, denoted by n!, is the product of all positive integers less than or equal to n. (0! = 1). This script uses a simple arithmetic for loop structure.

```
#!/bin/bash

# Set initial factorial value to 1
factorial=1
```

```
 read -p "Enter a positive integer: "
num
# Input validation if [
"$num" -lt 0 ]; then
    echo "Factorial is not defined for negative numbers."
exit 1
elif [ "$num" -eq 0 ]; then     echo
"The factorial of 0 is **1**."
exit 0 fi
# The arithmetic for loop structure: for ((INIT; CONDITION; STEP))
# Loop from 'num' down to 1, multiplying 'factorial' by the
current number
for (( i = 1; i <= num; i++ )); do
    # Use the 'let' or '((...))' for arithmetic
operations     let factorial=factorial*i done  echo "The
factorial of $num is: **$factorial**."
```