Linux Programming: Assignment-9

Samriddhi Guha | ENG24CY0157 | 3-C (25)

21. Write a shell script using if…else to check if a number is even or odd. (CO4)

The script uses the *modulo operator (%)* within arithmetic expansion $(( )) to check the remainder when the number is divided by 2. If the remainder is zero, the number is even; otherwise, it's odd.

```
#!/bin/bash
# Script to check if a number is even or odd

# 1. Read the number from the user
echo -n "Enter an integer: "
read NUM

# 2. Use arithmetic expansion to find the remainder (modulo)
# (( NUM % 2 )) calculates the remainder
REMAINDER=$(( NUM % 2 ))

# 3. Use if/else to check the remainder
if [ "$REMAINDER" -eq 0 ]; then
   echo "$NUM is an EVEN number."
else
   echo "$NUM is an ODD number."
fi
```

22. Explain the difference between if and case statements in bash. (CO4)

Both if and case statements are used to control conditional behavior, however, they are evaluated and organized differently in their handling of multiple branches.

if Statement (Conditional Branching):

- Purpose: To execute a command based on the exit status of the test command or expression. This is typically best for assessing complex logical conditions such as greater than or less than, file existence, or combining conditions with && or ||.
- Syntax: This is primarily concerned with sequential testing (if -> elif -> else).
- Example: *if [$A -gt $B]; then…*

case Statement (Pattern Matching):

- Purpose: To execute commands based on a single variable or value matching multiple patterns. This is typically best applied to patterns against a single key value that needs to be matched against a number of pre-defined options (like menu selections or checking input strings).
- Syntax: This is mostly concerned with matching a value against patterns if it matches any pattern (pattern1) and will be done under the case...in...esac conditions.
- Example: *case $CHOICE in "y"|"Y")...esac*

23. Write a script to find the largest of three numbers entered by the user. (CO4)

This script reads three numbers and uses *nested if/elif/else* statements for comparison.

```
#!/bin/bash
# Script to find the largest of three numbers

echo "Enter three numbers:"
echo -n "Number 1: "
read N1
echo -n "Number 2: "
read N2
echo -n "Number 3: "
read N3

# Start comparison: Assume N1 is largest, then check N2 and N3
if [ "$N1" -ge "$N2" ] && [ "$N1" -ge "$N3" ]; then
   LARGEST=$N1
elif [ "$N2" -ge "$N1" ] && [ "$N2" -ge "$N3" ]; then
   LARGEST=$N2
else
   # If N1 and N2 are not the largest, N3 must be
   LARGEST=$N3
fi

echo "-----------------------------------"
echo "The largest number among $N1, $N2, and $N3 is: $LARGEST"
```

24. 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). (CO4)

In Bash, to traverse an array you utilize the *for loop* with the special array expansion of ${array_name[@]} or ${array_name[*]} to iterate through individual elements' value in an array, regardless of the array's index.

Example:

```
#!/bin/bash
# Script to traverse and display array elements

# Array definition
# Note: Commas are treated as part of the string element in Bash
arr=("123," "Abs" "-2.3," "A" "23.56," "0")

echo "--- Traversing Array ---"
echo "Array Elements: ${arr[@]}"
echo "------------------------"
```

```
# Use a for-in loop structure to iterate over the values of all elements
for element in "${arr[@]}"; do
   # Display the current element's value
   echo "Element Value: $element"
done

# Another way: Using a C-style loop for index access (more explicit)
echo "------------------------"
echo "Traversing by Index:"
NUM_ELEMENTS=${#arr[@]}

for (( i=0; i < NUM_ELEMENTS; i++ )); do
   echo "Element at index $i: ${arr[i]}"
done
```

25. Write a shell script to loop through all files in the current directory and display their names. (CO4)

The script uses a simple *for loop* with the wildcard * to match and iterate over all files and directories in the current working directory.

```
#!/bin/bash
# Script to loop through all files and directories in the current directory

echo "--- Files and Directories in $(pwd) ---"
echo "---------------------------------------"

# Use the wildcard * to match all file and directory names
for item in *; do
   # Check if the item is a regular file
   if [ -f "$item" ]; then
      echo "FILE: $item"
   # Check if the item is a directory
   elif [ -d "$item" ]; then
      echo "DIR:  $item"
   else
      # Catch links, devices, etc.
      echo "OTHER: $item"
   fi
done

echo "---------------------------------------"
```

26. What is the difference between while and until loops in bash? (CO4)

The difference between while and until loops lies in their *test condition* and when they choose to execute the loop body.

Feature	while Loop	until Loop
Execution Condition	Executes AS LONG AS the test command/condition is TRUE (exit status).	Executes AS LONG AS the test command/condition is FALSE (non-zero exit status).
Logic	Condition is positive (continue if true).	Condition is negative (continue if false).
Syntax	while condition; do commands; done	until condition; do commands; done

27. Write a countdown timer script using a while loop. (CO4)

This script uses a *while loop and the sleep command* to create a simple countdown timer starting from 10 seconds.

```
#!/bin/bash
# Countdown timer script

COUNT=10

echo "--- Starting Countdown ---"

# Loop continues as long as COUNT is greater than 0
while [ $COUNT -gt 0 ]; do
    echo "Time remaining: $COUNT seconds"

    # Wait for 1 second
    sleep 1

    # Decrement the counter
    COUNT=$(( COUNT - 1 ))
done
```

```
echo "-------------------------"
echo "Countdown finished!"
```

28. How do you use break and continue statements in loops? Give examples. (CO4)

The break and continue are used inside loops to control the flow of execution:

break Statement:

- Purpose: Immediately exits the entire loop. Execution jumps to the command immediately following the done keyword. It's used when a specific condition is met and no further iteration is necessary.
- Example (Exiting on specific number):

```
for i in 1 2 3 4 5; do
    if [ $i -eq 4 ]; then
        echo "Found 4. Breaking the loop."
        break
    fi
    echo "Current number: $i"
done
```

continue Statement:

- Purpose: Skips the rest of the current loop iteration and immediately starts the next iteration. It's used when a specific condition is met, and the current set of commands should be ignored, but the loop needs to proceed.
- Example (Skipping on specific number):

```
for i in 1 2 3 4 5; do
    if [ $i -eq 3 ]; then
        echo "Skipping number 3."
        continue
    fi
    echo "Processing number: $i"
done
```

29. Write a script to check if a file exists or not using the if and else loop. (CO4)

The script uses the *test operator -f* inside an if/else block to determine if a given path corresponds to a regular file.

```
#!/bin/bash
```

```
# Script to check for file existence

FILE_NAME="notes.txt"

echo "Checking for file: $FILE_NAME"

# The -f operator tests if the file exists AND is a regular file
if [ -f "$FILE_NAME" ]; then
    echo "SUCCESS: The file '$FILE_NAME' exists in the current directory."
    echo "Permissions: $(ls -l "$FILE_NAME" | awk '{print $1}')"
else
    echo "FAILURE: The file '$FILE_NAME' does NOT exist."
fi
```

30. Write a script to calculate factorial of a number using for loop. (CO4)

The script calculates the factorial (e.g.,) using a *for loop* with C-style syntax.

```
#!/bin/bash
# Script to calculate factorial using a for loop

echo -n "Enter a number to calculate its factorial: "
read NUM

# Input validation
if [ "$NUM" -lt 0 ]; then
    echo "Error: Factorial is not defined for negative numbers."
    exit 1
fi

if [ "$NUM" -eq 0 ]; then
    echo "The factorial of 0 is 1."
    exit 0
fi

FACTORIAL=1

# C-style for loop: i starts at NUM, continues as long as i > 0, decrements i
for (( i=NUM; i > 0; i-- )); do
    # Arithmetic expansion to multiply the factorial by i
    FACTORIAL=$(( FACTORIAL * i ))
done

echo "-------------------------------------------------"
echo "The factorial of $NUM is: $FACTORIAL"
```