LP ASSIGNMENT-9

ENG24CY0159

SHADIK KHAN

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

This script works by using the modulo operator (%), which gives you the remainder of a division. A number is even if the remainder is 0 when divided by 2.

Bash

```
#!/bin/bash


# This script checks if a number is even or odd.


# Prompt the user for a number
read -p "Enter a number: " number


# Check the remainder when divided by 2
# The ((...)) syntax is for arithmetic operations
if (( number % 2 == 0 )); then
  echo "$number is an even number."
else
  echo "$number is an odd number."
fi
```

---

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

Think of it like this: if is for a series of yes/no questions, while case is for a multiple-choice question.

- **if Statement:** This is your general-purpose decision maker. You use it to test conditions that result in true or false. It's perfect for complex logic with different, unrelated checks.
    - **Use it for:** "Is this number greater than 10?", "Does this file exist?", "Is this string empty?"
- **case Statement:** This is a cleaner way to handle a "multiple-choice" scenario. You use it when you have a **single variable** that you need to compare against a **list of possible values (patterns)**. It makes your script much more readable than a long chain of if...elif...elif....
    - **Use it for:** "The user typed a command. Is it 'start', 'stop', 'restart', or something else?"

In short, if you're checking one variable against many possible patterns, use case. For everything else, if is your tool.

---

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

This script reads three numbers and then uses a series of if and elif (else if) statements to compare them and find the largest one.

Bash

```
#!/bin/bash


# Reads three numbers and finds the largest among them.


echo "Enter three numbers, separated by spaces:"
read n1 n2 n3


# Check if n1 is the greatest
# -gt means "greater than", && means "AND"
if [[ $n1 -ge $n2 && $n1 -ge $n3 ]]; then
  largest=$n1
```

```
# If not, check if n2 is the greatest
elif [[ $n2 -ge $n1 && $n2 -ge $n3 ]]; then
  largest=$n2
# Otherwise, n3 must be the greatest
else
  largest=$n3
fi

echo "The largest number is: $largest"
```

---

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

First, a quick correction on the array format in your question: in bash, array elements are separated by **spaces**, not commas. Bash also treats all elements as strings, so the numbers -2.3 and 23.56 will be handled as simple text.

Here's how you'd declare the array and loop through it. The special syntax "${arr[@]}" is the safest way to get all elements, as it correctly handles spaces within elements.

Example Script:

Bash

```
#!/bin/bash

# Correctly declare the array with space-separated elements
arr=(123 "Abs" -2.3 'A' 23.56 0)

echo "--- Traversing the array ---"

# Loop through each item in the array
```

```
for item in "${arr[@]}"
do
  echo "Element: $item"
done
```

---

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

This script uses a for loop with a wildcard * (also called a glob). The shell expands * into a list of all the non-hidden files and directories in the current location.

Bash

```
#!/bin/bash

# This script lists all items in the current directory.

echo "--- Items in $(pwd) ---"

for item in *
do
  # The -f check ensures we only print files, not directories
  if [ -f "$item" ]; then
    echo "File found: $item"
  fi
done
```

I've added an if [-f "$item"] check to make the script more robust, so it only tells you about actual files and ignores any sub-directories.

---

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

They are exact opposites and control a loop based on whether a condition is true or false.

- A **while loop** continues **as long as its condition is TRUE**.
    - Analogy: "Keep stirring **while** the soup is still lumpy." (You stop once the condition becomes false).
- An **until loop** continues **as long as its condition is FALSE**.
    - Analogy: "Keep stirring **until** the soup is smooth." (You stop once the condition becomes true).

You can achieve the same result with both, but one is often more natural to read than the other depending on the logic.

---

27. Write a countdown timer script using a while loop.

This script starts a counter at 10 and loops as long as the number is greater than zero. Inside the loop, it prints the number, pauses for one second using the sleep command, and then subtracts one from the counter.

Bash

```
#!/bin/bash


# A 10-second countdown timer.


count=10


echo "Starting countdown..."


# Loop while the count is greater than 0
while [ $count -gt 0 ]; do
  echo "$count..."
```

```
  sleep 1
  count=$((count - 1))
done


echo "Liftoff! 🚀 "
```

---

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

break and continue are used to control the flow of a loop from within.

- **break is the emergency brake.** It **stops the entire loop immediately** and the script continues with the code after the loop.
    - **Example:** Find the first .log file and then stop searching.

Bash

```
for file in *; do
  if [[ "$file" == *.log ]]; then
    echo "Found a log file: $file"
    break # Exit the loop now
  fi
done
```

- **continue is the "skip" button.** It **skips the rest of the current iteration** and jumps straight to the next one.
    - **Example:** Print all files *except* .txt files.

Bash

```
for file in *; do
  if [[ "$file" == *.txt ]]; then
    continue # Skip this item and go to the next one
  fi
  echo "Processing file: $file"
```

done

---

29. Write a script to check if a file exists or not using the if and else statement.

First, a small clarification: if...else is a **conditional statement** for making a decision, not a loop.

This script uses the -f test operator, which checks if a given path exists and is a regular **f**ile.

Bash

```
#!/bin/bash


# This script checks if a file exists.


# We'll check for the file provided as the first argument to the script
FILENAME=$1


if [ -f "$FILENAME" ]; then
  echo "SUCCESS: The file '$FILENAME' exists."
else
  echo "FAILURE: The file '$FILENAME' does not exist."
fi
```

How to run it:

Bash

```
./check_file.sh /etc/hosts  # This will likely succeed
./check_file.sh non_existent_file.txt # This will likely fail
```

---

30. Write a script to calculate the factorial of a number using a for loop.

The factorial of a number N (written as N!) is the product of all positive integers up to N (e.g., 5! = 1 * 2 * 3 * 4 * 5). This script uses a C-style for loop, which is very convenient for this kind of counting.

Bash

```
#!/bin/bash

# This script calculates the factorial of a number.

read -p "Enter a non-negative number: " number

# Initialize factorial to 1
factorial=1

# Handle the edge case of 0! = 1
if [ $number -eq 0 ]; then
  echo "The factorial of 0 is 1."
  exit 0
fi

# Loop from 1 up to the entered number
for (( i=1; i<=number; i++ )); do
  factorial=$((factorial * i))
done

echo "The factorial of $number is: $factorial"
```
