Conditional statements in shell scripting allow you to execute certain commands or code blocks based on the evaluation of conditions. These conditions can be based on comparisons, file tests, or any other logical checks.

1. Basic `if` Statement

The `if` statement is the most basic conditional statement. It evaluates a condition and, if the condition is true, executes the code block.

Syntax:

```
if [ condition ]; then
    # Code to execute if the condition is true
fi
```

Example:

```
#!/bin/bash

# Check if a number is positive
number=10

if [ $number -gt 0 ]; then
    echo "The number is positive."
fi
```

- `[ $number -gt 0 ]` checks if `number` is greater than 0 (`-gt` stands for "greater than").

2. `if-else` Statement

The `if-else` statement allows you to specify an alternative block of code to execute if the condition is false.

Syntax:

```
if [ condition ]; then
```

```
    # Code to execute if the condition is true
else
    # Code to execute if the condition is false
fi
```

Example:

```
#!/bin/bash

# Check if a number is positive or negative
number=-5

if [ $number -gt 0 ]; then
    echo "The number is positive."
else
    echo "The number is negative."
fi
```

3. `if-elif-else` Statement

The `if-elif-else` statement is used when you have multiple conditions to check. The `elif` keyword stands for "else if."

Syntax:

```
if [ condition1 ]; then
    # Code to execute if condition1 is true
elif [ condition2 ]; then
    # Code to execute if condition2 is true
else
    # Code to execute if neither condition1 nor condition2 is true
fi
```

Example:

```
#!/bin/bash

# Check if a number is positive, negative, or zero
number=0

if [ $number -gt 0 ]; then
    echo "The number is positive."
elif [ $number -lt 0 ]; then
    echo "The number is negative."
else
    echo "The number is zero."
fi
```

4. Comparison Operators

Shell scripting provides several comparison operators to use within conditional statements:

- **Integer Comparison:**
  - `-eq`: Equal to
  - `-ne`: Not equal to
  - `-lt`: Less than
  - `-le`: Less than or equal to
  - `-gt`: Greater than
  - `-ge`: Greater than or equal to

Example:

```
if [ $a -eq $b ]; then
    echo "a is equal to b"
fi
```

-
- **String Comparison:**
  - `=`: Equal to
  - `!=`: Not equal to
  - `-z`: String is null (zero length)
  - `-n`: String is not null

Example:

```
if [ "$str1" = "$str2" ]; then
    echo "Strings are equal"
fi
```

-

5. File Test Operators

Shell scripting also includes operators for testing file properties:

- **File Existence:**
  - `-e file`: File exists
  - `-f file`: File exists and is a regular file
  - `-d file`: Directory exists
  - `-r file`: File is readable
  - `-w file`: File is writable
  - `-x file`: File is executable

Example:

```
if [ -e /path/to/file ]; then
    echo "File exists."
else
    echo "File does not exist."
fi
```

-

6. Logical Operators

You can combine multiple conditions using logical operators:

- **AND (`&&`):** Both conditions must be true.
- **OR (`||`):** At least one condition must be true.
- **NOT (`!`):** Inverts the condition.

Example:

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

```
# Check if a file is readable and writable
file="/path/to/file"

if [ -r $file ] && [ -w $file ]; then
    echo "The file is readable and writable."
else
    echo "The file is not readable and writable."
fi
```

7. Case Statement

The `case` statement is used for matching a variable against a pattern. It is especially useful when you have multiple patterns to check.

Syntax:

```
case variable in
    pattern1)
        # Code to execute for pattern1
        ;;
    pattern2)
        # Code to execute for pattern2
        ;;
    *)
        # Default case (optional)
        ;;
esac
```

Example:

```
#!/bin/bash

# Simple menu-driven script
echo "Enter your choice (start/stop/restart):"
read choice

case $choice in
```

```
    start)
        echo "Starting the service..."
        ;;
    stop)
        echo "Stopping the service..."
        ;;
    restart)
        echo "Restarting the service..."
        ;;
    *)
        echo "Invalid choice."
        ;;
esac
```

- This script reads a user's choice and matches it against "start", "stop", and "restart".

8. Nested `if` Statements

You can also nest `if` statements inside other `if` statements to create more complex logic.

Example:

```
#!/bin/bash

# Check if a file exists and is readable
file="/path/to/file"

if [ -e $file ]; then
    if [ -r $file ]; then
        echo "The file exists and is readable."
    else
        echo "The file exists but is not readable."
    fi
else
    echo "The file does not exist."
fi
```

9. Using `test` Command

The `test` command (or `[ condition ]`) is used to evaluate expressions. You can use `test` explicitly or implicitly as shown in the examples above.

Example:

```
#!/bin/bash

# Using test command
number=10

if test $number -gt 0; then
    echo "The number is positive."
fi
```

Summary

- **`if`**: Executes a block of code if the condition is true.
- **`if-else`**: Provides an alternative block of code to execute if the condition is false.
- **`if-elif-else`**: Allows multiple conditions to be checked in sequence.
- **Comparison Operators**: Used for numeric and string comparisons.
- **File Test Operators**: Used to test file attributes like existence, readability, and writability.
- **Logical Operators**: Combine multiple conditions using `&&` (AND), `||` (OR), and `!` (NOT).
- **`case` Statement**: Matches a variable against multiple patterns, often used for menu-driven scripts.

Conditional statements are essential in shell scripting, enabling decision-making based on various criteria, which is crucial for automating tasks effectively.

4o