**User interaction**

**When you want to interact with a user through a shell script, you can use commands like read**

**read var\_name**

**read -p "msg" var\_name**

**#!/bin/bash**

**# Prompt the user for their name**

**echo "Enter your name:"**

**read name**

**# Greet the user**

**echo "Hello, $name! Welcome to the shell scripting tutorial."**

**The read command in shell scripting has several options, one of which is the -p option. The -p option allows you to prompt the user with a message directly without using a separate echo command.**

**#!/bin/bash**

**# Prompt the user for their name using the -p option**

**read -p "Enter your name: " name**

**# Greet the user**

**echo "Hello, $name! Welcome to the shell scripting tutorial."**

**The -r option with the read command in shell scripting is used to prevent backslashes from being interpreted as escape characters. This means that when the -r option is used, the read command will treat backslashes as literal characters.**

**#!/bin/bash**

**# Prompt the user for input without the -r option**

**read -p "Enter a string (with backslashes if needed): " input**

**# Display the entered string**

**echo "You entered: $input"**

**If the user enters Hello\World**

**You entered: HelloWorld**

**#!/bin/bash**

**# Prompt the user for input with the -r option**

**read -r -p "Enter a string (with backslashes if needed): " input**

**# Display the entered string**

**echo "You entered: $input"**

**If the user enters Hello\World**

**You entered: Hello\World**

**Arithmetic Operators**

**In shell scripting, arithmetic operations can be performed using a variety of methods. The most common methods include using expr, $((...)), and let. Here’s an overview of how to use these methods with arithmetic operators:**

### **Arithmetic Operators**

* **Addition (+): Adds two numbers.**
* **Subtraction (-): Subtracts the second number from the first.**
* **Multiplication (\*): Multiplies two numbers.**
* **Division (/): Divides the first number by the second (results in an integer, discarding any remainder).**
* **Modulus (%): Returns the remainder of the division.**
* **Exponentiation (\*\*): Raises the first number to the power of the second number (only supported in some shells like bash).**

### **Method 1: Using $((...)) (Arithmetic Expansion)**

**#!/bin/bash**

**# Define variables**

**a=10**

**b=5**

**# Display the results**

**echo "Sum: $(($a+$b))"**

**echo "Difference: $(($a-$b))"**

**echo "Product: $(($a\*$b))"**

**echo "Quotient: $(($a/$b))"**

**echo "Remainder: $(($a%$b))"**

**echo "Power: $(($a\*\*$b))"**

### **Method 2: Using expr (External Command)**

**#!/bin/bash**

**# Define variables**

**a=10**

**b=5**

**# Perform arithmetic operations using expr**

**sum=$(expr $a + $b)**

**difference=$(expr $a - $b)**

**product=$(expr $a \\* $b)**

**quotient=$(expr $a / $b)**

**remainder=$(expr $a % $b)**

**# Display the results**

**echo "Sum: $sum"**

**echo "Difference: $difference"**

**echo "Product: $product"**

**echo "Quotient: $quotient"**

**echo "Remainder: $remainder"**

**Note: When using expr, you must escape the multiplication operator (\*) with a backslash (\) to avoid shell interpretation.**

### **Method 3: Using let (Bash Built-in Command)**

**#!/bin/bash**

**# Define variables**

**a=10**

**b=5**

**# Perform arithmetic operations using let**

**let sum=a+b**

**let difference=a-b**

**let product=a\*b**

**let quotient=a/b**

**let remainder=a%b**

**let power=a\*\*b**

**# Display the results**

**echo "Sum: $sum"**

**echo "Difference: $difference"**

**echo "Product: $product"**

**echo "Quotient: $quotient"**

**echo "Remainder: $remainder"**

**echo "Power: $power"**

### **Explanation:**

* **$((...)): In this method, arithmetic operations are straightforward. The variables a and b are used directly within the arithmetic expression.**
* **expr: You need to use $(expr ...) to evaluate the expression and assign it to a variable. Notice that expr requires the multiplication operator (\*) to be escaped with a backslash (\).**
* **let: This command allows arithmetic directly on variables without needing $((...)). The arithmetic operation is evaluated and assigned directly to the variable.**

**Relational Operators**

**Relational operators in shell scripting are used to compare two values and return a boolean result (true or false). These operators are commonly used in conditional statements like if, while, and until.**

**Example :-**

**#!/bin/bash**

**# Numeric Comparison**

**a=15 b=20**

**# Numeric comparison using test command**

**test $a -lt $b**

**result=$?**

**echo “$result”**

#### **Types of Relational Operators**

**Relational operators in shell scripting can be broadly categorized into two types:**

1. **Letter-based Relational Operators: These are mainly used in the traditional test command ([...]).**
2. **Symbolic Relational Operators: These are used with arithmetic expansion ($((...))) and the [[...]] conditional expression syntax in modern shell scripts.**

#### **1. Letter-based Relational Operators**

**These operators are primarily used in the test command ([...]) and are ideal for numeric comparisons. They are supported in most Unix-like shells, making them highly portable.**

| **Operator** | **Description** | **Example** | **Explanation** |
| --- | --- | --- | --- |
| **-eq** | **Equal to** | **[ $a -eq $b ]** | **True if a is equal to b** |
| **-ne** | **Not equal to** | **[ $a -ne $b ]** | **True if a is not equal to b** |
| **-lt** | **Less than** | **[ $a -lt $b ]** | **True if a is less than b** |
| **-le** | **Less than or equal to** | **[ $a -le $b ]** | **True if a is less than or equal to b** |
| **-gt** | **Greater than** | **[ $a -gt $b ]** | **True if a is greater than b** |
| **-ge** | **Greater than or equal to** | **[ $a -ge $b ]** | **True if a is greater than or equal to b** |

**Example Usage:**

**#!/bin/bash**

**a=10**

**b=20**

**if [ $a -lt $b ]; then**

**echo "$a is less than $b"**

**else**

**echo "$a is not less than $b"**

**fi**

#### **2. Symbolic Relational Operators**

**Symbolic operators are more intuitive and resemble operators used in many programming languages. They are used with the [[...]] syntax for string comparisons and with $((...)) for numeric comparisons in modern Bash scripts.**

| **Operator** | **Description** | **Example** | **Explanation** |
| --- | --- | --- | --- |
| **==** | **Equal to** | **[[ $a == $b ]]** | **True if a is equal to b (string or numeric)** |
| **!=** | **Not equal to** | **[[ $a != $b ]]** | **True if a is not equal to b (string or numeric)** |
| **<** | **Less than** | **[[ $a < $b ]] (string)** | **True if a is less than b (lexicographically)** |
| **<** | **Less than** | **(( a < b )) (numeric)** | **True if a is less than b** |
| **<=** | **Less than or equal to** | **(( a <= b ))** | **True if a is less than or equal to b** |
| **>** | **Greater than** | **[[ $a > $b ]] (string)** | **True if a is greater than b (lexicographically)** |
| **>** | **Greater than** | **(( a > b )) (numeric)** | **True if a is greater than b** |
| **>=** | **Greater than or equal to** | **(( a >= b ))** | **True if a is greater than or equal to b** |

**Example Usage:**

**#!/bin/bash**

**a=10**

**b=20**

**if [[ $a == $b ]]; then**

**echo "$a is equal to $b"**

**else**

**echo "$a is not equal to $b"**

**fi**

**if (( a < b )); then**

**echo "$a is less than $b"**

**fi**

### **When to Use Letter-based vs. Symbolic Relational Operators**

* **Letter-based Operators (-eq, -ne, etc.):**
  + **Best suited for numeric comparisons in older shells or scripts that need to be highly portable across different Unix-like environments.**
  + **Typically used in scripts that rely on the traditional [...] syntax.**
* **Symbolic Operators (==, !=, <, >, etc.):**
  + **Preferred in modern Bash scripts for both numeric and string comparisons.**
  + **[[...]] syntax is safer for string comparisons, as it handles special characters and patterns more effectively.**
  + **$((...)) is ideal for numeric comparisons, offering a concise and readable syntax.**

**Practical Example: Combined Use of Relational Operators**

**#!/bin/bash**

**a=30**

**b=20**

**str1="apple"**

**str2="banana"**

**# Numeric comparison using symbolic operators**

**if (( a > b )); then**

**echo "$a is greater than $b"**

**fi**

**# String comparison using symbolic operators**

**if [[ $str1 < $str2 ]]; then**

**echo "$str1 comes before $str2 lexicographically"**

**fi**

**# Numeric comparison using letter-based operators**

**if [ $a -ne $b ]; then**

**echo "$a is not equal to $b"**

**fi**

**Note:- While [[ ... ]] can indeed work with letter-based operators for numeric comparisons, it is most commonly used for its enhanced capabilities with symbolic operators and advanced features. For modern Bash scripting, [[ ... ]] is generally preferred for its improved syntax and flexibility, but for legacy scripts and compatibility, [ ... ] remains a reliable choice.**

**Logical Operators**

**Logical operators in shell scripting allow you to combine multiple conditions or to perform logical operations to control the flow of your script.**

**They are also known as boolean operators. These are used to perform logical operations. They are of 3 types:**

* **Logical AND (&&): This is a binary operator, which returns true if both the operands are true otherwise returns false.**
* **Logical OR (||): This is a binary operator, which returns true if either of the operands is true or if both the operands are true. It returns false only if both operands are false.**
* **Not Equal to (!): This is a unary operator which returns true if the operand is false and returns false if the operand is true.**

#### **1. Logical AND**

* **Symbolic Representation: &&**
  + **Definition: Executes the second command only if the first command succeeds (i.e., returns an exit status of 0). Returns true if both operands are true.**

**Usage:  
  
command1 && command2**

**Example:  
  
mkdir new\_directory && cd new\_directory**

* **Letter-based Representation: -a**
  + **Definition: Used within [ ... ] to combine conditions. Returns true if both conditions are true.**

**Usage:  
  
[ CONDITION1 -a CONDITION2 ]**

**Example:  
  
a=10**

**b=20**

**if [ $a -lt $b -a -e "example.txt" ]; then**

**echo "$a is less than $b and file exists."**

**else**

**echo "One or both conditions are false."**

**fi**

#### **2. Logical OR**

* **Symbolic Representation: ||**
  + **Definition: Executes the second command only if the first command fails (i.e., returns a non-zero exit status). Returns true if either operand is true.**

**Usage:  
  
command1 || command2**

**Example:  
  
mkdir new\_directory || echo "Failed to create directory."**

* **Letter-based Representation: -o**
  + **Definition: Used within [ ... ] to combine conditions with OR. Returns true if at least one condition is true.**

**Usage:  
  
[ CONDITION1 -o CONDITION2 ]**

**Example:  
  
file1="example1.txt"**

**file2="example2.txt"**

**if [ -e "$file1" -o -e "$file2" ]; then**

**echo "At least one of the files exists."**

**else**

**echo "Neither file exists."**

**fi**

#### **3. Logical NOT**

* **Symbolic Representation: !**
  + **Definition: Negates the exit status of a command. Returns true if the command fails (i.e., returns a non-zero exit status).**

**Usage:  
  
! command**

**Example:  
  
file="example.txt"**

**if [ ! -e "$file" ]; then**

**echo "$file does not exist."**

**else**

**echo "$file exists."**

* + **fi**

**Here’s a detailed overview of logical operators in shell scripting, including their definitions, usage, and the symbolic and letter-based representations:**

### **Logical Operators in Shell Scripting**

#### **1. Logical AND**

* **Symbolic Representation: &&**
  + **Definition: Executes the second command only if the first command succeeds (i.e., returns an exit status of 0). Returns true if both operands are true.**

**Usage:  
  
command1 && command2**

**Example:  
  
mkdir new\_directory && cd new\_directory**

* **Letter-based Representation: -a**
  + **Definition: Used within [ ... ] to combine conditions. Returns true if both conditions are true.**

**Usage:  
  
[ CONDITION1 -a CONDITION2 ]**

**Example:**

**a=10**

**b=20**

**if [ $a -lt $b -a -e "example.txt" ]; then**

**echo "$a is less than $b and file exists."**

**else**

**echo "One or both conditions are false."**

**fi**

#### **2. Logical OR**

* **Symbolic Representation: ||**
  + **Definition: Executes the second command only if the first command fails (i.e., returns a non-zero exit status). Returns true if either operand is true.**

**Usage:  
  
command1 || command2**

**Example:  
  
mkdir new\_directory || echo "Failed to create directory."**

* **Letter-based Representation: -o**
  + **Definition: Used within [ ... ] to combine conditions with OR. Returns true if at least one condition is true.**

**Usage:  
  
[ CONDITION1 -o CONDITION2 ]**

**Example:  
  
file1="example1.txt"**

**file2="example2.txt"**

**if [ -e "$file1" -o -e "$file2" ]; then**

**echo "At least one of the files exists."**

**else**

**echo "Neither file exists."**

**fi**

#### **3. Logical NOT**

* **Symbolic Representation: !**
  + **Definition: Negates the exit status of a command. Returns true if the command fails (i.e., returns a non-zero exit status).**

**Usage:  
  
! command**

**Example:  
  
file="example.txt"**

**if [ ! -e "$file" ]; then**

**echo "$file does not exist."**

**else**

**echo "$file exists."**

**fi**

### **Summary of Logical Operators**

* **&& (Logical AND):**
  + **Function: Returns true if both operands are true.**
  + **Symbolic Usage: command1 && command2**
  + **Letter-based Usage: [ CONDITION1 -a CONDITION2 ]**
* **|| (Logical OR):**
  + **Function: Returns true if at least one operand is true.**
  + **Symbolic Usage: command1 || command2**
  + **Letter-based Usage: [ CONDITION1 -o CONDITION2 ]**
* **! (Logical NOT):**
  + **Function: Returns true if the operand is false.**
  + **Symbolic Usage: ! command**

**Test command**

**The test command evaluates the *Expression* parameter, and if the expression value is True, returns a zero (True) exit value. Otherwise, the test command returns a nonzero (False) exit value. The test command also returns a nonzero exit value if there are no parameters.**

**Or**

**The test command in Unix-like systems is used to evaluate conditional expressions. It's a fundamental command for checking file attributes, string comparisons, and numeric comparisons in shell scripts.**

## **Syntax**

**test** [**Expression**](https://www.ibm.com/docs/en/aix/7.2?topic=t-test-command#test__e342934441mart)

**OR**

**[** [**Expression**](https://www.ibm.com/docs/en/aix/7.2?topic=t-test-command#test__e342934441mart) **]**

**Requirements:**

* **In the second form of the command, the [ ] (brackets) must be surrounded by blank spaces**

| **Item** | **Description** |
| --- | --- |
| **-b *FileName*** | **Returns a True exit value if the specified *FileName* exists and is a block special file.** |
| **-c *FileName*** | **Returns a True exit value if the specified *FileName* exists and is a character special file.** |
| **-d *FileName*** | **Returns a True exit value if the specified *FileName* exists and is a directory.** |
| **-e *FileName*** | **Returns a True exit value if the specified *FileName* exists.** |
| **-f *FileName*** | **Returns a True exit value if the specified *FileName* exists and is a regular file.** |
| **-g *FileName*** | **Returns a True exit value if the specified *FileName* exists and its Set Group ID bit is set.** |
| **-h *FileName*** | **Returns a True exit value if the specified *FileName* exists and is a symbolic link.** |
| **-k *FileName*** | **Returns a True exit value if the specified *FileName* exists and its sticky bit is set.** |
| **-L *FileName*** | **Returns a True exit value if the specified *FileName* exists and is a symbolic link.** |
| **-n *String1*** | **Returns a True exit value if the length of the *String1* variable is nonzero.** |
| **-p *FileName*** | **Returns a True exit value if the specified *FileName* exists and is a named pipe (FIFO).** |
| **-r *FileName*** | **Returns a True exit value if the specified *FileName* exists and is readable by the current process.** |
| **-s *FileName*** | **Returns a True exit value if the specified *FileName* exists and has a size greater than 0.** |
| **-t *FileDescriptor*** | **Returns a True exit value if the file with a file descriptor number of *FileDescriptor* is open and associated with a terminal.** |
| **-u *FileName*** | **Returns a True exit value if the specified *FileName* exists and its Set User ID bit is set.** |
| **-w *FileName*** | **Returns a True exit value if the specified *FileName* exists and the write flag is on. However, the *FileName*will not be writable on a read-only file system even if test indicates true.** |
| **-x *FileName*** | **Returns a True exit value if the specified *FileName* exists and the execute flag is on. If the specified file exists and is a directory, the True exit value indicates that the current process has permission to search in the directory.** |
| **-z *String1*** | **Returns a True exit value if the length of the *String1* variable is 0 (zero).** |
| ***String1*= *String2*** | **Returns a True exit value if the *String1* and *String2* variables are identical.** |
| ***String1*!=*String2*** | **Returns a True exit value if the *String1* and *String2* variables are not identical.** |
| ***String1*** | **Returns a True exit value if the *String1* variable is not a null string.** |
| ***Integer1* -eq *Integer2*** | **Returns a True exit value if the *Integer1* and *Integer2* variables are algebraically equal. Any of the comparisons -ne, -gt, -ge, -lt, and -le can be used in place of -eq.** |
| ***file1* -nt *file2*** | **True if *file1* is newer than *file2*.** |
| ***file1* -ot *file2*** | **True if *file1* is older than *file2*.** |
| ***file1* -ef *file2*** | **True if *file1* is another name for *file2*.** |

### **Types of Comparisons with test**

1. **File Attributes**
   * **-e FILE: True if FILE exists.**
   * **-f FILE: True if FILE is a regular file.**
   * **-d FILE: True if FILE is a directory.**
   * **-r FILE: True if FILE is readable.**
   * **-w FILE: True if FILE is writable.**

**if [ -f "myfile.txt" ]; then**

**echo "myfile.txt exists and is a regular file."**

**else**

**echo "myfile.txt does not exist or is not a regular file."**

**fi**

**String Comparisons**

* **-z STRING: True if STRING is empty.**
* **-n STRING: True if STRING is not empty.**
* **STRING1 = STRING2: True if STRING1 is equal to STRING2.**

**Example: Checking if a String is Empty**

**#!/bin/bash**

**str=""**

**# Check if string is empty**

**if [ -z "$str" ]; then**

**echo "The string is empty."**

**else**

**echo "The string is not empty."**

**fi**

**Example: Checking if a String is Not Empty**

**#!/bin/bash**

**str="data"**

**# Check if string is not empty**

**if [ -n "$str" ]; then**

**echo "The string is not empty."**

**else**

**echo "The string is empty."**

**fi**

**Example: Checking if a File Exists**

**#!/bin/bash**

**file="example.txt"**

**# Check if file exists**

**if [ -e "$file" ]; then**

**echo "$file exists."**

**else**

**echo "$file does not exist."**

**fi**

**Example: Checking if a File is a Directory**

**#!/bin/bash**

**file="example\_directory"**

**# Check if file is a directory**

**if [ -d "$file" ]; then**

**echo "$file is a directory."**

**else**

**echo "$file is not a directory."**

**fi**

**Example: Checking if a File is Readable**

**#!/bin/bash**

**file="example.txt"**

**# Check if file is readable**

**if [ -r "$file" ]; then**

**echo "$file is readable."**

**else**

**echo "$file is not readable."**

**fi**

**Example: Checking if a File is Writable**

**#!/bin/bash**

**file="example.txt"**

**# Check if file is writable**

**if [ -w "$file" ]; then**

**echo "$file is writable."**

**else**

**echo "$file is not writable."**

**fi**

**Example: Checking if a File is Executable**

**#!/bin/bash**

**file="example\_script.sh"**

**# Check if file is executable**

**if [ -x "$file" ]; then**

**echo "$file is executable."**

**else**

**echo "$file is not executable."**

**fi**

**Conditional Statements**

1. **If-else**

**The if condition in shell scripting is used to execute commands based on whether a specified condition is true or false. It allows for branching in your script, enabling you to handle different scenarios based on dynamic input or state.**

### **Basic Syntax**

### **if [ CONDITION ]; then**

**# Commands to execute if CONDITION is true**

**else**

**# Commands to execute if CONDITION is false**

**fi**

### **Examples of if Conditions in Shell Script**

#### **1. Numeric Comparison**

**Example: Checking if a Number is Greater Than Another**

**#!/bin/bash**

**a=15**

**b=10**

**if [ $a -gt $b ]; then**

**echo "$a is greater than $b"**

**else**

**echo "$a is not greater than $b"**

**fi**

1. **If-elif-else**

**Syntax:**

**if [ CONDITION1 ]; then**

**# Commands to execute if CONDITION1 is true**

**elif [ CONDITION2 ]; then**

**# Commands to execute if CONDITION2 is true**

**elif [ CONDITION3 ]; then**

**# Commands to execute if CONDITION3 is true**

**else**

**# Commands to execute if neither CONDITION1 ,CONDITION2 nor CONDITION3 are true**

**fi**

**Using elif for Multiple Conditions**

**#!/bin/bash**

**num=15**

**if [ $num -lt 10 ]; then**

**echo "$num is less than 10."**

**elif [ $num -eq 15 ]; then**

**echo "$num is equal to 15."**

**else**

**echo "$num is greater than 10 and not equal to 15."**

**fi**

1. **case Statement**

**The case statement is used for multi-way branching, similar to switch-case statements in other programming languages. It matches a variable against several patterns.**

#### **Syntax:**

**case $variable in**

**pattern1)**

**# Commands to execute if $variable matches pattern1**

**;;**

**pattern2)**

**# Commands to execute if $variable matches pattern2**

**;;**

**\*)**

**# Commands to execute if no patterns match**

**;;**

**esac**

**#!/bin/bash**

**echo "Please select an option:"**

**echo "1. Show current date"**

**echo "2. Show current directory"**

**echo "3. Exit"**

**read -p "Enter your choice [1-3]: " choice**

**case $choice in**

**1)**

**echo "Current date and time:"**

**date**

**;;**

**2)**

**echo "Current directory:"**

**pwd**

**;;**

**3)**

**echo "Exiting. Goodbye!"**

**exit 0**

**;;**

**\*)**

**echo "Invalid choice. Please select a number between 1 and 3."**

**;;**

**esac**

### **Explanation:**

1. **Prompt User:**
   * **echo commands are used to display the menu options to the user.**
   * **read -p is used to get the user’s choice.**
2. **case Statement:**
   * **case $choice in begins the case statement, which evaluates the value of the variable choice.**
   * **Each pattern (1, 2, 3, \*) corresponds to a different action:**
     + **1): Executes if the user enters 1. It displays the current date and time using the date command.**
     + **2): Executes if the user enters 2. It shows the current directory using the pwd command.**
     + **3): Executes if the user enters 3. It exits the script with a message.**
     + **\*): The default case, executed if the user enters anything other than 1, 2, or 3. It shows an error message.**
3. **esac:**
   * **Ends the case statement.**