

How to Use the If else Statement in Bash

Learn how to use if else statements in Bash scripting to create conditional logic in your shell scripts. Master syntax, examples, and best practices.

Contents

01	Introduction	3
02	If Statement	3
03	If-else Syntax in Bash	4
04	Create If-else Statements in Bash Shell	5
05	Use If-else Statements with Variables	7
06	Use If-elif-else Statements	8
07	Use Multiple Conditions in If-else Statements in Bash	9
08	Nested If-else Statements in Bash	10
09	Conclusion	11

Introduction

The `if-else` statement in Bash is a conditional statement that executes specific commands depending on the results of an evaluation. The statement evaluates expressions to check whether a condition returns true or false. The `if` statement runs when a condition returns true and the `else` statement runs when the condition returns false.

This article explains how to use the `if-else` statement in Bash to test conditions and execute specific commands when the result is true or false.

If Statement

The `if` statement evaluates the result of a condition and executes the respective commands when the condition evaluates to true. The statement has the following components:

```
if condition
then
  # command
fi
```

In the above code:

- `if`: Defines the condition that you want to evaluate.
- `then`: Defines the commands to execute if the condition returns true.
- `fi`: Defines the end of the statement.

Example:

BASH

```
#!/bin/bash

if [ 1 -eq 1 ];
then
```

```
    echo "1 is equal to 1"  
fi
```

Output:

```
1 is equal to 1
```

The above script evaluates the equality of numeric values and returns the result `1 is equal to 1` if the condition returns true.

If-else Syntax in Bash

The `If-else` statement evaluates a condition as true or false and executes the `if` part when true and the `else` part when false. The statement executes alternative commands if a condition does not match the `if` block. The following is the `if-else` statement syntax.

```
if condition  
then  
    # command  
else  
    # command  
fi
```

For example:

```
BASH
```

```
#!/bin/bash  
  
if [ 1 -eq 1 ];  
then  
    echo "1 is equal to 1"  
  
else  
    echo "The numbers are not equal".  
fi
```

Output:

```
1 is equal to 1
```

In the above script, the `if-else` statement executes different commands depending on whether the condition is true or false. The `else` block executes alternative commands if the condition is false, while the `if` block executes commands if the condition is true.

Create If-else Statements in Bash Shell

`If-else` statements use conditional expressions to evaluate the result of a condition. You can replace the test operator with `[]` or `[[` to evaluate a condition and execute specific commands. Follow the steps below to create a bash shell `If else` statement to test a condition and execute specific commands.

1. Create an `If-else` statement to test whether a number is greater than or equal to another number.

```
BASH
#!/bin/bash

if [ 10 -gt 5 ];
then
    echo "The number is greater than 5"

else
    echo "The number is less than 5".
fi
```

Output:

```
The number is greater than 5
```

2. Create an `If-else` statement that checks if a `data.txt` file exists and creates it if it does not exist.

```
BASH
```

```
#!/bin/bash

if [ -f data.txt ];
then

    echo "File exists"

else
    touch data.txt
    echo "New file created"
fi
```

Output:

```
New file created
```

3. Create another `If-else` statement that compares the values of two strings.

```
BASH
```

```
#!/bin/bash

user="admin"

administrator="admin"

if [[ "$user" == "$administrator" ]];
then
    echo "The strings match"
else
    echo "The strings do not match"
fi
```

Output:

```
The strings match
```

Use If-else Statements with Variables

Variables store data and values in Bash for flexible operations. You can combine `If-else` statements with variables to evaluate conditions and manipulate variable data based on different results.

1. Create a new `auth_users` variable and set the value to `admin`. Then, add a new `read` statement to capture a user's input to a new `user` variable. Declare an `If-else` statement that compares the `user` input variable to `auth_users` to evaluate whether the condition is true or false and run specific commands.

```
BASH

#!/bin/bash

auth_users="admin"

read -p "What's your username? " user

if [[ "${user}" == "${auth_users}" ]];
then
    echo
    "You are an administrative user authorized to manage this
system"
else
    echo
    "You are a standard user, you can run basic commands on the
system"
fi
```

Output:

```
What's your username? admin
You are an administrative user authorized to manage this system
```

The above script runs the `If-else` statement and displays the `You are an administrative user authorized to manage this system` message if the condition

returns true. If the condition is false, the script displays `You are a standard user, you can run basic commands on the system` message.

Use If-elif-else Statements

If-else-if (`If-elif-else`) is a conditional structure that compares multiple conditions to filter and execute matching commands. The following is the `If-elif-else` command syntax.

```
if condition
then
    # Commands
elif condition
then
    # Commands
else
    echo # Commands
fi
```

Change the previous `if-else` statement to include a new `elif` statement and output another result if the user input is empty.

BASH

```
#!/bin/bash

auth_users="admin"

read -p "What's your username? " user

if [[ "${user}" == "${auth_users}" ]];
then
    echo "You are an administrative user authorized to manage this system"
elif [[ -z "$user" ]]
then
    echo "Please enter a username"
    read user
    echo "Hello $user, Greetings from Vultr"
else
    echo "You are a standard user, you can run basic commands on
```

```
the system"  
fi
```

Output:

```
What's your username?  
Please enter a username  
example  
Hello example, Greetings from Vultr
```

The above script runs the `if` block when the condition is true. If the user input is empty, the `elif` block queries for a new variable value while `else` displays a `You are a standard user, you can run basic commands on the system` message when the condition is false.

Use Multiple Conditions in If-else Statements in Bash

You can use multiple conditions in bash If-else statements using logical operators such as `AND` and `OR` to evaluate two or more commands in one condition. This approach is important when the second command depends on the success or failure of the first command.

For example:

```
BASH  
  
#!/bin/bash  
  
if [ 10 -gt 5 ] && [[ 5 -lt 8 ]];  
then  
    echo "The number is greater than 5"  
  
else  
    echo "The number is less than 5"  
fi
```

Output:

```
The number is greater than 5
```

In the above script, the `AND` operator (`&&`) tests if the first command is successful before executing the second command.

Nested If-else Statements in Bash

Nested if-else statements allow you to place one if-else statement inside another. This structure is useful when evaluating multiple conditions to meet a specific result to allow further decision-making.

BASH

```
#!/bin/bash

if condition
then
    # Commands
else
    # Commands
if condition
then
    # Commands
else
    # Commands
fi
fi
```

For example:

BASH

```
#!/bin/bash

if [ 10 -gt 5 ] && [[ 5 -lt 8 ]];
then
    echo "The number is greater than 5"
else
    echo "The number is less than 5"
if [[ 1 -eq 1 ]];
```

```
then
    echo "The second number is equal to 1"
else
    echo "The second number is not equal to 1"
fi
fi
```

Output:

```
The number is greater than 5
```

The above script uses a nested `If-else` statement in which the inner condition executes only if the outer condition triggers the `else` block.

Conclusion

You have used the `If-else` statement in Bash to create and test conditions. `If-else` is an important conditional statement that evaluates conditions as `true` or `false` before executing a set of instructions. You can also use `If-else` with other Bash components, such as loops to create advanced scripts.



VULTR

