Conditions/If else statements Scripts, Case Statements Script

One of the most important parts of conditional programming is the if-else statements. An if-else statement allows you to execute iterative conditional statements in your code. We use if-else in shell scripts when we wish to evaluate a condition, then decide to execute one set between two or more sets of statements using the result. This essentially allows us to choose a response to the result that our conditional expression evaluates.

How do if-else in-shell scripts work?

Now we know what is an if-else function and why is it important for any programmer, regardless of their domain. To understand if-else in shell scripts, we need to break down the working of the conditional function. Let us have a look at the syntax of the if-else condition block.

```
if [condition]
then
  statement1
else
  statement2
fi
```

Here we have four keywords, namely *if, then, else* , and *fi.*

1. The keyword *if* is followed by a *condition*.
2. This *condition* is evaluated to decide which statement will be executed by the processor.
3. If the *condition* evaluates to TRUE, the processor will execute the statement(s) followed by the keyword *then*. In the syntax, it is mentioned as *statement 1*.
4. In a case where the *condition* evaluates to FALSE, the processor will execute the statement(s) followed by the keyword *else*. This is denoted as *statement2* in the function syntax.

An important thing to keep in mind is that, like C programming, shell scripting is case-sensitive. Hence, you need to be careful while using the keywords in your code.

How to use if-else in a shell script

It is easy to see the syntax of a function and believe you know how to use it. But it is always a better choice to understand a function through examples because they help you understand the role those different aspects of a function play. Here are some useful examples of if-else in shell scripts to give you a better idea of how to use this tool.

Command	Description
&&	Logical AND
$0	Argument 0 i.e. the command that's used to run the script
$1	First argument (change number to access further arguments)
-eq	Equality check
-ne	Inequality check
-lt	Less Than
-le	Less Than or Equal
-gt	Greater Than
-ge	Greater Than or Equal

1. Using if-else to check whether two numbers are equal

When trying to understand the working of a function like if-else in a shell script, it is good to start things simply. Here, we initialize two variables a and b, then use the if-else function to check if the two variables are equal. The bash script should look as follows for this task.

```
#!/bin/bash
m=1
n=2
if [ $n -eq $m ]
then
        echo "Both variables are the same"
else
        echo "Both variables are different"
fi
fi
```

Output:

```
Both variables are different
```

2. Using if-else to compare two values

The more common use of if-else in shell scripts is for comparing two values. Comparing a variable against another variable or a fixed value helps is used in a variety of cases by all sorts of programmers. For the sake of this example, we will be initializing two variables and using the if-else function to find the variable which is greater than the other.

```
#!/bin/bash
a=2
b=7
if [ $a -ge $b ]
then
  echo "The variable 'a' is greater than the variable 'b'."
else
  echo "The variable 'b' is greater than the variable 'a'."
fi
```

Output:

```
The variable 'b' is greater than the variable 'a'.
```

3. Using if-else to check whether a number is even

Sometimes we come across situations where we need to deal with and differentiate between even and odd numbers. This can be done with if-else in shell scripts if we take the help of the modulus operator. The modulus operator divides a number with a divisor and returns the remainder. As we know all even numbers are a multiple of 2, we can use the following shell script to check for us whether a number is even or odd.

```
#!/bin/bash
n=10
if [ $((n%2))==0 ]
then
  echo "The number is even."
else
  echo "The number is odd."
fi
```

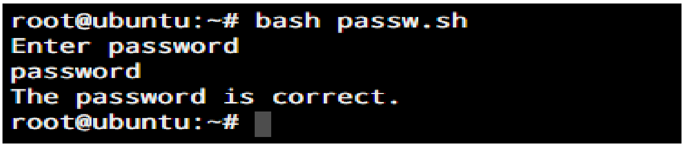
Output:

```
The number is even
```

4. Using if-else as a simple password prompt

The if-else function is known for its versatility and range of applications. In this example, we will use if-else in a shell script to make the interface for a password prompt. We will ask the user to enter the password and store it in the variable *pass.* If it matches the pre-defined password, which is ‘password’ in this example, the user will get the output as -“The password is correct”. Else, the shell script will tell the user that the password was incorrect and ask them to try again.

```
#!/bin/bash
echo "Enter password"
read pass
if [ $pass="password" ]
then
  echo "The password is correct."
else
  echo "The password is incorrect, try again."
fi
```



```
root@ubuntu:~# bash passw.sh
Enter password
password
The password is correct.
root@ubuntu:~#
```

Case Statements Script

The basic syntax of the **case...esac** statement is to give an expression to evaluate and to execute several different statements based on the value of the expression.

The interpreter checks each case against the value of the expression until a match is found. If nothing matches, a default condition will be used.

```
case word in
  pattern1)
    Statement(s) to be executed if pattern1 matches
    ;;
  pattern2)
    Statement(s) to be executed if pattern2 matches
    ;;
  pattern3)
    Statement(s) to be executed if pattern3 matches
    ;;
  *)
   Default condition to be executed
   ;;
esac
```

Here the string word is compared against every pattern until a match is found. The statement(s) following the matching pattern executes. If no matches are found, the case statement exits without performing any action.

There is no maximum number of patterns, but the minimum is one. When the statement(s) part executes, the command; indicates that the program flow should jump to the end of the entire case statement. **Example:**

```
#!/bin/sh


FRUIT="kiwi"
case "$FRUIT" in
   "apple") echo "Apple pie is quite tasty."
   ;;
   "banana") echo "I like banana nut bread."
   ;;
   "kiwi") echo "New Zealand is famous for kiwi."
   ;;
esac
```

Upon execution, you will receive the following result −

```
New Zealand is famous for kiwi.
```

A good use for a case statement is the evaluation of command line arguments as follows −

```
#!/bin/sh
option="${1}"
case ${option} in
  -f) FILE="${2}"
     echo "File name is $FILE"
     ;;
  -d) DIR="${2}"
     echo "Dir name is $DIR"
     ;;
  *)
     echo "`basename ${0}`:usage: [-f file] | [-d directory]"
     exit 1 # Command to come out of the program with status 1
     ;;
esac
```

Here is a sample run of the above program –

```
$./test.sh
test.sh: usage: [ -f filename ] | [ -d directory ]
$ ./test.sh -f index.htm
$ vi test.sh
$ ./test.sh -f index.htm
File name is index.htm
$ ./test.sh -d unix
Dir name is unix
$
```