Shell Scripting

18 February 2025 07:05 AM

And (&&) Operator in ShellScript

The second command will only execute if the first command has executed successfully i.e, its exit status is zero. This operator can be used to check if the first command has been successfully executed. This is one of the most used commands in the command line.

Syntax:

```
command1 && command2
```

command2 will execute if command1 has executed successfully. This operator allows us to check the exit status of command1. and it returns true only and only if all commands execute successfully.

```
┌──(gaurav㉿learning-ocean)-[~/shellscript-youtube]
└─$ ping -c 1 8.8.8.8>/dev/null && echo "Internet working fine."
Internet working fine.
```

Example:

```
#!/bin/bash
# os == linux && user == root
OS_TYPE=$(uname)
if [[ ${OS_TYPE} == "Linux" &&  ${UID} -eq 0 ]]
then
   echo "user is root user and os is linux."
fi
```

let's execute the above program as a non-root user and see the output

```
┌──(gaurav㉿learning-ocean)-[~/shellscript-youtube]
└─$ ./if-and-operator.sh
```

now let's run the same program as a root user.

```
┌──(gaurav㉿learning-ocean)-[~/shellscript-youtube]
└─$ sudo su
[sudo] password for gaurav:
┌──(root💀learning-ocean)-[/home/kali/shellscript-youtube]
└─#  ./if-and-operator.sh
user is root user and os is linux.
```

OR (||) Operator in ShellScript

This is logical **OR**. If one of the operands is true, then the condition becomes true.

Syntax:

```
command1 && command2
```

command2 will execute if command1 has failed.
and it returns false only and only if all commands return not zero exit code.

```
┌──(gaurav㉿learning-ocean)-[~/shellscript-youtube]
└─$ ping -c 1 8.8.8.8>/dev/null || echo "Internet is not working."
Internet is not working.
```

Example:

```
#!/bin/bash
# os == linux && user == root
OS_TYPE=$(uname)
if [[ ${OS_TYPE} == "Linux" || ${UID} -eq 0 ]]
then
  echo "user is root user or os is linux."
fi
```

let's execute the above program as a non-root user and see the output

```
┌──(gaurav㉿learning-ocean)-[~/shellscript-youtube]
└─$ ./if-or-operator.sh
user is root user or os is linux.                                    1
```

now let's run the same program as a root user.

```
┌──(gaurav㉿learning-ocean)-[~/shellscript-youtube]
└─$ sudo su                                                          1 ×
[sudo] password for gaurav:
┌──(root💀learning-ocean)-[/home/kali/shellscript-youtube]
└─#  ./if-or-operator.sh
```

```
user is root user or os is linux.
```
Example 2:
```
#!/bin/bash
read -p "do you want to continue (Y/y/yes) " uservalue
if [[ ${uservalue,,} == 'y' || ${uservalue,,} == 'yes' ]]
then
  echo "you want it"
else
  echo "you dont want it."
fi
```
executing the above script four-time and supply different outputs and check the output in the below section.
```
┌──(gaurav㉿learning-ocean)-[~/shellscript-youtube]
└─$ ./if-or-operator.sh
do you want to continue (Y/y/yes) y
you want it
┌──(gaurav㉿learning-ocean)-[~/shellscript-youtube]
└─$ ./if-or-operator.sh
do you want to continue (Y/y/yes) Y
you want it
┌──(gaurav㉿learning-ocean)-[~/shellscript-youtube]
└─$ ./if-or-operator.sh
do you want to continue (Y/y/yes) yes
you want it
┌──(gaurav㉿learning-ocean)-[~/shellscript-youtube]
└─$ ./if-or-operator.sh
do you want to continue (Y/y/yes) Yes
you want it
```

If-Else in ShellScript

If the specified condition is not true in the if part then the else part will be executed.
Syntex:
```
if [ expression ]
then
   statement1
else
   statement2
fi
```
Example:
```
#!/bin/bash
name=""
othername="saurav sharma"
if [[ -n ${name} ]]
then
   echo "length of string is non zero"
else
   echo "length of string is zero"
fi
if [[ -z ${name} ]]
then
   echo "length of string is zero -two"
else
   echo "length of string is non zero. = two"
fi
if [[ ${name} == ${othername} ]]
then
   echo "both string are equals - three"
else
   echo "both string are not equals. - three"
fi
if [[ ${name} != ${othername} ]]
then
   echo "both string are not equals -four"
else
   echo "both strings are equals -four"
fi
echo "I am Here"
```
output:
```
┌──(gaurav㉿learning-ocean)-[~/shellscript-youtube]
└─$ ./if-else.sh
length of string is zero
length of string is zero -two
```

```
both string are not equals. - three
both string are not equals -four
I am Here
```

Elif in ShellScript

if..elif..else..fi statement (Else If ladder)

To use multiple conditions in one if-else block, then elif keyword is used in shell. If the expression1 is true then it executes statements 1 and 2, and this process continues. If none of the conditions is true then it processes else part.

Syntax

```
if [ expression1 ]
then
   statement1
   statement2
   .
   .
elif [ expression2 ]
then
   statement3
   statement4
   .
   .
else
   statement5
fi
```

Example:

```
#!/bin/bash
number=4
if [[ ${number} -eq 10 ]]
then
  echo "number is 10"
elif [[ ${number} -lt 10 ]]
then
  echo "number is less then 10"
elif [[ ${number} -lt 5 ]]
then
  echo "number is less then 5"
else
  echo "number is grater then 10"
fi
```

Output

```
┌──(gaurav㉿learning-ocean)-[~/shellscript-youtube]
└─$ ./elif.sh
number is less then 10
```

Case in ShellScript

You can use multiple **if..elif** statements to perform a multiway branch. However, this is not always the best solution, especially when all of the branches depend on the value of a single variable.

Shell supports **case...esac** statement which handles exactly this situation, and it does so more efficiently than repeated if...elif statements.

The **case** statement saves going through a whole set of **if .. then .. else** statements. Its syntax is really quite simple:

Example:

```
#!/bin/bash
action=${1,,}
# start,stop,restart,reload
case ${action} in
   start)
      echo "going to start"
      echo "actionone two"
      ;;
   stop)
      echo "going to stop"
      ;;
   reload)
      echo "going to reload"
```

```
    ;;
  restart)
    echo "going to restart"
    ;;
  *)
    echo "please enter correct command line args."
esac
```

Let's run the above program with one command line argument.

```
┌──(gaurav㉿learning-ocean)-[~/shellscript-youtube]
└─$ ./casestatement.sh START
going to start
actionone two

┌──(gaurav㉿learning-ocean)-[~/shellscript-youtube]
└─$ ./casestatement.sh start
going to start
actionone two

┌──(gaurav㉿learning-ocean)-[~/shellscript-youtube]
└─$ ./casestatement.sh stop
going to stop

┌──(gaurav㉿learning-ocean)-[~/shellscript-youtube]
└─$ ./casestatement.sh sToP
going to stop
```

Case Statement with Regex

Example

```
read -p "enter y or n: " ANSWER
case "$ANSWER" in
  [Yy] | [Yy][Ee][Ss])
    echo "you answer yes"
    ;;
  [Nn] | [Nn][Oo])
    echo "you answer no"
    ;;
  *)
    echo "Invalid Answer"
    ;;
    exit
esac
```

output

```
┌──(gaurav㉿learning-ocean)-[~/shellscript-youtube]
└─$ ./casethree.sh
enter y or n: y
you answer yes
┌──(gaurav㉿learning-ocean)-[~/shellscript-youtube]
└─$ ./casethree.sh
enter y or n: ye
Invalid Answer

┌──(gaurav㉿learning-ocean)-[~/shellscript-youtube]
└─$ ./casethree.sh
enter y or n: yes
you answer yes
┌──(gaurav㉿learning-ocean)-[~/shellscript-youtube]
└─$ ./casethree.sh
enter y or n: n
you answer no
```

Example-2

```
#!/bin/bash
read -p "enter y or n" answer
case ${answer,,} in
  [y]*)
    echo "you enter Yes"
    ;;
  [n]* )
    echo "you enter no"
    ;;
  *)
    echo "Invalid Anser"
    ;;
esac
```

```
┌──(gaurav㉿learning-ocean)-[~/shellscript-youtube]
└─$ ./casestatement-2.sh
```

```
enter y or ny
you enter Yes

┌──(gaurav㉿learning-ocean)-[~/shellscript-youtube]
└─$ ./casestatement-2.sh
enter y or nyeeeeeee
you enter Yes
```

While Loop in ShellScript

A **while loop** is a statement that iterates over a block of code till the condition specified is evaluated to true. We can use this statement or loop in our program when do not know how many times the condition is going to evaluate to false before evaluating to true.
This repeats until the condition becomes false.
syntax:

```
while [[ condition ]]
do
   # statements
   # commands
done
while [ condition ]
do
   # statements
   # commands
done
```

Example-1

```
#!/bin/bash
while [[ $answer != "yes" ]]
do
   read -p "please enter yes " answer
done
```

Example-2

```
#!/bin/bash
# example of infinite loop
while true
do
   echo "this is test"
done
```

output

```
this is test
this is test
...
...
```

Example-3

```
#!/bin/bash
read -p "please enter a number " number
initNumber=1
while [[ ${initNumber} -le 10 ]]
do
   echo $((initNumber*number))
   ((initNumber++))
done
```

output:

```
┌──(gaurav㉿learning-ocean)-[~/shellscript-youtube]
└─$ ./while-loop.sh
please enter a number 2
2
4
6
8
10
12
14
16
18
20
```

Read File in ShellScript

we can read a file with the help of read and while. The return code of **read** command is zero, unless the end-of-file is encountered.

```
#!/bin/bash
file_path="/etc/passwd"
while read line
do
   echo "$line"
   sleep 0.20
done < $file_path
```

Until Loop

The Until loop is used to iterate over a block of commands until the required condition is false.

Syntax:

```
until [ condition ];
do
 block-of-statements
done
```

Here, the flow of the above syntax will be --

- Checks the condition.
- if the condition is false, then executes the statements and goes back to step 1.
- If the condition is true, then the program control moves to the next command in the script.

Example

```
#!/bin/bash
read -p "please enter a number" number
initNumber=1
until [[ initNumber -eq 11 ]]
do
   echo $((initNumber*number))
   ((initNumber++))
done
```

output

```
┌──(gaurav㉿learning-ocean)-[~/shellscript-youtube]
└─$ ./until-loop.sh
please enter a number3
3
6
9
12
15
18
21
24
27
30
```

For Loop

The for loop moves through a specified list of values until the list is exhausted.

- Keywords are for, in, do, done
- List is a list of variables which are separated by spaces. If list is not mentioned in the for statement, then it takes the positional parameter value that were passed into the shell.
- Varname is any variable assumed by the user.

```
#!/bin/bash
for variableName in item1 item2 item3 item4 item5 item6
do
 echo "${variableName}"
done
```

output:

```
item1
item2
item3
item4
item5
item6
```

we can use range in for loop.

```
#!/bin/bash
read -p "please enter a number " number
for variableName in {1..10}
do
  echo $((variableName*number))
done
```

output:

```
please enter a number 2
2
4
6
8
10
12
14
16
18
20
```

```
#!/bin/bash
for variableName in "gaurav Sharma" "ankit Joshi" "rajkumar meena"
do
 echo "${variableName}"
done
```

output:

```
gaurav sharma
ankit Joshi
rajkumar meena
```

```
#!/bin/bash
for i in *
do
  echo $i
done
```

output: will print all the files and folder name of present present working directory.

```
#!/bin/bash
for i in $(ls *.txt)
do
  echo "$i"
done
```

output: it will print all the file name with txt extension