Shell Scripting

19 February 2025 07:38 AM

$@ and $* in ShellScript

$@ behaves like $* except that when quoted the arguments are broken up properly if there are spaces in them.

```
for var in "$@"
do
   echo "$var"
done
```

Gives this:

```
$ sh test.sh 1 2 '3 4'
1
2
3
4
```

Now change "$@" to $*:

```
for var in $*
do
   echo "$var"
done
```

And you get this:

```
$ ./test.sh 1 2 '3 4'
1
2
3
4
```

```
#!/bin/bash
echo "=========== loop one ========"
for i in "$*"
do
   echo $i
done
echo "=========== loop two ========"
for i in "$@"
do
   echo $i
done
```

output:

```
┌──(gaurav㉿learning-ocean)-[~/shellscript-youtube]
└─$ ./commandline-args.sh gaurav saurav amit manish
=========== loop one ========
gaurav saurav amit manish
=========== loop two ========
gaurav
saurav
amit
manish
```

```
┌──(gaurav㉿learning-ocean)-[~/shellscript-youtube]
└─$ ./commandline-args.sh gaurav saurav amit "manish sharma"
=========== loop one ========
gaurav saurav amit manish sharma
=========== loop two ========
gaurav
saurav
amit
manish sharma
```

Continue statement in ShellScript

The **continue** statement is similar to the **break** command, except that it causes the current iteration of the loop to exit, rather than the entire loop. This statement is useful when an error has occurred but you want to try to execute the next iteration of the loop.

Syntax

```
continue
```

Like with the break statement, an integer argument can be given to the continue command to skip commands from nested loops.

```
continue n
```

Here **n** specifies the **nth** enclosing loop to continue from.

Example

```
#!/bin/bash
initNumber=1
while [[ ${initNumber} -lt 10 ]]
do
  ((initNumber++))
  if [[ ${initNumber} -eq 5 ]]
  then
    continue
  fi
  echo ${initNumber}
done
```

Output

```
┌──(gaurav㉿learning-ocean)-[~/shellscript-youtube]
└─$ ./continue-statement.sh
2
3
4
6
7
8
9
10
```

Break Statement in Shellscript

The **break** statement is used to terminate the execution of the entire loop, after completing the execution of all of the lines of code up to the break statement. It then steps down to the code following the end of the loop.

Syntax

The following **break** statement is used to come out of a loop.

```
break
```

The break command can also be used to exit from a nested loop using this format.

```
break n
```

Here **n** specifies the **nth** enclosing loop to the exit from.

example

```
#!/bin/bash
initNumber=1
while [[ ${initNumber} -lt 10 ]]
do
   echo ${initNumber}
   if [[ ${initNumber} -eq 5 ]]
   then
     echo "condition is true number is ${initNumber} going to break the loop."
     break;
   fi
   ((initNumber++))
done
```

Nested If-Else

we can define if-else inside if-else.
A nested if-else block can be used when one condition is satisfied then it again checks another condition.
example

```
#!/bin/bash
number=9
if [[ ${number} -gt 10 ]]
then
   if [[ $number -gt 50 ]]
   then
      if [[ ${number} -gt 100 ]]
      then
         echo "number is grater then 100"
      fi
   else
      echo "number is in between 11 to 50"
   fi
else
```

```
   echo "number is less then or equal to 10"
fi
┌──(gaurav㉿learning-ocean)-[~/shellscript-youtube]
└─$ ./nested-if-else.sh
number is less then or equal to 10
```

Nested Loop In ShellScript

Nested for loops means loop within loop. They are useful for when you want to repeat something several times for several things.
example:

```
#!/bin/bash
initNumber=1
while [[ ${initNumber} -lt 3 ]]
do
   for i in item1 item2 item3
   do
      echo "${initNumber} - ${i}"
   done
   ((initNumber++))
done
```

output:

```
┌──(gaurav㉿learning-ocean)-[~/shellscript-youtube]
└─$ ./nested-loop.sh
1 - item1
1 - item2
1 - item3
2 - item1
2 - item2
2 - item3
```

nested loop with break level example

```
#!/bin/bash
initNumber=1
while [[ ${initNumber} -lt 3 ]]
do
   for i in item1 item2 item3
   do
      echo "${initNumber} - ${i}"
      if [[ $i == item2 ]]
      then
         break 2
      fi
   done
   ((initNumber++))
done
```

output:

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

```
1 - item1
1 - item2
```

Select Loop in ShellScript

The select loop is an infinite loop that only ends when there's a keyboard interrupt or abreak statementis encountered. But that's not what makes it unique or interesting. **The select statement allows users to choose from multiple options by default** and it will prompt the user for an input. You do not have to write any code to accept user input as the select loop is pre-built to handle it. This loop can be used to make menus within your script while keeping the script looping infinitely. Another benefit of the select loop in shell scripts is that it can be combined with theswitch case statementsto create really interactive menus or script pivots. Let's learn how to make use of this loop and work with it.

```
#!/bin/bash
PS3="please select os? "
select os in linux windows mac
do
  case ${os} in
    linux)
      echo "you selected linux"
      echo "thanks."
      break
      ;;
    windows)
      echo "you selected windows"
      echo "thanks."
      break
      ;;
    mac)
      echo "you selected mac"
      echo "thanks."
      break
      ;;
    *)
      echo "Invalid"
  esac
done
```

output:

```
┌──(gaurav㉿learning-ocean)-[~/shellscript-youtube]
└─$ ./select-statement.sh
1) linux
2) windows
3) mac
please select os? 1
you selected linux
thanks.
```

let's run the same program with some invalid value.

```
┌──(gaurav㉿learning-ocean)-[~/shellscript-youtube]
└─$ ./select-statement.sh
1) linux
2) windows
3) mac
please select os? 5
Invalid
please select os? 4
Invalid
please select os? 8
Invalid
please select os? 6
Invalid
please select os? 2
you selected windows
thanks.
```