BATCH : BATCH 48

LESSON : **BATCH SCRIPTING**

DATE : 03.01.2021

SUBJECT : **BASH LOOPS**

techproeducation

techproeducation

techproeducation

techproeducation

techproedu

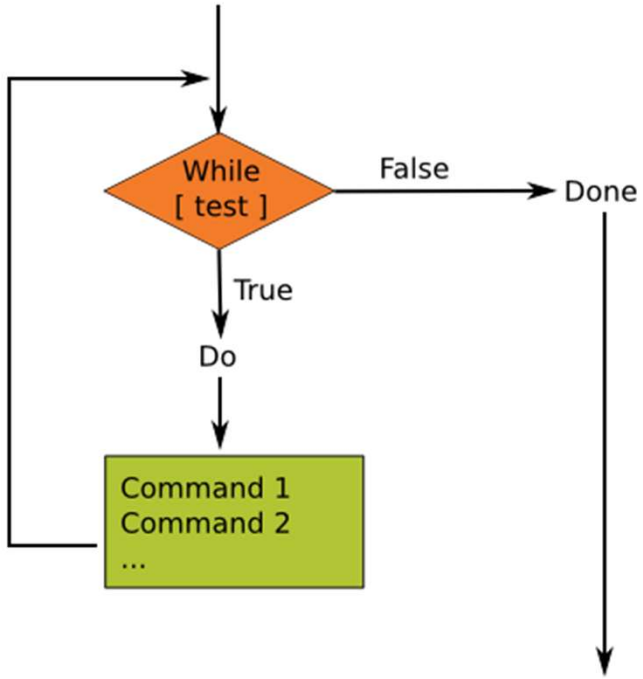
techproeducation.com

info@techproeducation.com

+1 (917) 768-7466

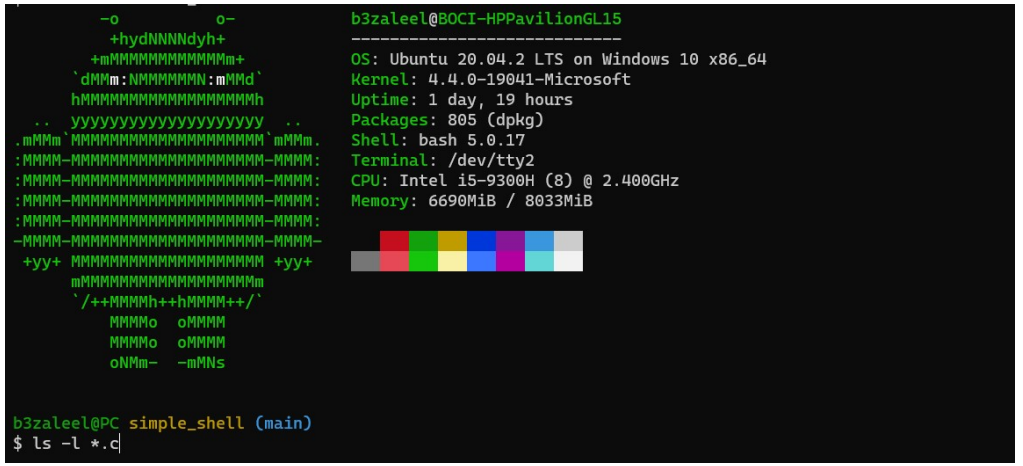
LOOPS

- Loops is used for repetitive tasks. During the loops, we'll cover
  - For Loop
  - While Loop
  - Continue and Break Statements
  - Select Loop
  - Case Statements



SHELLS

- Today, we also dive into more about the shells:
  - dash vsbash
  - Working with bash in other Shells
  - EXIT CODES



LOOPS

- **For Loop**
- for loops is used when you want to run same command multiple times
- Execute a command or a set of commands many times
- Iterate through files

```
for i in 0 1 2 3 4 5 6 7 8 9
do
    echo $i
done
```

LOOPS

- **For Loop**
- Iterate through lines within a file
- Iterate through the output of a command
- for loops can be written in multiple ways:

```
for file in $(ls)
do
    echo $file
done
```

LOOPS

- **While Loop**
- While loop works just like the for loop, except theat it executes the loop as long as the condition is true.
- Execute a command or a set of commands multiple times but you are not sure how many times.
- Execute a command or a set of commands until a specific condition occurs

```
number=10

while [[ $number -le 100  ]]
do
  echo $number
  ((number++))
done
echo "While loop is on the $number th number now"
```

LOOPS

- **Break & Continue Statements**
- **Break**
- The break statement is used to terminate the execution of the entire loop.

```
#!/bin/bash

numara=20

while [[ $numara -gt 1  ]]
do
  echo $numara
  ((numara++))
  if [[ $numara -eq 100 ]]
  then
    break
  fi
done
```

LOOPS

- **Break & Continue Statements**
- **Continue**
- The Continue statement is similar to the Break command, except it causes the current iteration of the loop to exit, instead of the whole loop.

```
#!/bin/bash

number=1

until [[ $number -lt 1  ]]
do
  ((number++))

  fivess=$(($number % 5))

  if [[ $tens -eq 0 ]]
  then
    continue
  fi

  echo $number

  if [[ $number -gt 100 ]]
  then
    break
  fi
done
```

LOOPS

- **Select Loops**
- The Select Loop generates a numbered menu from which users can select options. It's helpful when you need to ask the user to select one or more items from a list of options.

```
#!/bin/bash

read -p "Input first number: " first_number
read -p "Input second number: " second_number

PS3="Select the operation: "

select operation in addition subtraction multiplication division exit
do
  case $operation in

    multiplication)
       echo "result= $(( $first_number * $second_number))"
       ;;
    division)
       echo "result= $(( $first_number / $second_number))"
    ;;
    addition)
      echo "result= $(( $first_number + $second_number))"
    ;;
    subtraction)
       echo "result= $(( $first_number - $second_number))"
    ;;
    exit)
       break
    ;;
    *)
       echo "Wrong choice..."
    ;;
  esac
done
```

LOOPS

- **Case Statements**
- Case statement is used for simplifying multiple condition check with multiple different choices.

```
echo "1. Shutdown"
echo "2. Restart"
echo "3. Exit Menu"
read -p "Enter your choice: " choice

case $choice in

    1) shutdown now
       ;;
    2) shutdown -r now
       ;;
    3) break
       ;;
    *) continue
       ;;

esac
```

LOOPS

- **Case Statements**
- Case statement can also be written along with the **while** loop as shown:

```
while true
do
  echo "1. Shutdown"
  echo "2. Restart"
  echo "3. Exit Menu"
  read -p "Enter your choice: " choice

  case $choice in

      1) shutdown now
         ;;
      2) shutdown -r now
         ;;
      3) break
         ;;
      *) continue
         ;;

  esac
done
```

SHELLS – bash vs dash

- **bash**
  - Bourne Shell is, in fact, the bash shell itself, it is linked to the bash.
  - Bash continues to be used as the default interactive/login shell for users, but Dash is the one at /bin/sh and the one which is executed for system scripts such as init scripts.

```
#!/bin/bash
# set the STRING variable
STRING="Hello World!"
# print the contents of the variable
echo $STRING
```

- **dash**
  - Dash is very fast, but also is very closely POSIX-compatible - a standard that is aligned closely with the Bourne shell. So in a way, by switching from Bash to Dash we are moving back to a shell more closely aligned with Bourne.

```
chris@ubuntu:~$ bash
chris@ubuntu:~$ zsh
chris@ubuntu ~ % tcsh
ubuntu:~> dash
$
```

SHELLS – Working with bash in Other Shells

- **Working with bash in Other Shells**
- If you want to run the script from another shell, that's not a bash shell, then you must just run it through the bash shell by prefixing the bash command.

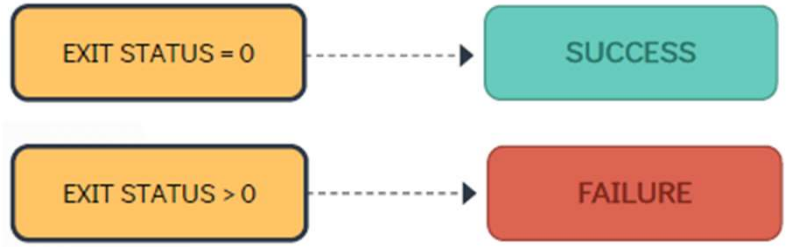
```
ubuntu@ubuntu-VirtualBox:~/code$ bash elseif_example.sh
Enter your lucky number
101
You got 1st prize
ubuntu@ubuntu-VirtualBox:~/code$ bash elseif_example.sh
Enter your lucky number
999
You got 3rd prize
ubuntu@ubuntu-VirtualBox:~/code$ bash elseif_example.sh
Enter your lucky number
100
Sorry, try for the next time
ubuntu@ubuntu-VirtualBox:~/code$
```

SHELLS – Exit Codes

- **EXIT CODES**

- An exit code, or sometimes known as a return code, is the code returned to a parent process by an executable.

- If a command runs successfully, shell returns 0.
- If it fails, shell returns a value greater than 0.
- Command to check the exit code of the last executed command is `echo $?`




```
$ echo ?
0
```