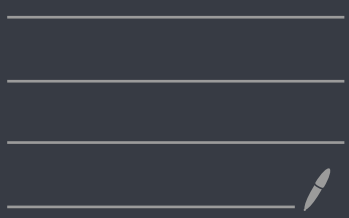
Bash Shell & Scripts



what is Bash? (Bourne again shell)

→ $ bash version (To check bash version)

→ Pipe & Redirection

. it sends output of one process to another

For ex: cat lora.txt || less
(pipe)

→ Redirection → stadaad input
→ standard ouput
→ stand error

send the result in other file

For ex: ls > list.txt (Added)

result in list.txt

2) ls >> list.txt (is added not replaced)

For error ls /notreal 1> output.txt (output)
2> error.txt (error)

(it is an error)

in bash ex. $ echo Hello there → Hello there (result.)
$ printf Hello → Hello $

→ brackets & braces

( )	{ }	[ ]
Parenthesis	Braces	Brackets

→ expansions & substitutions
(tells bash to generate or calculate value not known before script runs)

1) ~ → represents user's home variable (username)

2) {a,b,c} → Brace expansion creates sets or ranges.

Brace Expansion

eg:
echo {a..z}
Result: a, b, c, ---, z.

echo {1..30..3}
Result: 1, 4, 7, 10, 13 .... 30.

(every 2 number) 4) $ echo {a . z..2}
Result = a . e g ? ....

3) touch file_{01..12}{a..d}

result = will create file from

file_01a, file_02b ---- , file_12d

→ To delete all the file :

$ rm file_*

⇒ Parameter Expansion

${ } → setting a value & retreiving it.

(variable)

Eg : $ greeting = "hello there!"

$ echo ${greeting} (calling the variable)

2) $echo ${greeting : 6}

→ can get value after pos. (starting with 1)

result = there!

3) $echo ${greeting : 6 : 3}

(till 3rd word from 6th)

result = the

4) $echo ${greeting / there / everybody}

result = greeting everybody (there replaced by everybody)

5) $ echo ${greeting//e/_}
result = h_llo th_r_!

6) $ echo $greeting:4:3
result = hello there!:4:3

→ Command Substitution

$(...) → it is used to get values from other programs into our program.

Eg: uname -r (to know the release version)

result = 5.4.0---

by using $(...) → $ echo "The Python version is $(uname -r)."
result = The kernal... 5.4.0.---

→ $((...)) → Arithmetic expansion does calculations.

Eg: echo $((2+2))
result = 4

2) echo $((4*5))
result = 20

Running Multiple Commands

- one liners (in one line, seperated by ";", can be very long)

- Create Bash script
  - $ myscript.sh

- Executable bash script
  - first line (#! /usr/bin/env bash)
    echo "hello"

Eg: # This is a comment

Need to make executable.
- chmod +x myscript

can run locally ./myscript

Echo ...

Eg: $ echo hello world
result - hello word

2) $ worldsize = big
$ echo hello $worldsize world
(big)

result = hello big world

3) echo "The Kernel is $(uname -r)."
result = The Kernel is 5.4.0-1086-azure.

4) echo "the kernel is $(uname -r)"
result = "the kernel is $(uname -r)"

5) echo -n ("No newline")
result = No newLine $

Bash Variables

Eg: mygreeting = Hello
greeting2 = "Hello world!"
number = 16

→ $ echo $ mygreeting
result = Hello

→ $ echo $ greeting2
result = Hello world!
$ echo $ number
16

→ variables can be changed.

→ $ declare -r string = "Scott" (read-only)

(cannot change string)

$ echo $ string (result = scott)

string = "Daniel"

$ echo $ string (result = Daniel)

→ $ declare -l lowerstring = "THIS IS GOOD"

$ echo $ lowerstring (result = "this is good")

→ $ declare -u upperstring = "this is no good"

$ echo $ upperstring - vica versa.

→ Arithmetic operations

$ echo $((4+4)) = 8

$ b = 5
$ echo $b → 5

$b = $b + 2
$ echo $b → 5 + 2

↗ Basic calculator

we can use bc and awk (pre-installed)

Eg: $ declare -i c=1
$ declare -i d=3
→ (integer)

```
$ e= $( echo " scale=3 ; $c/$d " | bc)
```

scale → till how many decimal points

$ echo $e → 0.333

→ $ echo $RANDOM (give random no.)

Eg: $ echo $((1 + RANDOM %10))
(under 10th no.)

→ 8

[ ] → test built in

→ whenever we run comparison:
0 means success, 1 means failure.

(0 = true, 1 = false)

$ help test (will show file comparison options)

Eg: [-d /bin] ; echo $?
→ 0 → yes → (is bin a directory?)

2) [-d /bin/bash] ; echo $?
→ 1 → NO → (is bash directory?)

3) [" cat" = "dog "] ; echo $?
→ 1

4) [4 -lt 5] ; echo $?
(less than)
→ 0

5) [! 4 -lt 3] ; echo $?
→ 0
(False changed to True)

→ Extensive test = [[ ]]

eg: [[-d a && -a /bin/bash]] ;
echo $?
→ 0

eg: [-z $myvar]
→ To check

if it is empty

$ [[-d ~]] && echo ~ is a directory

→ /home/codespace is a directory.

(only print if it is true).

↳ More Simply $ true && echo "Success"

→ Success

2) $ false && echo "Success"

→ X

→ echo -e ...

$ echo -e "Name \t\t Number";
echo -e "Scott \t\t 123"

→ Name Number
Scott 123

→ For errors or color formatting:

ulinered = "\033[4;31;40m"
red = "\033[31;40m"
none = "\033[0m"

$ echo -e $ulinered "ERROR:"
$none $red "Something went wrong."
$none.

→ ERROR: something went wrong.

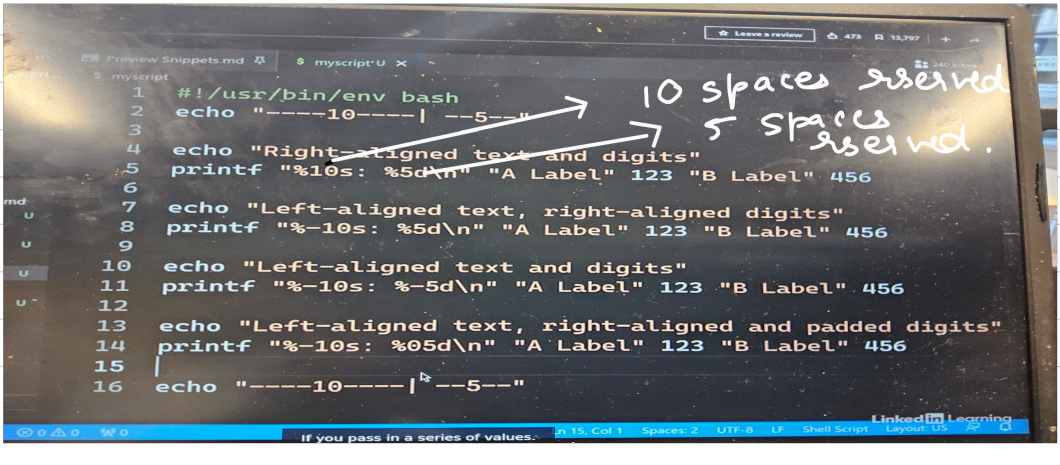
→ Printf

Eg: $ printf "The results are: %d and %d \n"
$((2+2)) $((3/1))

→ The results are: 4 and 3

%d - digit

%s - string



```
$ date +%.S
$ date +%.Y-%.m-%.d \
        %.H:%.M:%.S
$ printf "%.C.Y-%.m-%.d
        %.H:%.M:%.S)T\n"
        $(date +%.S)
```

-> 2022-08-26 19:14:00

-> Arrays

```
$ snacks = ("apple" "banana"
            "orange")
```

```
$ echo ${snacks[2]}
  -> Oxrange
```

```
$ snacks += ("mango")
  (will add at last of array)
```

```
$ for i in {0..6}, do echo
  "$i : ${snacks[i]}";
  done
```

0 : apple
1 : banana
2 : Orange --- etc.

```
$ declare -A office
$ office[city] = "san Fransisco"
$ office["building"] = "HQ west"
```

(if spaces is used)

```
$ echo ${office["building"]}
  is in ${office[city]}
```

→ HQ west is in San Fransico.