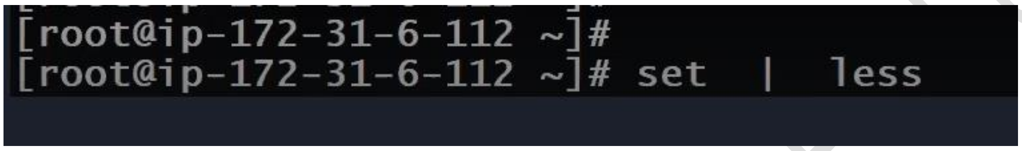
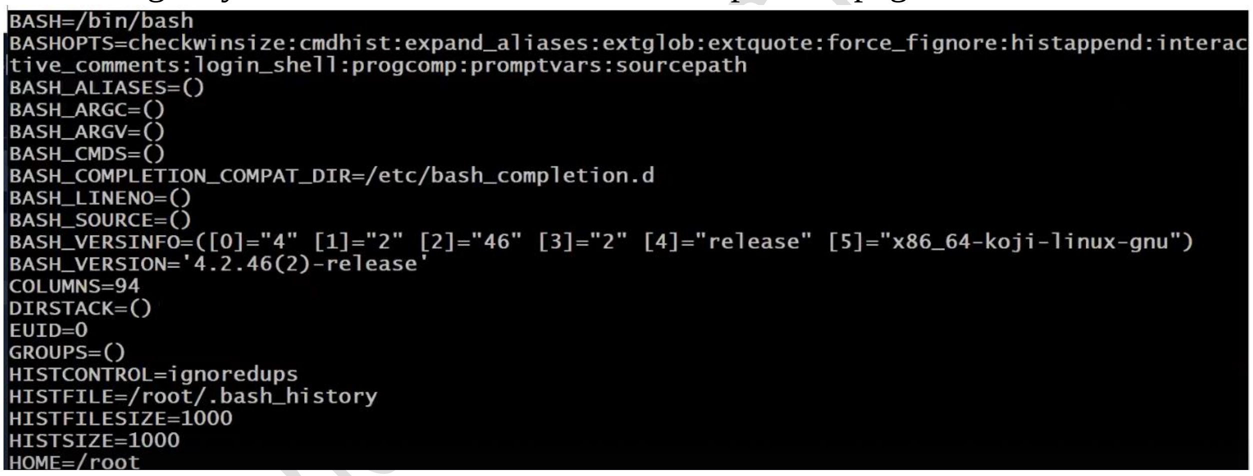
Shell and Shell Scripting Session No.3.2

- All the variables we have created in the previous class are known as User-Defined Variables. However, there are many pre-created variables given by the system.
- To see all those variables, use the below command



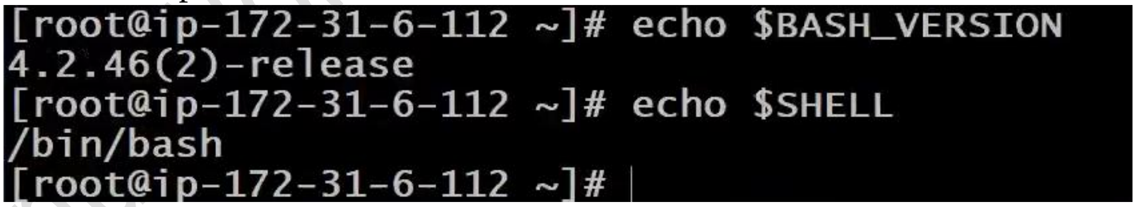
```
[root@ip-172-31-6-112 ~]#
[root@ip-172-31-6-112 ~]# set  |  less
```

- This will give you all those variables from the top of the page



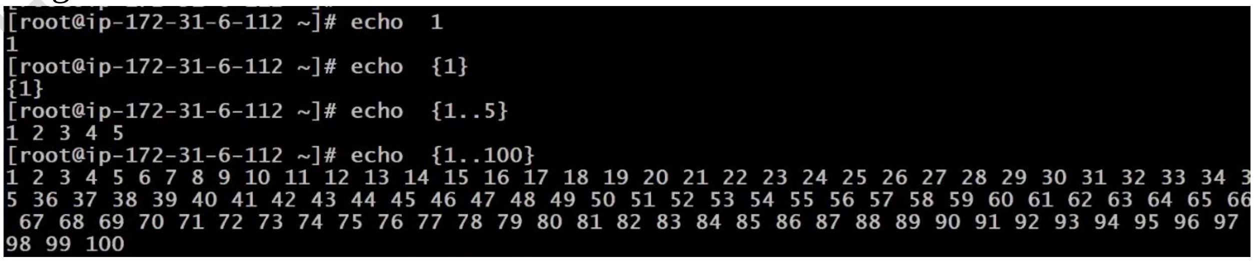
```
BASH=/bin/bash
BASHOPTS=checkwinsize:cmdhist:expand_aliases:extglob:extquote:force_fignore:histappend:interac
tive_comments:login_shell:progcomp:promptvars:sourcepath
BASH_ALIASES=()
BASH_ARGC=()
BASH_ARGV=()
BASH_CMDS=()
BASH_COMPLETION_COMPAT_DIR=/etc/bash_completion.d
BASH_LINENO=()
BASH_SOURCE=()
BASH_VERSINFO=([0]="4" [1]="2" [2]="46" [3]="2" [4]="release" [5]="x86_64-koji-linux-gnu")
BASH_VERSION='4.2.46(2)-release'
COLUMNS=94
DIRSTACK=()
EUID=0
GROUPS=()
HISTCONTROL=ignoredups
HISTFILE=/root/.bash_history
HISTFILESIZE=1000
HISTSIZE=1000
HOME=/root
```

- Some examples are the version or name of the shell



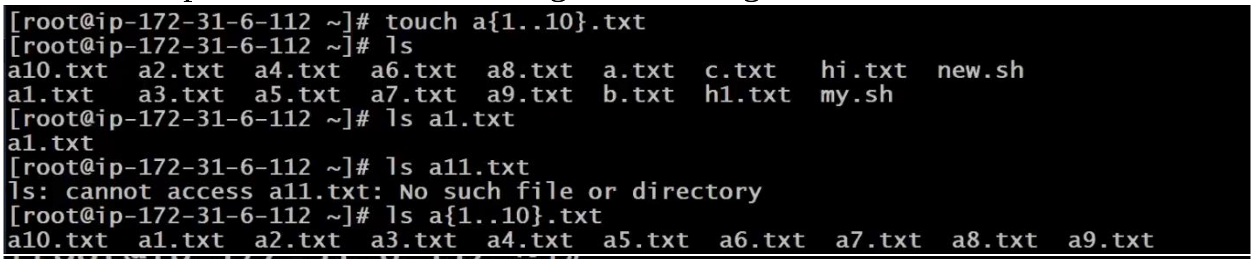
```
[root@ip-172-31-6-112 ~]# echo $BASH_VERSION
4.2.46(2)-release
[root@ip-172-31-6-112 ~]# echo $SHELL
/bin/bash
[root@ip-172-31-6-112 ~]#
```

- We can also use { } to give range also. Like below we use (..) to give range.

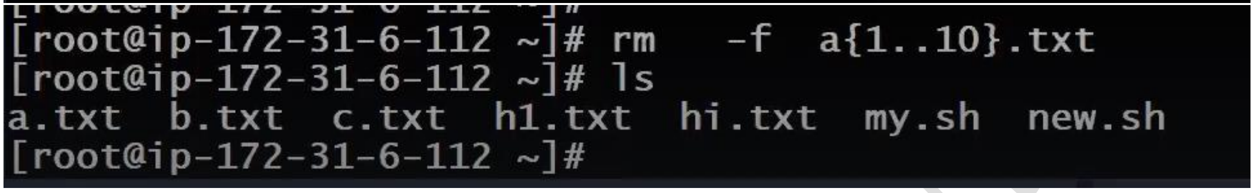


```
[root@ip-172-31-6-112 ~]# echo  1
1
[root@ip-172-31-6-112 ~]# echo  {1}
{1}
[root@ip-172-31-6-112 ~]# echo  {1..5}
1 2 3 4 5
[root@ip-172-31-6-112 ~]# echo  {1..100}
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 3
5 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66
 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97
98 99 100
```

- Same concept we can use for creating or removing files.

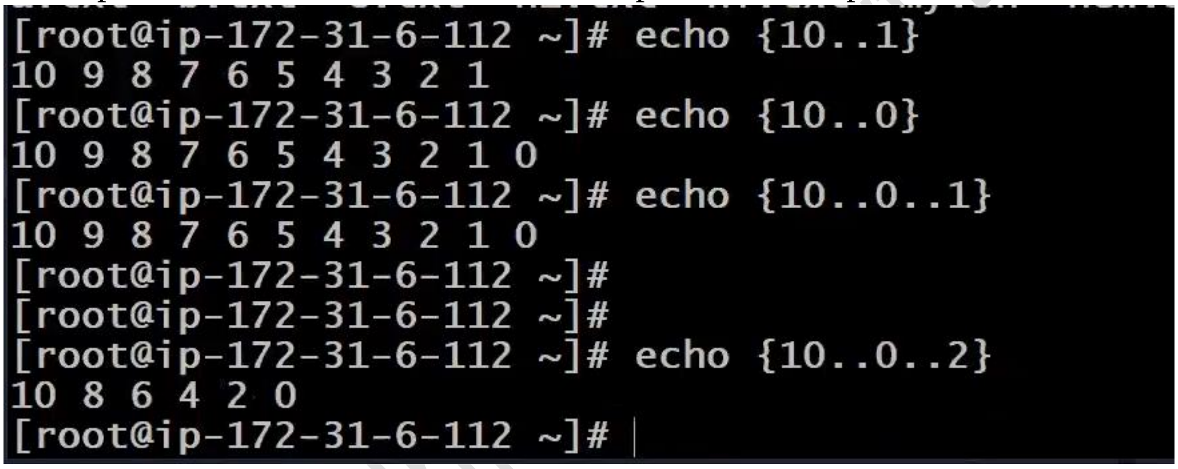


```
[root@ip-172-31-6-112 ~]# touch a{1..10}.txt
[root@ip-172-31-6-112 ~]# ls
a10.txt  a2.txt  a4.txt  a6.txt  a8.txt  a.txt  c.txt   hi.txt  new.sh
a1.txt   a3.txt  a5.txt  a7.txt  a9.txt  b.txt  h1.txt  my.sh
[root@ip-172-31-6-112 ~]# ls a1.txt
a1.txt
[root@ip-172-31-6-112 ~]# ls a11.txt
ls: cannot access a11.txt: No such file or directory
[root@ip-172-31-6-112 ~]# ls a{1..10}.txt
a10.txt  a1.txt  a2.txt  a3.txt  a4.txt  a5.txt  a6.txt  a7.txt  a8.txt  a9.txt
```



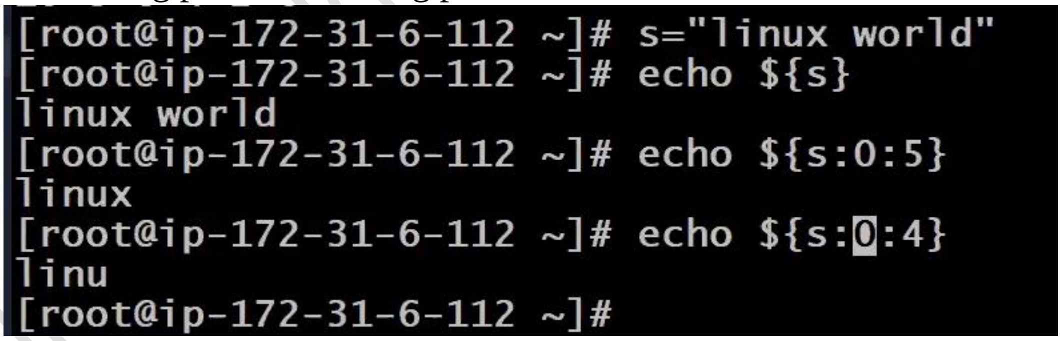
```
[root@ip-172-31-6-112 ~]# rm   -f  a{1..10}.txt
[root@ip-172-31-6-112 ~]# ls
a.txt  b.txt  c.txt  h1.txt  hi.txt  my.sh  new.sh
[root@ip-172-31-6-112 ~]#
```

- It can print in reverse order also with 2 steps or more steps.



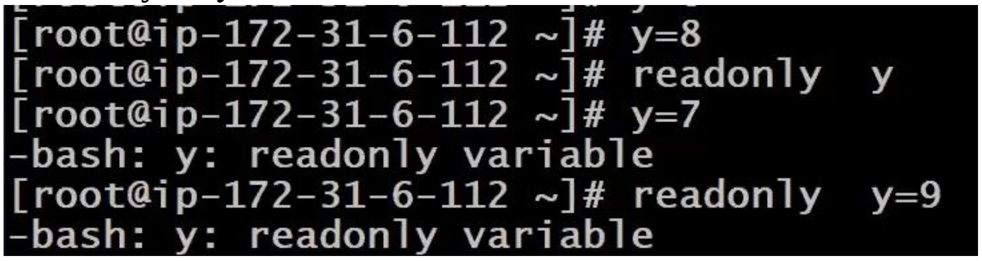
```
[root@ip-172-31-6-112 ~]# echo {10..1}
10 9 8 7 6 5 4 3 2 1
[root@ip-172-31-6-112 ~]# echo {10..0}
10 9 8 7 6 5 4 3 2 1 0
[root@ip-172-31-6-112 ~]# echo {10..0..1}
10 9 8 7 6 5 4 3 2 1 0
[root@ip-172-31-6-112 ~]#
[root@ip-172-31-6-112 ~]#
[root@ip-172-31-6-112 ~]# echo {10..0..2}
10 8 6 4 2 0
[root@ip-172-31-6-112 ~]#
```

- For doing the slicing operator in shell, we use curly brackets with ‘s’ and the starting point and ending point.



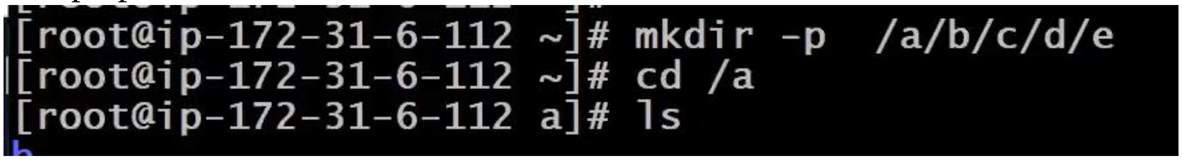
```
[root@ip-172-31-6-112 ~]# s="linux world"
[root@ip-172-31-6-112 ~]# echo ${s}
linux world
[root@ip-172-31-6-112 ~]# echo ${s:0:5}
linux
[root@ip-172-31-6-112 ~]# echo ${s:0:4}
linu
[root@ip-172-31-6-112 ~]#
```

- If you want to make a variable read-only it cannot be changed, we use the ‘read-only’ keyword.



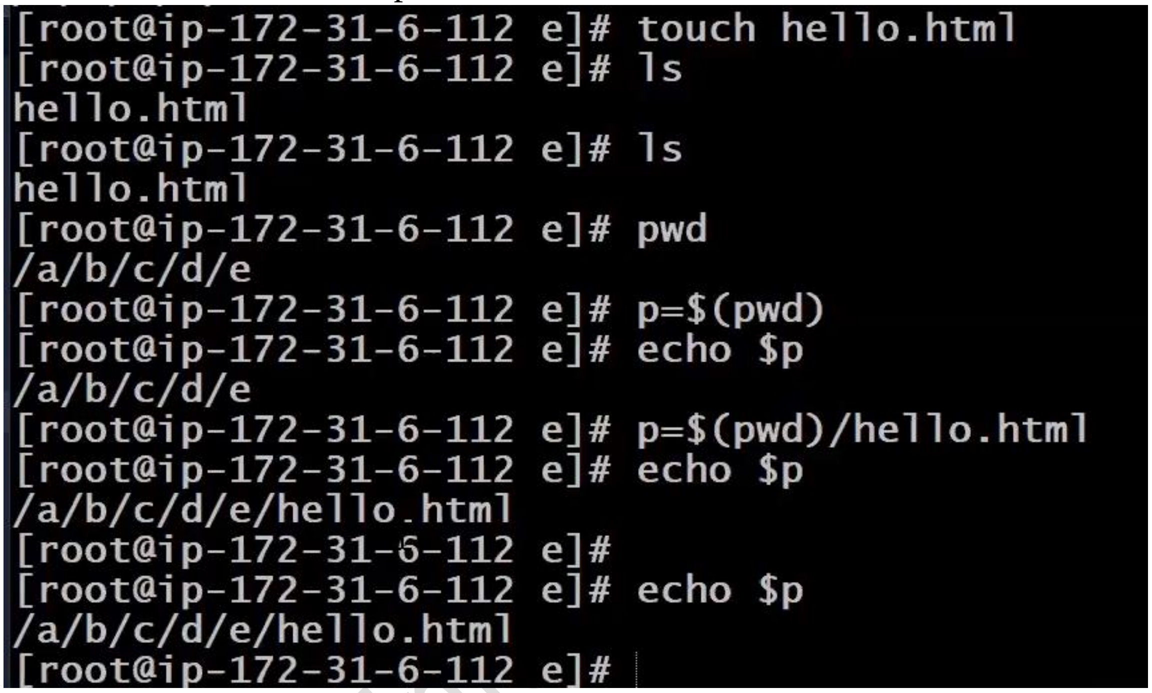
```
[root@ip-172-31-6-112 ~]# y=8
[root@ip-172-31-6-112 ~]# readonly  y
[root@ip-172-31-6-112 ~]# y=7
-bash: y: readonly variable
[root@ip-172-31-6-112 ~]# readonly  y=9
-bash: y: readonly variable
```

- For creating many directories continuously, one inside the other we use the -p option.



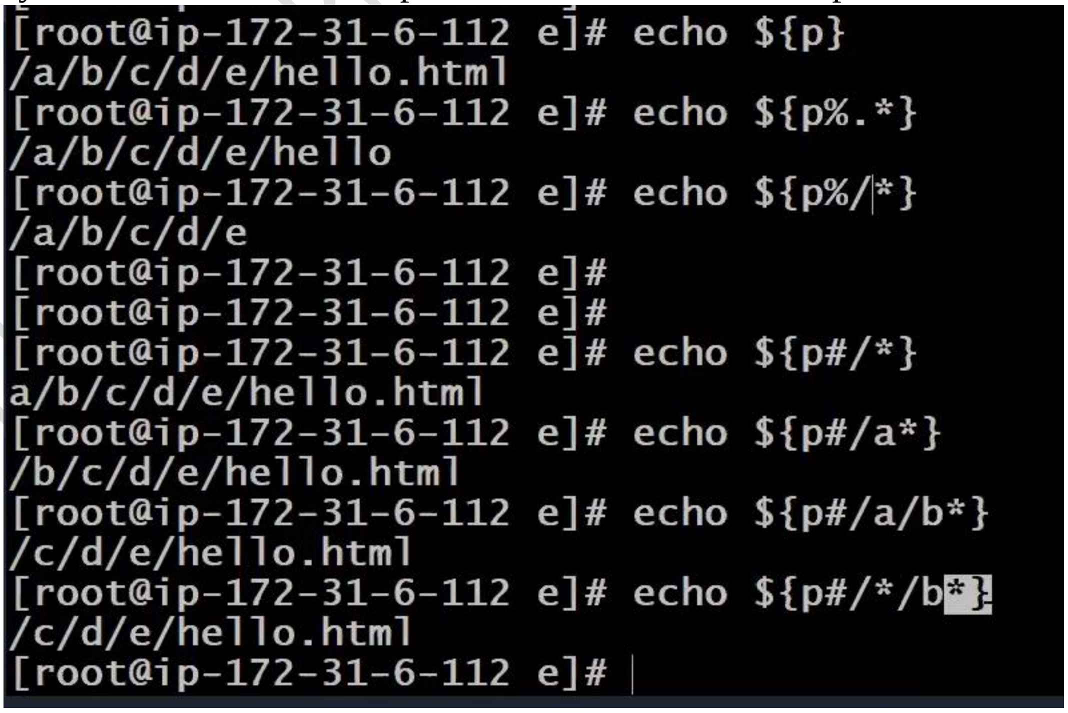
```
[root@ip-172-31-6-112 ~]# mkdir -p  /a/b/c/d/e
[root@ip-172-31-6-112 ~]# cd /a
[root@ip-172-31-6-112 a]# ls
```

- We can also store the full path of this in a variable.



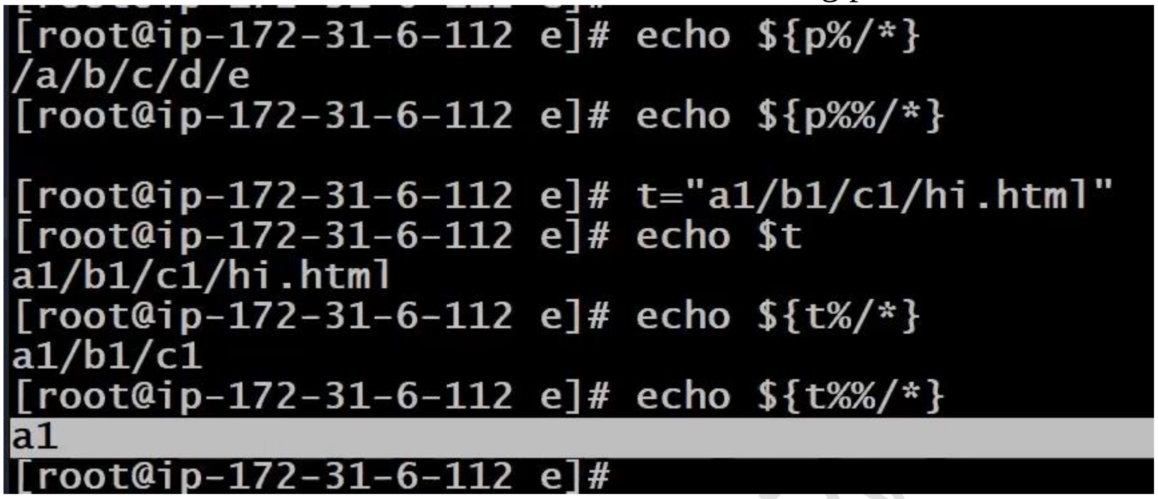
```
[root@ip-172-31-6-112 e]# touch hello.html
[root@ip-172-31-6-112 e]# ls
hello.html
[root@ip-172-31-6-112 e]# ls
hello.html
[root@ip-172-31-6-112 e]# pwd
/a/b/c/d/e
[root@ip-172-31-6-112 e]# p=$(pwd)
[root@ip-172-31-6-112 e]# echo $p
/a/b/c/d/e
[root@ip-172-31-6-112 e]# p=$(pwd)/hello.html
[root@ip-172-31-6-112 e]# echo $p
/a/b/c/d/e/hello.html
[root@ip-172-31-6-112 e]#
[root@ip-172-31-6-112 e]# echo $p
/a/b/c/d/e/hello.html
[root@ip-172-31-6-112 e]#
```

- As we know the % symbol will be removed from the suffix side but the # symbol removed from the prefix side and return the output.



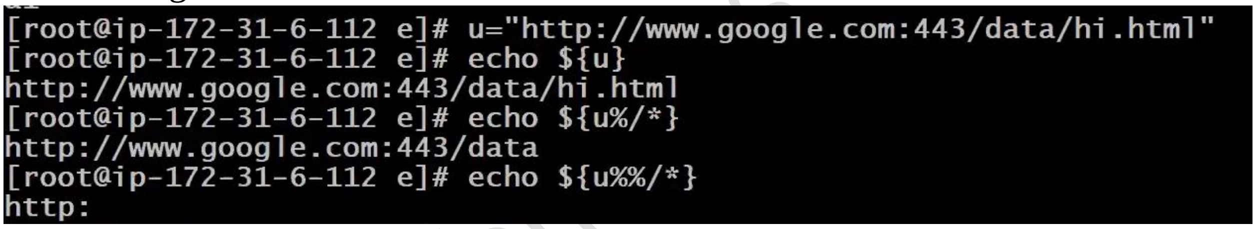
```
[root@ip-172-31-6-112 e]# echo ${p}
/a/b/c/d/e/hello.html
[root@ip-172-31-6-112 e]# echo ${p%.*}
/a/b/c/d/e/hello
[root@ip-172-31-6-112 e]# echo ${p%/*}
/a/b/c/d/e
[root@ip-172-31-6-112 e]#
[root@ip-172-31-6-112 e]#
[root@ip-172-31-6-112 e]# echo ${p#/*}
a/b/c/d/e/hello.html
[root@ip-172-31-6-112 e]# echo ${p#/a*}
/b/c/d/e/hello.html
[root@ip-172-31-6-112 e]# echo ${p#/a/b*}
/c/d/e/hello.html
[root@ip-172-31-6-112 e]# echo ${p#/*/b*}
/c/d/e/hello.html
[root@ip-172-31-6-112 e]#
```

- And we have another symbol also that is %%. This will go on till last / and whatever after it removes and returns the remaining part.



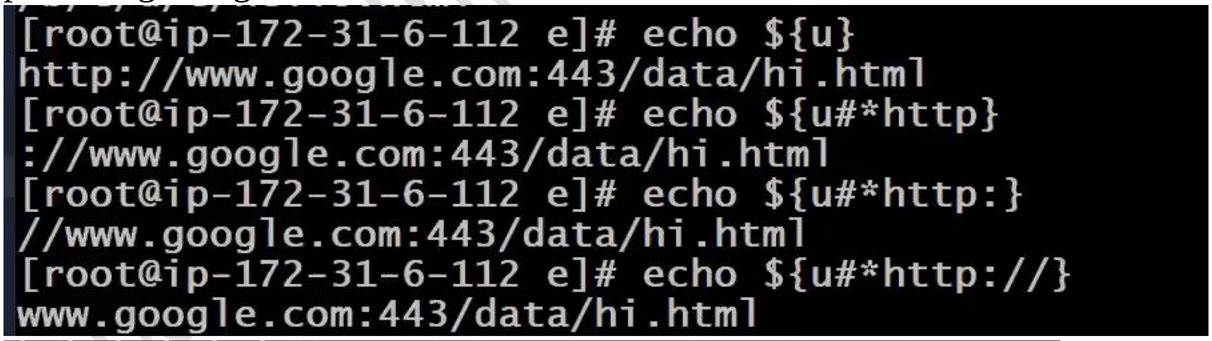
```
[root@ip-172-31-6-112 e]# echo ${p%/*}
/a/b/c/d/e
[root@ip-172-31-6-112 e]# echo ${p%%/*}

[root@ip-172-31-6-112 e]# t="a1/b1/c1/hi.html"
[root@ip-172-31-6-112 e]# echo $t
a1/b1/c1/hi.html
[root@ip-172-31-6-112 e]# echo ${t%/*}
a1/b1/c1
[root@ip-172-31-6-112 e]# echo ${t%%/*}
a1
[root@ip-172-31-6-112 e]#
```

- So one % goes till first / from the suffix side and removes whatever after it and %% goes till last / and removes whatever after it and returns

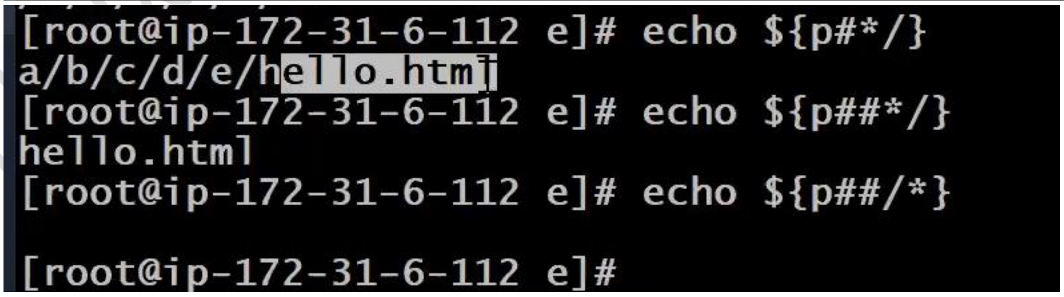


```
[root@ip-172-31-6-112 e]# u="http://www.google.com:443/data/hi.html"
[root@ip-172-31-6-112 e]# echo ${u}
http://www.google.com:443/data/hi.html
[root@ip-172-31-6-112 e]# echo ${u%/*}
http://www.google.com:443/data
[root@ip-172-31-6-112 e]# echo ${u%%/*}
http:
```

- And # symbol do the same thing as % and ## same as %% but from prefix beginning.

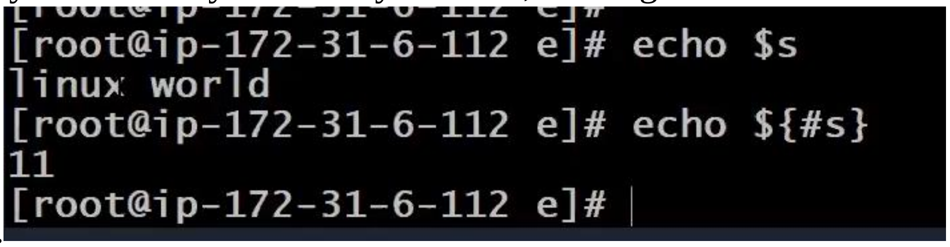


```
[root@ip-172-31-6-112 e]# echo ${u}
http://www.google.com:443/data/hi.html
[root@ip-172-31-6-112 e]# echo ${u#*http}
://www.google.com:443/data/hi.html
[root@ip-172-31-6-112 e]# echo ${u#*http:}
//www.google.com:443/data/hi.html
[root@ip-172-31-6-112 e]# echo ${u#*http://}
www.google.com:443/data/hi.html
```



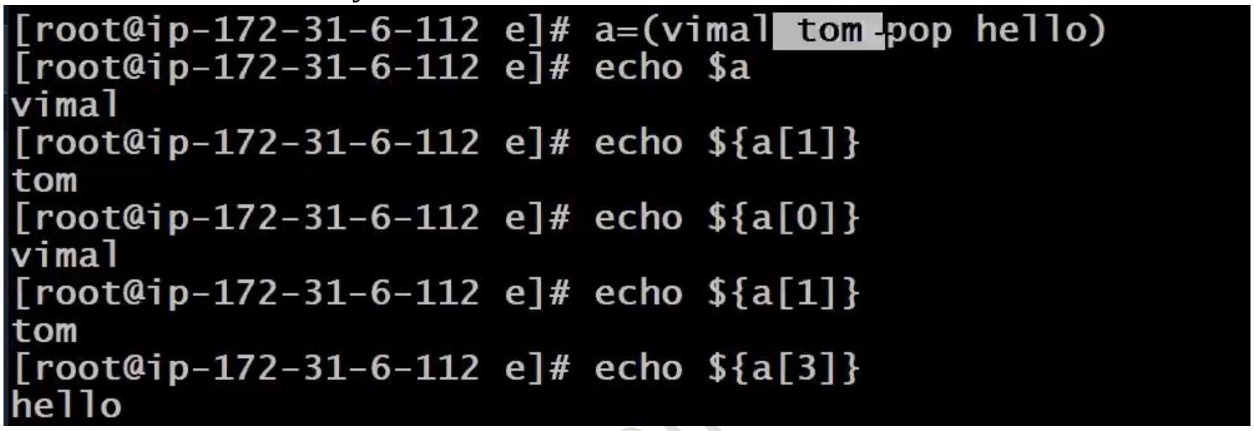
```
[root@ip-172-31-6-112 e]# echo ${p#*/}
a/b/c/d/e/hello.html
[root@ip-172-31-6-112 e]# echo ${p##*/}
hello.html
[root@ip-172-31-6-112 e]# echo ${p##/*}

[root@ip-172-31-6-112 e]#
```

- And if you write only # with any variable, it will give the number of words.



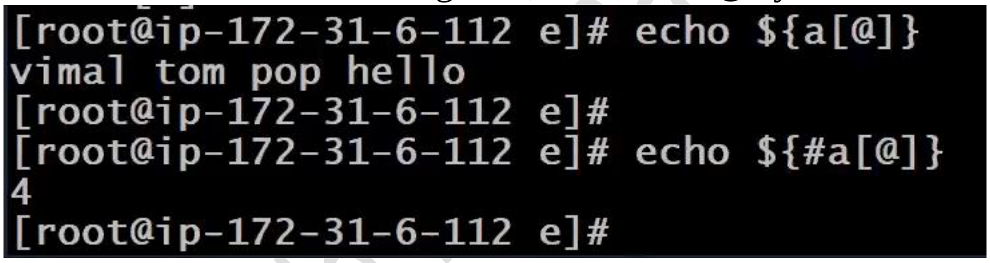
```
[root@ip-172-31-6-112 e]# echo $s
linux world
[root@ip-172-31-6-112 e]# echo ${#s}
11
[root@ip-172-31-6-112 e]#
```

- We can also create arrays in the shell.



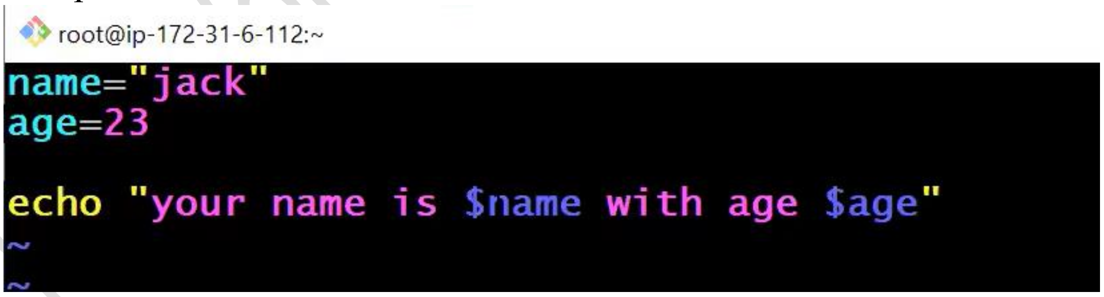
```
[root@ip-172-31-6-112 e]# a=(vimal tom pop hello)
[root@ip-172-31-6-112 e]# echo $a
vimal
[root@ip-172-31-6-112 e]# echo ${a[1]}
tom
[root@ip-172-31-6-112 e]# echo ${a[0]}
vimal
[root@ip-172-31-6-112 e]# echo ${a[1]}
tom
[root@ip-172-31-6-112 e]# echo ${a[3]}
hello
```

- And to return all the items together, we use the @ symbol.



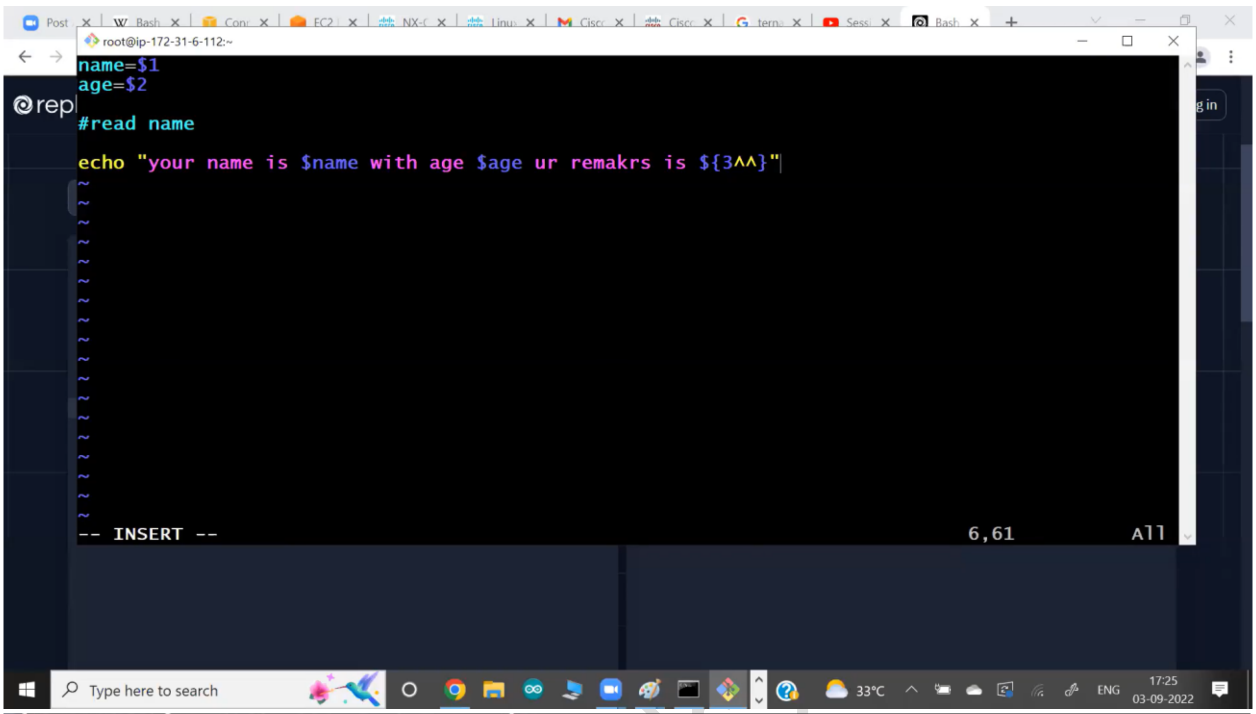
```
[root@ip-172-31-6-112 e]# echo ${a[@]}
vimal tom pop hello
[root@ip-172-31-6-112 e]#
[root@ip-172-31-6-112 e]# echo ${#a[@]}
4
[root@ip-172-31-6-112 e]#
```

- The concept of using the variables between the string is known as 'String Interpolation'

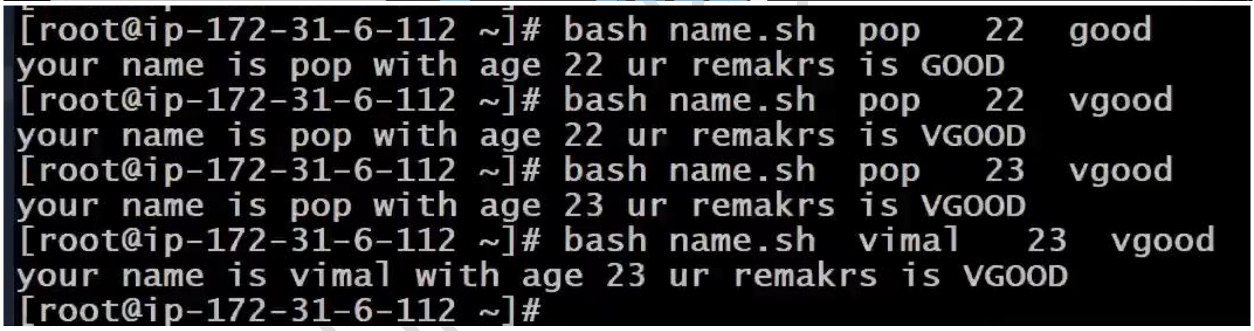


```
root@ip-172-31-6-112:~
name="jack"
age=23

echo "your name is $name with age $age"
~
~
```

- Using the read option for the input prompt is not a good practice in the real world because the script is running in another system most time or with the help of some third-party tool. So instead we prefer to take the input from arguments while running the script.
- When we write any argument in the command, it is stored in the special variable and that is 1. The first argument is stored in 1 variable and second variable in 2 variables and so on

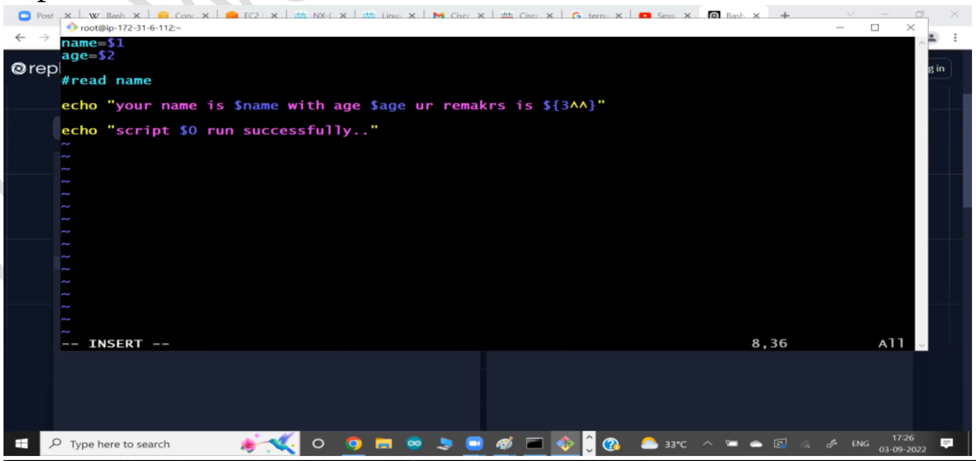


```
name=$1
age=$2

#read name

echo "your name is $name with age $age ur remakrs is ${3^^}"
~
-- INSERT --                                                6,61          All
```

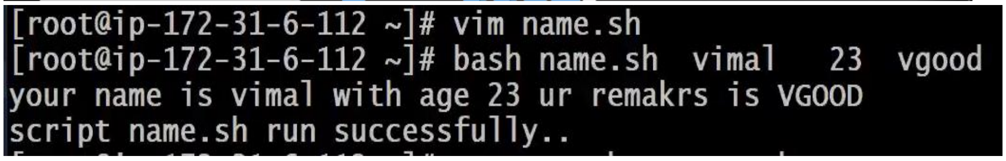


```
[root@ip-172-31-6-112 ~]# bash name.sh  pop   22  good
your name is pop with age 22 ur remakrs is GOOD
[root@ip-172-31-6-112 ~]# bash name.sh  pop   22  vgood
your name is pop with age 22 ur remakrs is VGOOD
[root@ip-172-31-6-112 ~]# bash name.sh  pop   23  vgood
your name is pop with age 23 ur remakrs is VGOOD
[root@ip-172-31-6-112 ~]# bash name.sh  vimal   23  vgood
your name is vimal with age 23 ur remakrs is VGOOD
[root@ip-172-31-6-112 ~]#
```

-
-
- Or we can do it in another way also and ^, ^^ symbols turn the data into capital letters.

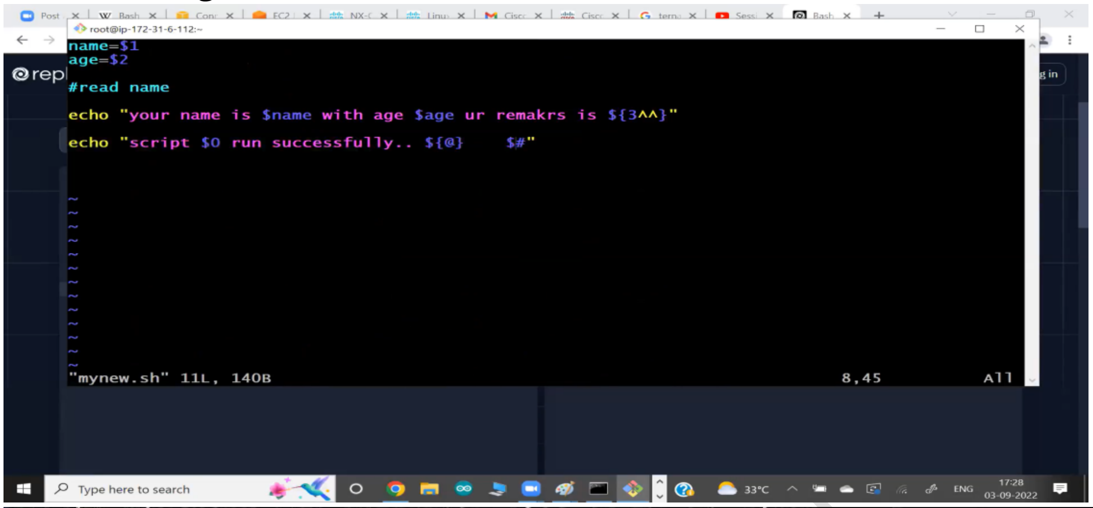


```
name=$1
age=$2

#read name

echo "your name is $name with age $age ur remakrs is ${3^^}"

echo "script $0 run successfully.."
~
-- INSERT --                                   8,36          All
```

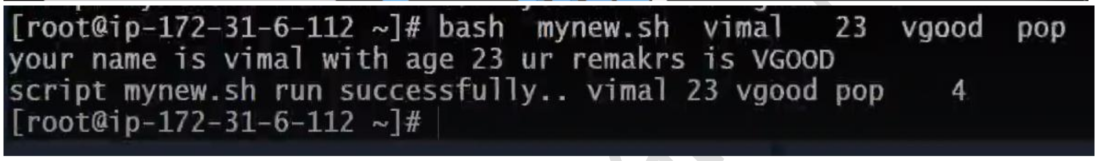


```
[root@ip-172-31-6-112 ~]# vim name.sh
[root@ip-172-31-6-112 ~]# bash name.sh  vimal   23  vgood
your name is vimal with age 23 ur remakrs is VGOOD
script name.sh run successfully..
```

- And all the arguments we give in the command line are stored in a variable '@'. So to print all the arguments we use @ and # will check the number of arguments.

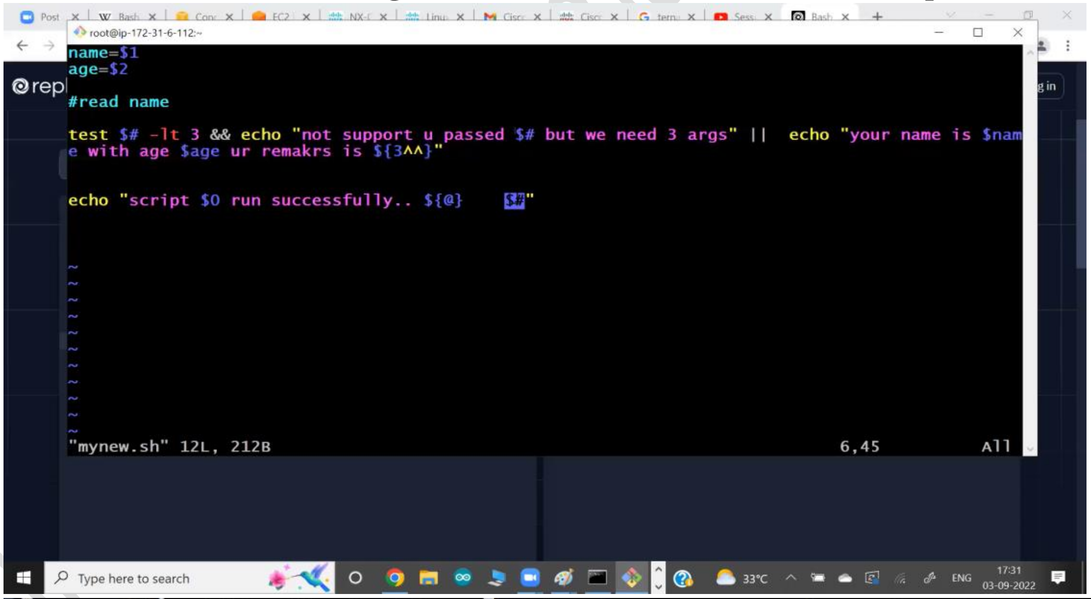


```
name=$1
age=$2

#read name

echo "your name is $name with age $age ur remakrs is ${3^^}"

echo "script $0 run successfully.. ${@}     $#"

"mynew.sh" 11L, 140B                                              8,45          All
```

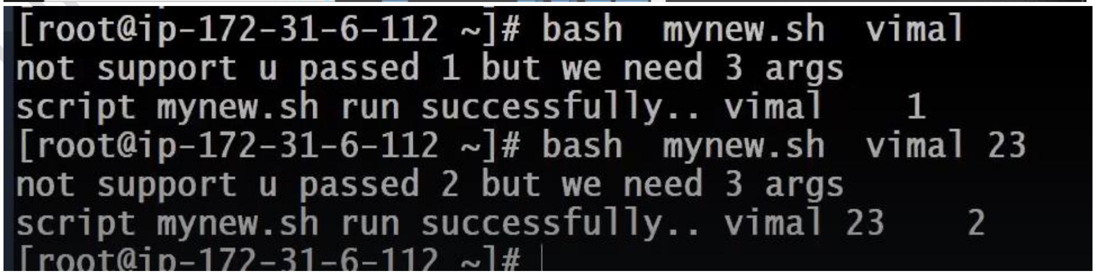


```
[root@ip-172-31-6-112 ~]# bash  mynew.sh  vimal   23  vgood  pop
your name is vimal with age 23 ur remakrs is VGOOD
script mynew.sh run successfully.. vimal 23 vgood pop     4
[root@ip-172-31-6-112 ~]#
```

- So based on this knowledge, we can also create one more script.

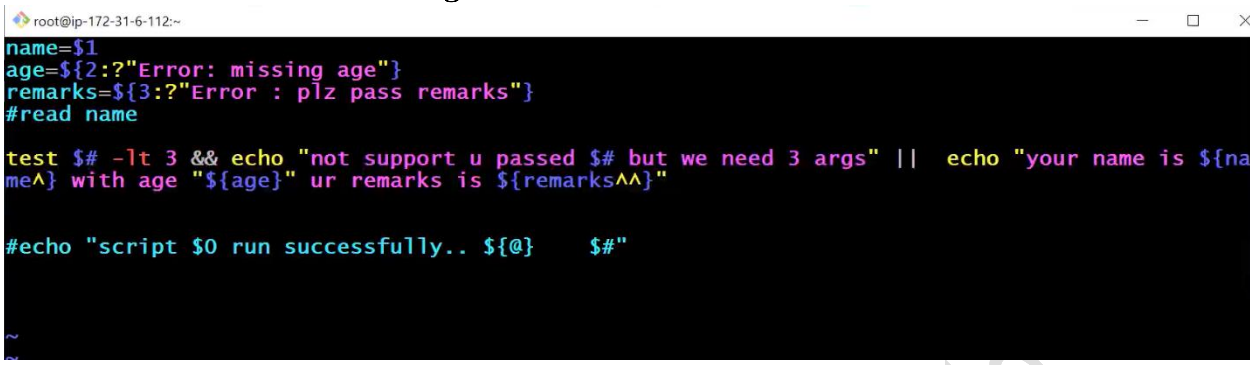


```
name=$1
age=$2

#read name

test $# -lt 3 && echo "not support u passed $# but we need 3 args" ||  echo "your name is $nam
e with age $age ur remakrs is ${3^^}"

echo "script $0 run successfully.. ${@}     $#"

"mynew.sh" 12L, 212B                                              6,45          All
```



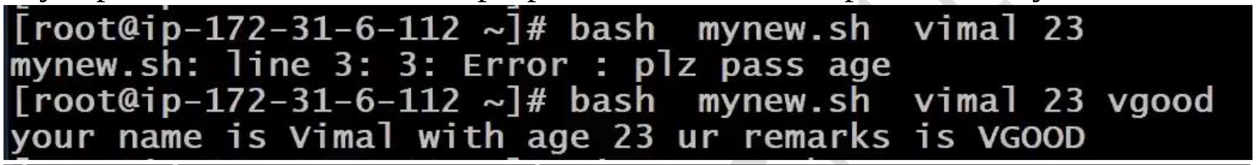
```
[root@ip-172-31-6-112 ~]# bash  mynew.sh  vimal
not support u passed 1 but we need 3 args
script mynew.sh run successfully.. vimal     1
[root@ip-172-31-6-112 ~]# bash  mynew.sh  vimal 23
not support u passed 2 but we need 3 args
script mynew.sh run successfully.. vimal 23     2
[root@ip-172-31-6-112 ~]#
```

- We can also use our knowledge now and make it better.

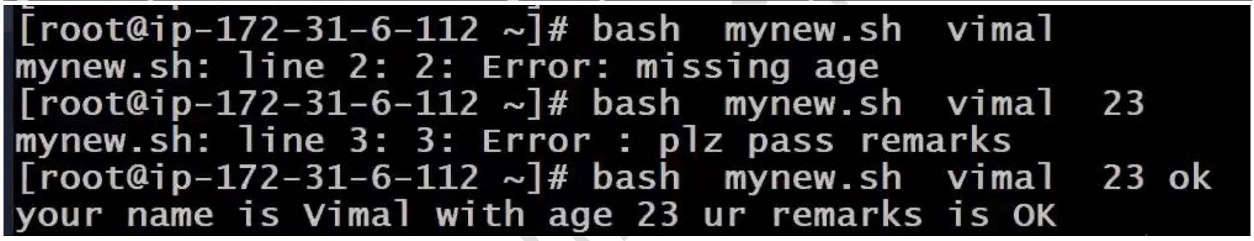


```
name=$1
age=${2:?"Error: missing age"}
remarks=${3:?"Error : plz pass remarks"}
#read name

test $# -lt 3 && echo "not support u passed $# but we need 3 args" ||  echo "your name is ${na
me^} with age "${age}" ur remarks is ${remarks^^}"


#echo "script $0 run successfully.. ${@}    $#"
```

- Now with this code, we will pass proper validation that if we miss to pass any input then it will return the proper error that will help us to rectify.



```
[root@ip-172-31-6-112 ~]# bash  mynew.sh  vimal 23
mynew.sh: line 3: 3: Error : plz pass age
[root@ip-172-31-6-112 ~]# bash  mynew.sh  vimal 23 vgood
your name is Vimal with age 23 ur remarks is VGOOD
```



```
[root@ip-172-31-6-112 ~]# bash  mynew.sh  vimal
mynew.sh: line 2: 2: Error: missing age
[root@ip-172-31-6-112 ~]# bash  mynew.sh  vimal  23
mynew.sh: line 3: 3: Error : plz pass remarks
[root@ip-172-31-6-112 ~]# bash  mynew.sh  vimal  23 ok
your name is Vimal with age 23 ur remarks is OK
```