**Shell Script:**

To export a path as system path:

Ex: if we want to make a path as system path

$ export PATH=${PATH}:/home/ec-2user/usercmds/bin

So the default which also look into this folder also, if any script present in the location it will run from command line.



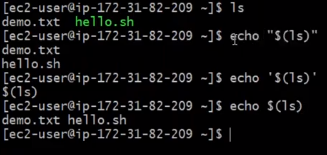
But is it temporary if we want make it permeant

Change it in the .bashrc

And then run

$source .bashrc (it’s a permanent change)

**User required text print**



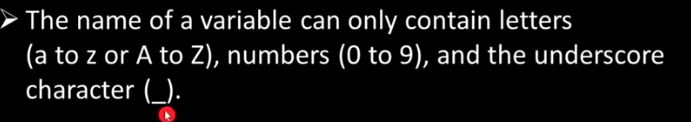
If we want to print the ls output through echo message then we can use $ echo “(ls)”

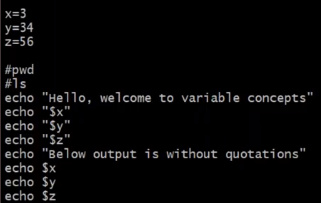
$ echo ‘(ls)’

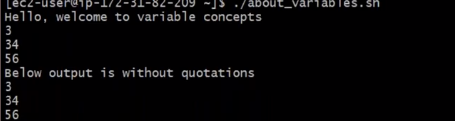
$ echo $ls ### there is variation in the outputs check the about command.

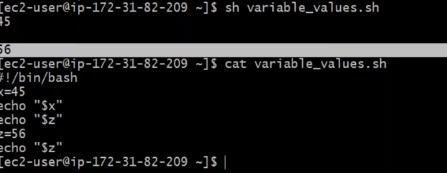
* If we use command in the double quotes there is different if we use txt in the double quote there will be different output.

**Variables:**

****

****

****

****

**By default if there is no variable values then it will show blank space. No space while defining the variable value**

**If we add \n then we get output in different line**

**Variable shellnot start with number**

To print date.log with variable

todays\_date=$(date +”%Y-%m-%d”)

log\_file\_name=$todays\_date

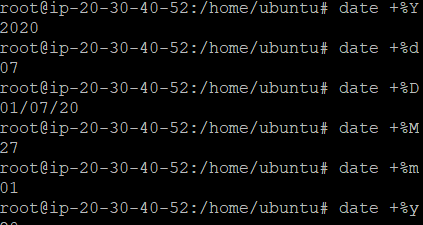
touch ${log\_file\_name}.log

store the variable into file system,memory.

If you define your variable in the shell script they are ony vaild with in script.

If command line then until computer restart.

It will be permanent in .bashrc or bashprofile

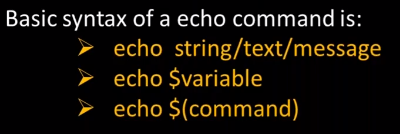


The “env” command will display all the variables in the system

Special variable : echo $?

If we get the output as 0 last command is successfully some thing else then the last command is error.

**Echo command**



-n = it will append the next line to it

Ex: my\_name=$ashok

echo -n "your name $my\_name"

echo -n "($pwd)"

echo $(pwd)

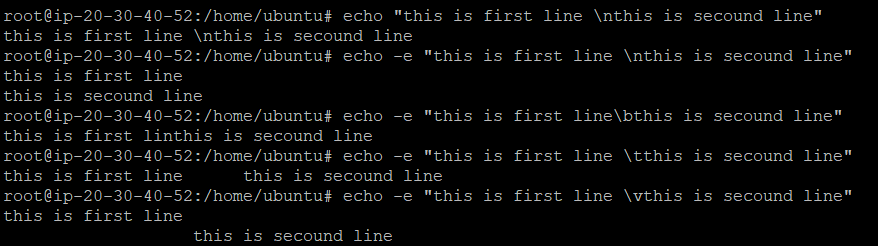
output: 

observation: when declaring command in echo then don’t use double quote.

-n: it will link the next command output to self line.

\n: basically it will create a new line but if we use in shell

Then it will not work.



\b: one back move of the cursor

\v: vertical of the text

1. **Input and output commands**

echo 🡪 output command

read 🡪 input command

(command line arguments as a input.)

1. **Awk command**

Awk systax

$Awk options ‘pattern/condition {action}’ filename

$command | awk options ‘pattern/condition {action}’

Options:

-F fs to specify a filed separator

-v var=value To declare a variable

-f file To specify a file that contains awk script

$awk –f awk\_script.awk filename

$awk ‘BEGIN {print “your file of /etc/passwd”} /root/ {print $0} END {print “file printed”}’ /etc/passwd

BEGIN {

print "test"

}

/root/ {

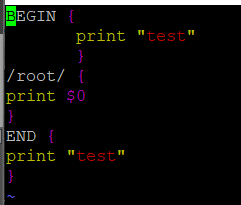
print $0

}

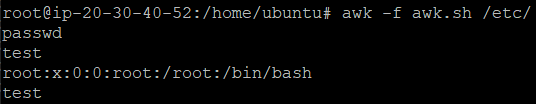
END {

print "test"

}



$awk –f awk.script /etc/passwd

Output: 

To initialize a variable and print the variable.

$ awk 'BEGIN { a=5 ; print a }' # when you write a script no need to provide ; insisted of that you have to provide \n

$echo “2 6” | awk ‘ { print “a=” $1 , “b=” $2 } ’

Output : a=2 b=6 # here 2 treated as 1 and 6 treated as 2 as place values

\* By default awk treated your records as fields.

\* shebang line for aws script is #!/bin/awk # use which awk output

\* in awk script no need to write $a symbol in the command it will automatically fetch the value of a

**Using awk in shell script:**

**#!/bin/bash**

**pwd**

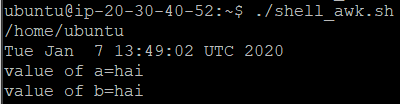
**date**

**a=$(awk 'BEGIN { print "hai" }’ )**

**b=$(awk 'BEGIN { print "hai" }' )**

**echo "value of a=$a"**

**echo "value of b=$b"**

Output: 

1. **How to read variables for awk script**

Some commands usage with awk

1. Awk –F ‘:’ ‘{ print $1 }’ /etc/passwd 🡪 here : this will work as a separator.
2. **Cut Command:**