cheat.sh is the scripting cheatsheet

SHELL SCRIPTING:

It is a series of commands put together in a text file and execute them. Instead of executing every time

* It is not mandatory that always shell scripting file ends with a .Sh extension but it will tell u on seeing that it is a clear shell file
* Shell script written in one shell doesn’t run in other shell.
* System related things, daily tasks, automation we go with scripting, where as we will go for programming for business applications

SHELL:

A shell is a command line interpreter which takes the command from users and execute them.

It acts as an interface between kernel and user. Since we always work on CLI. Cli is present in shell

(Interpreter means executing line by line.)

TYPES OF SHELL:

There are three types of shell

1. C shell
2. Korn shell
3. Bourne shell
4. Power shell

The latest shell is BASH (Bourne Against shell) which is most widely used and efficient.

Shell prompt for bash shell is sh

USES OF SHELL SCRIPTING:

1. Automating repetitive tasks
2. Scheduling multiple jobs at a time
3. Customizing your work environment like displaying “hey” when you logged in every time.
4. Save lot of time

Q) How to know which shell is using?

# echo $0

#! Is called shebang

It Is used to pass instruction to program **/bin/sh.**

**/bin/sh is the location of intrepeter**

HOW TO WRITE SHELL SCRIPT:

1. Open a file using vi editor with an extension. Ex: vi file1.sh
2. Start the script with shebang **#! /bin/sh**
3. Write comment with hash mark Ex: #display names
4. Write commands line by line Ex: who
5. Close vi editor by pressing EX:(esc +: +wq)
6. Now run the file. Ex: bash file1.sh (or) source file1.sh (or) ./file1.sh
7. Change permissions if required Ex: chmod u+X file1

If your script is in /home/mani/Desktop and you are in (/) then you can execute script by ./home/mani/Desktop/file1.sh

(or) source /home/mani/Desktop/file1.sh

(or) bash /home/mani/Desktop/file1.sh

SHELL SCRIPT SYNTAX:

Command option input

Ex:- cat –n filename, ls –l /home

SHELL VARIABLES:

A variable duty is to store the value. ex: a=20

1. System variables:

These are predefined variables

Ex: echo, Pwd

1. User-defined variables:

These will be defined by user

Ex: a=20

* Variable name should not start with number

{{{{{{Parameters are the variable which hold values

Argument are the values which are passed to parameters

Ex:- method (int x , int y)-----🡪parameters

Int x=25;------🡪argument1

Int y =40;---------🡪argument2

Arguments are passed during execution of program}}}}}}

1. *How to print a name?*

vi file.sh

#!

#printing a name in shell scripting

clear

var1=mark

var2=john

echo $var1 $var2

Esc+: wq

# bash file1.sh

2. *How to read user input from keyboard*?

“read” commands take input from keyboard and assign it to a variable

read is system variable

Ex: read variable (or) read name (or) read value

vi file2.sh

echo enter name of a person

read nam or read nam1 nam2 nam3

echo “entered person name is : $nam”

or echo “entered person name is : $nam1, nam2, nam3”

*-a flag for array:*

read –a names (where a flags stands for array)

echo “names : ${name[0]} , ${name[1]}”

*-p flag to print in same line:*

read -p “username :” user\_var

echo “username : $user\_var”

*-sp flag to hide text:*

This is used when typing password

read –sp password

echo “enter password: $password”

3. *How to pass an arguments*?

$0 returns script name/ also bash name

$1-$9 return whatever we pass

Beyond $9 we have to use ${10} and ${11} and so on ${n}

$$ returns pid of script we are running

$# returns arguments we have passed

$\*

$@

$n

$!

$\_

$? The status of last executed command whether it is successful or not (if returned 0 then last command executed successfully)

Ex:-

vi file7.sh

#! /bin/bash

#passing an argument

echo “the script name is: $0”

echo the first argument is :$1

echo the second argument is :$2

echo total arguments are:$#

echo status of last command:$?

running script as follows:

scriptname arg1 arg2

ex:- ./file7.sh 3 5

o/p:-

“the script name is: file7.sh”

the first argument is :3

the second argument is :5

total arguments are:2

if you have 11 arguments and don’t like to type 11 lines every time then you can go for array.

arg=(“$@”)

echo ${arg[0]} ${arg[1]} ${arg[2]}…………………………. ${arg[n]}

or $@

*Integer comparisons*:

1) -gt = greater than if [“$a” -gt “$b”]

2) -ge= greater than or equal to if [“$a” -ge “$b”]

3) -lt= less than if [“$a” -lt “$b”]

4) -le= less than or equal to if [“$a” -le “$b”]

5) -eq=equal to if [“$a” -eq “$b”]

6) -ne=not equal to if [“$a” -ne “$b”]

If you use >,>= (or) <=, < then you have to use like this ex: ((“a” <= “b”)) in integer comparison

*String comparisons*:

1) = equal to if [“$a” = “$b”]

2) != not equal to if [“$a” != “$b”]

3) > greater than if [[“$a” > “$b”]]

4) < less than if [[“$a” < “$b”]]

If you use >,>= (or) <=, < then you have to use like this ex: [[“a” <= “b”]]

4. *if then fi syntax*

if [ conditional expression ]

then

statement1

statement2.

fi

Now let us take an example

vi file9.sh

#! /bin/bash

#if then fi

clear

echo "enter count value to check if it is greater than 9"

read count

if [ $count -gt 9 ]

then

echo "yes $count is greater than 9"

fi

5. *if else fi syntax:-*

If [ conditional expression ]

then

statement1

statement2

else

statement3

statement4

fi

vi file10.sh:-

#! /bin/sh

#comparison of strings

clear

read var1

read var2

if [ "$var1" = "$var2" ]

then

echo "condition is true"

else

echo "condition is false"

fi

6. *if elif else fi syntax:-*

If [conditional expression1]

then

statement .

elif [ conditional expression2 ]

then

statement

else

statement

fi

vi file11.sh:

#! /bin/sh

#if elif else fi

clear

read var0

read var1

read var2

if [ $var0 = $var1 ]

then

echo "condition is true"

elif [ $var0 = $var2 ]

then

echo "condition elif is true"

else

echo "condition is false"

fi

7. *File test operator(testing a file)*?

File test operator is to check if file exists or not and it is a regular file or block file (img,binary,etc) or character file(data ,text)

So there are certain flags which does that

1. \c to keep the curson on same line
2. –e to check file exists or not
3. –f to check if it is regular file or not
4. –s checks if file size is greater than zero then condition is true
5. –b checks if it is block file or not
6. –c checks if it is character device file or not
7. –r checks if a file has read permission
8. –w checks if a file has write permission
9. –x checks if a file has executable permission or not
10. –O to check owner of a file
11. –u to check setuid of a file
12. –g to check setgid of a file
13. –G groupid of a file
14. –k if stickybit is applied on a file
15. –L to check symbolic link of a file

vi file12.sh:

#! /bin/bash

#file test operator

echo -e "enter filename :\c"

read file\_name

if [ -e $file\_name ]

then

echo "yes file esists"

else

echo "no file doesnt exists"

fi

7. *How to add or append data to the file in output*?

vi file13.sh

#! /bin/bash

#appending data to the output file at end

echo -e "enter filename :\c"

read file\_name

if [ -f $file\_name ]

then

if [ -w $file\_name ]

then

echo "yes file has write permission enter if you want some data to be appended"

cat >> $file\_name

else

echo "no file doesnt have write permission"

fi

else

echo "$file\_name doesnt exists"

fi

8. *Logical operators:-*

(i) &&--------------------it is used for two or more conditions

And returns true if all conditions are true

Ex:- age =25

if [ “$age” –gt 18 ] && [ “$age” –lt 50]

since 25 is in between 18 and 50 it satisfied both the conditions

vi file14.sh:

#! /bin/bash

#logical and operator

clear

echo "enter the age"

read age

if [ "$age" -gt 18 ] && [ "$age" -lt 50 ]

then

echo "yes entered age is right and condition is satisfied"

else

echo "entered age is wrong and condition is not satisfied"

fi

(ii) ||---------------------- it is used for two or more conditions and return true if one of the condition satisfies

Ex:- age =60

if [ “$age” –gt 18 ] || [ “$age” –lt 50]

here 60 >18 so first condition satisfied so it returns true

vi file15.sh:

#! /bin/bash

#logical or operator

clear

echo "enter country name"

read country

if [ "$country" = japan ] || [ "$countrty" = america ]

then

echo "$country gets indian citizenship"

else

echo "$country wont get indian citizenship"

fi

9. *Arithmetic operators:-*

+ is addition

- is subtraction

\* is multiplication

/ is division

% is modulus

Usage:-

echo $(( num1 + num2))

vi file16.sh:

#! /bin/bash

#arithmetic operators

read num1

read num2

echo "sum of two num is: $(( num1 + num2 ))"

echo "the diff of two numbers is: $(( num1 - num2 ))"

echo "the prod of two numbers is: $(( num1 \* num2 ))"

echo "the div of two numbers is: $(( num1 / num2 ))"

echo "the mod of two numbers is: $(( num1 % num2 ))"

10. *Arithmetic operators for decimals:-*

Arithmetic operators doesn’t work in case of decimals so we use a command “bc” which stands for basic calculation.

Usage:

echo “$num1 + $num2” | bc

vi file17.sh:

! /bin/bash

#arithmetic operators for decimals

clear

read num1

read num2

echo "$num1+$num2" | bc

11. *Case statement:-*

It is used to simplify nested if statements

Usage:-

read variable

case $variable in

pattern1)

Statement1;;

pattern2)

Statement2 ;;

pattern3)

Statement3;;

.

.

.

\*)

Statement default ;;

esac

in the above pic it reads a variable if the variable matches pattern1 it executes statment1 else go to next pattern and so on

if you enter invalid option it executes default statement

vi file18.sh:

read fruit

case $fruit in

"apple")

echo "Apple pie is quite tasty." ;;

"banana")

echo "I like banana nut bread." ;;

"kiwi")

echo "New Zealand is famous for kiwi." ;;

\*)

echo "entered invalid option" ;;

esac

vi file19.sh:

#! /bin/bash

#case 2 example

clear

echo "enter a character numeric or speacial character"

read value

case $value in

[a-z] )

echo "enterd value is small letters" ;;

[A-Z] )

echo "entered value is capital letters" ;;

[0-9] )

echo "entered value is numeric" ;;

? )

echo "entered value is special character" ;;

\* )

echo "invalid input"

esac

\*\*\*\*\*if caps doesn’t work type on terminal LANG=C

12. *Array variables:-*

Bash supports simple one dimensional array

ARRAYNAME=( ‘element1’ ‘element2’ ‘element3’)

element1 at 0th index position

element2 at 1st index position

element3 at 2nd index position

array length is 3

1. To print all elements of an array?

echo "${ARRAYNAME[@]}"

1. To print an element at particular index?

echo "${os[x]}" where x=index number

1. To print all the array indexes?

echo "${!os[@]}"

1. To print the array length?

echo "${#os[@]}"

1. To add an array at nth index?

ARRAYNAME[n]='xxxxx'

1. To remove nth element in an array?

unset ARRAYNAME[n]

you can also add string to an array

string=manikanth

echo “${string[0]}”

in this case there will be only one index position ie…, 0th position

vi file20.sh:

#!

#simple 1d array

os=( 'windows' 'mac' 'linux')

os[3]='solaris'

unset os[2]

echo "it prints elements of an array"

echo "${os[@]}"

echo "it prints element at that index"

echo "${os[1]}"

echo "if there are n number in an array it prints n number of indexes"

echo "${!os[@]}"

echo "to print how many elements are there in a array"

echo "${#os[@]}"

echo "adding elements in (n+1)array"

echo "to remove an element from particular index"

13. *How to display contents of a file:-*

vi file22.sh:

#! /bin/bash

#read a file

echo "enter a filename to display its content"

read file\_name

cat "$file\_name"

o/p:-

enter the full path of an file you want to see the content

1.while loop:-

Loops are used to execute statements repeatedly.

while [ condition ]

do

Statement1

Statement2

done

if the while condition is true the statements gets executed.

Loop terminates if the condition is false

vi file21.sh:

#!

#while loops

clear

i=1

while [ $i -le 10 ]

do

echo "hi $i"

((i++))

done

2. until loop:-

Until loop is similar to while loop but while loop gets executed if condition is true

But until loop executes if condition is false

until [ condition ]

do

Statements

Or

commands

done

vi file23.sh:

#!

#until loops

clear

i=1

until [ $i -gt 10 ]

do

echo "hi $i"

((i++))

done

3.for loop:-

for loop is used to execute the statements sequentially according to the condition given

if value is specified from 1 to 10 then it will execute from 1st statement to 10th statement

(I)

for variablename in list

do

command1

command2

done

here list means any numbers , words , files ,etc

(II)

for variablename in command1 command2 ............

do

$variablename

done

(III)

for (( expr1; expr2; expr3 ))

do

command1

command2

done

(IV)

for variablename in file1 file2 ………filen

do

command1 on $variablename

command2 on $variablename

done

(IV)vi file24.sh:

#! /bin/bash

#for loop2

for var in f1 f2

do

cat $var

done

(II)vi file24.sh:-

#! /bin/bash

# for loop

for var in pwd who ls

do

echo "----------$var------------------"

$var

done

1. vi file26.sh:-

#! /bin/bash

#for loop3

for fruit in apple banana orange mango

do

echo "$fruit"

done

1. vi file27.sh:-

#! /bin/bash

#for loop3

for (( i=1; i<=5; i++ ))

do

echo $i

done

4.break:-

Break is used to terminate or come out of the loop if the specified condition matches.

vi file25.sh:-

#! /bin/bash

#break

for (( i=1; i<=10; i++ ))

do

if [ $i -gt 5 ]

then

break

fi

echo $i

done

here if i>5 then iteration comes out of loop and thus stops

5.continue:-

continue means skipping the condition provided

vi file26.sh:-

#! /bin/bash

#continue

for (( i=1; i<=10; i++ ))

do

if [ "$i" -eq 5 ] || [ "$i" -eq 10 ]

then

continue

fi

echo $i

done

here the iteration skips for 5 and 10.

6.functions:

Using functions we can break down the code into smaller logical parts. so we can easily check program part by part. We can use where ever we want

Syntax:-

function function\_name () {

commands

}

function\_name arguments

(or)

name () {

commands

}

now the arguments will be passed into function\_name

vi file27.sh:-

#! /bin/bash

#functions

function name(){

echo $1 $2

}

name tom bob

here tom will be passed into name function of argument1($1)

and bob will be passed into name function of argument2($2)

7.local and global variables:

Local variables are defined inside a function where as global variables are defined out side the function

Syntax:-

function print () {

lname=$1---------------------------------🡪 local variable

}

gname=“tom”--------------------------------------🡪global variable

echo “the name is : $gname”

print max

here inside print function name is local variable ,$1 value will be stored in name

print max means max will be stored inside print function $1

but tom value will be stored in global variable of name and it echos tom name

first tom name will be printed then print function will be called and prints max

note:-

to avoid confusing about local and global variables then declare local before local variable as follows

function print () {

local lname $1---------------------------------🡪 localkeyword local variable

}

gname=“tom”--------------------------------------🡪global variable

echo “the name is : $gname”

print max

if you declare local keyword before local variable them it is restricted to that function

in this case lname is restricted to print function

8.read only:

read only command used for functions and variables. once declared read only to a variable ,and we cant over write .

syntax:-

readonly var

(or)

Readonly –f functionname

vi file29.sh:-

#! /bin/bash

#readonly

var=80

readonly var

var=50

echo $var

function hello() {

echo "Hello world"

}

hello

readonly -f hello

function hello(){

echo "Hello world again"

}

In the above example we declared var=80 and made final by declaring readonly to var even if we assign 50 to var and gives echo we still see var=80

9.debugging a script:

If you want to know where script is going wrong you can use this debugging feature and fix that bug

Syntax:-

bash –x ./filename.sh

(or)

set –x-------------------------------🡪here is the point where debugging starts

code

code

code

Set +x-----------------------------🡪 here is the point where debugging stops

vi file30.sh:

#! /bin/bash

#debugging

set -x

read a

read b

if [ "$a" -eq "$b" ]

then

echo "both numbers are same"

else

echo "both numbers are different"

fi

set +x

(or)

bash -x file30.sh

10.executing multiple script:

If there are two shell script files and you want execute them sequentially then put 2nd script in first script as follows

vi file1.sh vi file2.sh

#! /bin/bash #! /bin/bash

echo “hai” echo “gud mrng”

bash file2.sh

now run the first script

#bash file1.sh

o/p:-

hai

gud mrng

SCRIPT FOR SENDING A MAIL TO NOTIFY IF SOMETHING IS FAILED OR NOT:-

if [ $? -ne 0 ]

then

<send me email however you want to about failure>

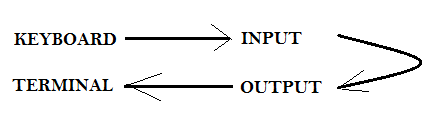
else

<send me email about success how ever you want>

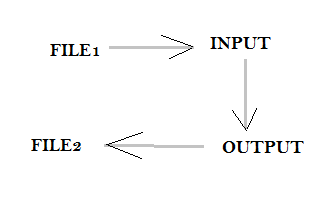
fi

REDIRECTIONS:-

Generally we give input from keyboard and output will be displayed on terminal



We can replace keyboard and terminal with files



I have a file where I put uptime command inside it, the result will be displayed in file2

This way is called STDIN (standard input) and STDOUT (standard output)

1. STDIN is represented as <

Ex:- mail -s subject [mani@gmail.com](mailto:mani@gmail.com) <file1

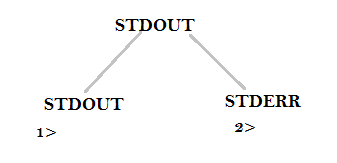
Now file1 content will be sent to mail

1. STDOUT is represented as >

Ex: - ls >file2

Now the result of ls will be stored in file2

Stdout is again classified into two types



2>&1 means both error and output will go into a single file

>/dev/null means your output will be thrashed or store in a null file

COMMUNICATION:-

1. mail=used to send mail
2. wall=it send message to all users

ex:- wall “stop all your work”

DELIMITERS:-

Which separate fields

Ex: - cat /etc/passwd

Mani:x:0:0:mani:mani:/sbin/login

Here : is called delimiter