**SECTION: 11**

**Arithmetic Operators :**

* Shell script variables are by default treated as strings, not numbers, which adds some complexity to doing math in shell script.
* There are different ways to perform arithmetic operations.
  + Using declare
  + Using expr
  + Using let
  + **Using (()) (For integers)**
  + **Using bc (For integers and float numbers) base calculator**

**Example 1: Using (())**

x=

y=5

((sum=x+y))

echo "$sum"

12

((sum=x+10))

echo "$sum"

17

((sum=2+10))

echo "$sum"

12

((sub=x-y))

echo "$sub"

2

((x++))

echo "$x"

8

echo "$((x=4+3))"

7

**Example 2: Using bc ---🡪to get floating point numbers.**

x=2

y=3.5

bc<<<"$x+$y"

5.5

x=6

y=745

bc<<<”scale=1;$y/$x” ----🡪to get floating point numbers.

124.1

x=6

y=745

bc<<<”scale=2;$y/$x” ----🡪to get floating point numbers.

124.16

Simple Shell Script using Arithmatic operators:

#!/usr/bin/bash

X=5

Y=89

((sum=x+y))

echo “The addition of $X and $Y is : $sum”

Result=$(bc<<<”scale=2;$Y/$X”)

echo “The division of $Y with $X is : $Result”

echo “The addition of $5 and $89 is : 94”

echo “The division of $89 with $5 is : 17.800”

#!/usr/bin/bash **Input and output commands**

read -p "Enter your 1st value : " X

read -p "Enter your 2nd value : " Y

((sum=x+y))

echo “The addition of $X and $Y is : $sum”

#!/usr/bin/bash **Input and output commands**

read -p "Enter your 1st value : " X

read -p "Enter your 2nd value : " Y

Result=$(bc<<<”scale=2;$Y+$X”)

echo “The addition of $Y with $X is : $Result”

#!/usr/bin/bash **Input and output commands**

read -p "Enter your 1st value : " X

read -p "Enter your 2nd value : " Y

Result=$(bc<<<”scale=2;$Y/$X”)

echo “The division of $Y with $X is : $Result”

**Case statement :**

|  |
| --- |
| Syntax:  case $var in  opt1)  statements  ;;  Opt2)  statements  ;;  \*)  statements  ;;  esac |

EXAMPLE1:

#!/usr/bin/bash **Input and output commands** simple\_cal.sh

read -p "Enter your num1 : " a

read -p "Enter your num2 : " b

read -p "Select your option 1)Add, 2)Sub, 3)Mul, 4)Div : " opt

case $opt in

1)

echo “you selected Addition”

echo "The addition of $a and $b is: $((a+b))"

;;

2)

echo “you selected Subtraction”

echo "The sub of $a and $b is: $((a-b))"

;;

3)

echo “you selected Multiplication”

echo "The mul of $a and $b is: $((a\*b))"

;;

4)

echo “you selected Addition”

echo "The div of $a with $b is: $((a/b))"

;;

\*)

echo "Your entered invalid option"

;;

esac

./simple\_cal.sh

Enter your num1 : 3

Enter your num2 : 6

Select your option 1)Add, 2)Sub, 3)Mul, 4)Div : 1

You selected Add

The addition of 3 and 6 is : 9

[or]

EXAMPLE2: **Input and output commands**

#!/usr/bin/bash

clear

read -p "Enter num-1: " a

read -p "Enter num-2: " b

echo "==========MENU==============="

echo " 1. Addition"

echo " 2. Sub"

echo " 3. Mul"

echo " 4. Div"

echo "========================="

read -p "Enter your option(1-4): " opt

case $opt in

1)

echo "The addition of $a and $b is: $((a+b))"

;;

2)

echo "The sub of $a and $b is: $((a-b))"

;;

3)

echo "The mul of $a and $b is: $((a\*b))"

;;

4) echo "The div of $a with $b is: $((a/b))"

;;

\*)

echo "Your entered invalid option"

;;

esac

./simple\_cal.sh

Enter your num-1 : 3

Enter your num-2 : 6

**Enter your option(1-4): 1**

The addition of 3 and 6 is : 9

EXAMPLE 3: **Input and output commands** pattern\_for\_case.sh

#!/usr/bin/bash

clear

read -p "Enter any number: " a

case $a in

[0-9])

echo "you enterd single number"

;;

[a-z])

echo "you entered lower case alph"

;;

[A-Z])

echo "you enterd upper case alph"

;;

\*)

echo "Unable to identify your input"

;;

esac

./pattern\_for\_case.sh

Enter Any Num : 2

You Entered a Digit

EXAMPLE 4:

#!/usr/bin/bash

clear

read -p "Enter your file extention: " ext

case $ext in

".txt")

ls -lrt \*.txt

;;

".sh")

ls -lrt \*.sh

;;

\*)

echo "Sorry!. You entered invalid file extention"

;;

esac

**SECTION: 12**

**Test Command :**

It is a command to judge conditions.

* Simple Syntax:
* test condition or [ condition ] or [[ condition ]]
* Note: [[ ]] works with bash/ksh/zsh shells.
* It will return exit status as 0 or 1. (echo $? )
* 0 -- Condition is true or test is successful
* 1 -- Condition is false or test is failed
* How to make condition to work with test command ?
* Comparison Operators
* File Test Operators

Comparison Operators with test command :

* Numbers:
* [[ int1 -eq int2 ]] -- It return true if they are equal else false
* [[ int1 -ne int2 ]] -- It return true if they are not equal else false
* [[ int1 -lt int2 ]] -- It return true if int1 is less than int2 else false
* [[ int1 -le int2 ]] -- It return true if int1 is less than or equal to int2 else false
* [[ int1 -gt int2 ]] -- It return true if int1 is greater than int2 else false
* [[ int1 -ge int2 ]] -- It return true if int1 is greater than or equal to int2 else false
* [[ ! int1 -eq int2 ]] -- It reverse the result
* Strings:
* [[ -z str ]] -- It return true if the length of the str is zero else false
* [[ -n str ]] -- It return true if the length of the str is no-zero else false
* [[ str1 == str2 ]] -- It return true if both the strings are equal else false
* [[ str1 != str2 ]] -- It return true if both the strings are equal else false

File test Operators with test command :

* [[ -d file ]] -- It return true if the file/path is directory else false
* [[ -f file ]] -- It return true if the file/path is a file else false
* [[ -e file ]] -- It return true if the file/path is exists else false
* [[ -r file ]] -- It return true if the file/path is readable else false
* [[ -w file ]] -- It return true if the file/path is writable else false
* [[ -x file ]] -- It return true if the file/path is executable else false

Advanced file test operators are listed below:

a : True if the file exists.

b : True if the file exists and is a block special file.

c : True if the file exists and is a character special file.

d : True if the file exists and is a directory.

e : True if the file exists.

f : True if the file exists and is a regular file.

g : True if the file exists and its SGID bit is set.

h : True if the file exists and is a symbolic link.

k : True if the file exists and its sticky bit is set.

p : True if the file exists and is a named pipe (FIFO).

r : True if the file exists and is readable.

s : True if the file exists and has a size greater than zero.

t : True if file descriptor is open and refers to a terminal.

u : True if the file exists and its SUID (set user ID) bit is set.

w : True if the file exists and is writable.

x : True if the file exists and is executable.

O : True if the file exists and is owned by the effective user ID.

G : True if the file exists and is owned by the effective group ID.

L : True if the file exists and is a symbolic link.

N : True if the file exists and has been modified since it was last read.

S : True if the file exists and is a socket.

**Command Chaining Operators :**

This concept is useful to write simple and short shell scripts.

* Chaining of Linux commands means, combining several commands and make them execute based upon the behavior of operator used in between them.
* The different Command Chaining Operators are:
* Semi-colon Operator ;
* Logical AND Operators &&
* Logical OR Operator ||
* Logical AND – OR Operators && ||

Note: Behaviour of operators

* cmd1 ; cmd2 --Run cmd1 and then cmd2, regardless of the success or failure of cmd1
* cmd1 && cmd2 -- Run cmd2 only if cmd1 succeeded
* cmd1 || cmd2 -- Run cmd2 only if cmd1 failed
* cm1 && cmd2 || cmd3 -- Run cmd2 if cm1 is success else run cmd3

Example 1:

ls;pwd;date

folder1 practise project1 project2 project3

/home/ubuntu

Wed Jun 2 11:43:33 UTC 2021

echo $?

0

Example 2:

which docker && docker -version

/usr/bin/docker

Docker version 20.10.8, build 3967b7d

echo $?

0

[or]

Don’t want to display the 1st command o/p,display only 2nd command o/p.

Error into stand o/p place then stand o/p going to nullify

which docker 2>&1 1>/dev/null && docker –v

Docker version 20.10.8, build 3967b7d

[or]

which docker >/dev/null && docker -v

Docker version 20.10.8, build 3967b7d

which java && echo " java is installed"

java is installed

Example 3:

hjgjhgjhghj || ls

hjgjhgjhghj: command not found

folder1 practise project1 project2 project3

echo $?

0

which apache2 || echo "apache2 is not installed"

apache2 is not installed

Example 4:

which apache2 && echo "apache2 installed" || echo "apache2 not installed"

apache2 not installed

***if it is shell script we need to write this much of lengthy script.***

#!/usr/bin/env bash

which docker 2>&1 1>/dev/null

if [[ $? -eq 0]]

then

echo "docker is installed on this host $(hostname -s)"

else

echo "docker is not installed on this host $(hostname -s)"

fi

**Executing Block of code using {}**

Ex 1:

pavankumar@pavankumar:~$ cat script.sh

#!/usr/bin/env bash

{ ls;pwd;date; }

pavankumar@pavankumar:~$ ./script.sh

2 Desktop Documents Downloads Firstproject get-docker.sh Music Pictures Public script.sh Templates Videos

/home/pavankumar

Monday 13 September 2021 01:56:02 PM IST

Ex 2:

pavankumar@pavankumar:~$ cat script.sh

#!/usr/bin/env bash

which docker && { echo "docker is intalled on this host"; echo "The docker version is $(docker -v)"; }

pavankumar@pavankumar:~$ ./script.sh

/usr/bin/docker

docker is intalled on this host

The docker version is Docker version 20.10.8, build 3967b7d

Ex 3:

1. If cmd1 is success then execute cmd2, cmd3 .
2. If cmd1 is not success then execute cmd4 command.
3. Don’t display the cmd1 o/p whether it is success or not. Nullifying both success/failure o/p

pavankumar@pavankumar:~$ cat script.sh

#!/usr/bin/env bash

which docker 2>&1 1>/dev/null && { echo "docker installed";echo "The docker version is :$(docker -v)"; } || echo "the docker is not installed "

pavankumar@pavankumar:~$ ./script.sh

docker installed

The docker version is :Docker version 20.10.8, build 3967b7d

**SECTION: 13**

**Simple if Conditional Statements:**

***Syntax:***

* **cmd1 && cmd2**

**if cmd1**

**then**

**cmd2**

**fi**

* **cmd1 && { cmd2 ; cmd3; }**

**if cmd1**

**then**

**cmd2**

**cmd3**

**fi**

**Ex1 :**

**#!/usr/bin/env bash**

**Which docker 2>&1 1>/dev/null && { echo “docker is installed on this host” ; echo The docker version is :$(docker -v) ; }**

**(or)**

**#!/usr/bin/env bash**

**if which docker 2>/dev/null 1>/dev/null**

**then**

**echo "Docker is installed on this host"**

**echo "The Docker version is : $(docker -v)"**

**fi**

**(or)**

**#!/usr/bin/env bash**

**which docker 2>&1 1>/dev/null**

**if [[$? -eq 0]] $? Is the exit status of the command**

**then**

**echo "Docker is installed on this host"**

**echo "The Docker version is : $(docker -v)"**

**fi**

**Ex :**

**if true**

**then**

**echo “Always this will execute”**

**fi**

**if false**

**then**

**echo”It won’t execute”**

**fi**

**if-else Conditional Statements:**

***Syntax:***

* **cmd1 && cmd2 || cmd3**

**if cmd1**

**then**

**cmd2**

**else**

**cmd3**

**fi**

* **cmd1 && { cmd2 ; cmd3; } ||cmd4**

**if cmd1**

**then**

**cmd2**

**cmd3**

**else**

**cmd4**

**fi**

**Simple shell Scripts:**

|  |
| --- |
| **Note: for root user id will always “0”** |

**whoami/id/id -u/id -un (we can use anyone of them)**

**EX:1 To verify the shell script running user is root or not**

root@pavankumar:~# cat root\_or\_not.sh

#!/usr/bin/env bash (Normal way)

id -u 1>/dev/null 2>/dev/null && && echo "you are root"||echo "you are not root"

#!/usr/bin/env bash (with variables using command chaining operators)

userId=$(id -u)

[[ $userId -eq 0 ]] && echo "you are root"||echo "you are not root"

[or]

#!/usr/bin/env bash (without variables using command chaining operators)

[[ $(id -u) -eq 0 ]] && echo "you are root"||echo "you are not root"

[or]

#!/usr/bin/env bash

if [[ $(id -u) -eq 0 ]] (using if-else conditional statements)

then

echo "you are root"

else

echo "you are not root"

fi

root@pavankumar:~# ./root\_or\_not.sh

you are root

**EX:2 User is having sudo priviliges or not.**

pavankumar@pavankumar:~/pavan$ cat isUserhavingSudoPriv.sh

#!/usr/bin/env bash

sudo -v 1>/dev/null 2>/dev/null && echo "The User $(id -un) is having sudo privilages on this host $(hostname)" || echo "The user $(id -un) is not having sudo privilages on this host $(hostname)"

Explanation :-

sudo -v ----->will check the user have sudo privileges or not .

echo $? --🡪with this exit status command we can identify that user have privileges or not. If it’s “0” user have sudo privileges .if it’s have “1” user don’t have sudo privileges.

#!/usr/bin/env bash (Normal way)

sudo -v 1>/dev/null 2>/dev/null && echo "The User $(id -un) is having sudo privilages on this host $(hostname)" || echo "The user $(id -un) is not having sudo privilages on this host $(hostname)"

[or]

#!/usr/bin/env bash (with variables using command chaining operators)

A=$(sudo -v)

[[ $A -eq 0 ]] && echo "The User $(id -un) is having sudo privilages on this host $(hostname)" || echo "The user $(id -un) is not having sudo privilages on this host $(hostname)"

[or]

#!/usr/bin/env bash (without variables using command chaining operators)

[[ $(sudo -v) -eq 0 ]] && echo "The User $(id -un) is having sudo privilages on this host $(hostname)" || echo "The user $(id -un) is not having sudo privilages on this host $(hostname)"

[or]

#!/usr/bin/env bash (using if-else conditional statements)

if [[ $(sudo -v) -eq 0 ]]

then

echo "The User $(id -un) is having sudo privilages on this host $(hostname)"

else

echo "The user $(id -un) is not having sudo privilages on this host $(hostname)"

fi

**Shell script to start docker service**

**systemctl start docker**

**echo $?**

**systemctl stop docker**

**echo $?**

**(Q1) you can run systemctl start docker command from the root user using simple shell script.so first log into root user.**

**#!/usr/bin/env bash (using simple if-else condition)**

**if systemctl status docker 1>/dev/null 2>/dev/null**

**then**

**echo "Already docker is up and running"**

**else**

**echo "starting docker service"**

**systemctl start docker**

**echo "successfully started docker service"**

**fi**

**(Q2) you can run systemctl start docker command if you are the root user otherwise show error message.**

**if [[ $(id -u) -eq 0 ]]**

**then**

**if systemctl status docker 1>/dev/null 2>/dev/null**

**then**

**echo "Already docker is up and running"**

**else**

**echo "starting docker service"**

**systemctl start docker**

**echo "successfully started docker service"**

**fi**

**else**

**echo “sorry you are not allowed to start the docker service.Because you are not the root user”**

**fi**

**(Q3) if you are root user then execute systemctl start docker command .**

**Even though you are not user suppose you may have sudo privileges then you are also allowed to execute systemctl start docker command**

**if [[ $(id -u) -eq 0 ]]**

**then**

**if systemctl status docker 1>/dev/null 2>/dev/null**

**then**

**echo "Already docker is up and running"**

**else**

**echo "starting docker service"**

**systemctl start docker**

**echo "successfully started docker service"**

**fi**

**else**

**if sudo -v 1>/dev/null 2>/dev/null**

**then**

**if systemctl status docker 1>/dev/null 2>/dev/null**

**then**

**echo "Already docker is up and running"**

**else**

**echo "starting docker service"**

**systemctl start docker**

**echo "successfully started docker service"**

**fi**

**else**

**echo “sorry you are not allowed to start the docker service.Because you are not the root user and also you don’t have the sudo privileges to start it”**

**fi**

**fi**

**Logical operators:**

* Logical AND Operator && or -a
  + will compare two inputs & if both are true, it will return true else false.
* Logical OR Operator || or -o
  + will check two conditions will return true, if any of them is true & return false when both are false.
* Logical NOT Operator !
  + will return true when the condition is false & return false if the condition is true

**Note:**

**-a, -o are the old versions if you use those you need to give only single brackets .Don’t give double brackets**

**LOGICAL NOT OPERATOR**

**Since we are using not operator it will give opposite answer.**

**pavankumar@pavankumar:~/pavan$ cat logicaloperators.sh**

**#!/usr/bin/env bash**

**read -p "Enter any number: " num**

**if [[ ! $num -ge 50 ]]**

**then**

**echo "$num is greater than or equal to 50"**

**else**

**echo "$num is lesser than 50"**

**fi**

**Note: -p is used to display prompt message in read command**

**[or]**

**we can use single or double square brackets for Logical NOR operator**

**pavankumar@pavankumar:~/pavan$ cat logicaloperators.sh**

**#!/usr/bin/env bash**

**read -p "Enter any number: " num**

**if [ ! $num -ge 50 ]**

**then**

**echo "$num is greater than or equal to 50"**

**else**

**echo "$num is lesser than 50"**

**fi**

**(Q1) To find out the value is greater than or equal to the given number**

**pavankumar@pavankumar:~/pavan$ cat logicaloperators.sh**

**#!/usr/bin/env bash**

**read -p "Enter any number: " num**

**if [[ $num -ge 50 ]]**

**then**

**echo "$num is greater than or equal to 50"**

**else**

**echo "$num is lesser than 50"**

**fi**

**LOGICAL AND OPERATOR**

**(Q1) To find out the value is in between the range to the given number**

**pavankumar@pavankumar:~/pavan$ cat logicaloperators.sh**

**#!/usr/bin/env bash**

**read -p "Enter any number: " num**

**if [[ $num -ge 50 ]] && [[ $num -le 100 ]]**

**then**

**echo "$num is in the range between 50-100"**

**else**

**echo "$num is not in the range between 50-100"**

**fi**

**[or]**

**Note: whenever we are using logical operators between two conditions. we can club them in single test condition.**

**pavankumar@pavankumar:~/pavan$ cat logicaloperators.sh**

**#!/usr/bin/env bash**

**read -p "Enter any number: " num**

**if [[ $num -ge 50 && $num -le 100 ]]**

**then**

**echo "$num is in the range between 50-100"**

**else**

**echo "$num is not in the range between 50-100"**

**fi**

**[or]**

**using old version logical operator (not recomended)**

**pavankumar@pavankumar:~/pavan$ cat logicaloperators.sh**

**#!/usr/bin/env bash**

**read -p "Enter any number: " num**

**if [ $num -ge 50 -a $num -le 100 ]**

**then**

**echo "$num is in the range between 50-100"**

**else**

**echo "$num is not in the range between 50-100"**

**fi**

**LOGICAL OR OPERATOR**

**Advanced version**

**#!/usr/bin/env bash**

**read -p "Enter your confirmation to start docker : (say yes or no)" num**

**if [[ $num == yes ]] || [ $num == y ]]**

**then**

**echo "starting docker ..."**

**sudo systemctl start docker**

**else**

**echo "skipping...."**

**fi**

**[or]**

**#!/usr/bin/env bash**

**read -p "Enter your confirmation to start docker : (say yes or no)" num**

**if [[ $num == yes|| $num == y ]]**

**then**

**echo "starting docker ..."**

**sudo systemctl start docker**

**else**

**echo "skipping...."**

**fi**

**[or]**

**#!/usr/bin/env bash**

**read -p "Enter your confirmation to start docker : (say yes or no)" num**

**if [[ $num =~ yes|y ]]**

**then**

**echo "starting docker ..."**

**sudo systemctl start docker**

**else**

**echo "skipping...."**

**fi**

**[or]**

**using old version logical operator (not recomended)**

**#!/usr/bin/env bash**

**read -p "Enter your confirmation to start docker : (say yes or no)" num**

**if [ $num == yes -o $num == y ]**

**then**

**echo "starting docker ..."**

**sudo systemctl start docker**

**else**

**echo "skipping...."**

**fi**

**Difference between [ and [[**

**1.We can avoid exceptions**

1. **if you use single sq brackets then you must use -o for OR , -a for AND otherwise it will give exceptions.**

**If you use double sq brackets then you can use any -o, -a, &&, ||.**

1. **For single sq brackets Suppose you execute without giving input then it will give exceptions.**

**For double sq brackets if you execute without giving input also then it won’t give any exceptions**

**#!/usr/bin/env bash**

**read -p "Enter your confirmation to start docker : (say yes or no)" num**

**if [ $num == yes -o $num == y ]**

**then**

**echo "starting docker ..."**

**sudo systemctl start docker**

**else**

**echo "skipping...."**

**fi**

**2.With doubt square brackets we can use regular expressions**

**if [[ $num == yes || $num == y ]] ===> if [[ $num ==~ yes|y ]]**

**3.With doubt square brackets we can use > or < symbols**

**You can use < or > using (()) also.**

**If -elif-elif-elif-else conditional statements**

**pavankumar@pavankumar:~/pavan$ cat actionsonDocker.sh**

**#!/usr/bin/env bash**

**###################################################**

**# #**

**# Author: P Pavankumar #**

**# Date : 09/19/2021 #**

**# Actions: start, stop. Restart and version #**

**# #**

**###################################################**

**(Q1) if you are not the root user exit , if you are the root user then give the i/p for start, stop, restart, version.**

**If [[ $(id -u) -ne 0 ]]**

**then**

**echo “you are not allowed to run this script.Because you are not the root user”**

**exit 1**

**fi**

**read -p "Enter your option (start, stop, restart, version): " num**

**if [[ $num == start ]]**

**then**

**echo "starting docker..."**

**systemctl start docker**

**elif [[ $num == stop ]]**

**then**

**echo "stopping docker"**

**systemctl stop docker**

**elif [[ $num == restart ]]**

**then**

**echo "Restarting docker"**

**systemctl restart docker**

**elif [[ $num == version ]]**

**then**

**echo "The version of docker is : $(docker -v)"**

**else**

**echo "your option is invalid"**

**echo "valid options are start, stop,restart and version "**

**fi**

**(Q2) if you are not the root user exit , if you are the root user then give the i/p for start, stop, restart, version. Use case statements**

How to handle command line arguments?

**pavankumar@pavankumar:~/pavan$ cat actionsonService.sh**

**#!/usr/bin/env bash**

**if [[ $# -ne 2 ]]**

**then**

**echo "Hey admin please run this script as follows"**

**echo "usage: $0 <ServiceName> <ActiontoExecuteOnService>"**

**echo "valid ActiontoExecuteOnService are: start stop restart status"**

**exit 1**

**fi**

**ServiceName=$1**

**Action=$2**

**sudo systemctl $Action $ServiceName**

**Note:**

**$# ===> represents number of arguments.**

**exit 1 ===>you can give any non-zero value**

**Be clear while creating command line arguments as a developer we need to give some additional info for script .**

**SECTION: 14**

**Scheduling jobs with at command**

at command is very useful for scheduling one time tasks.

Example:

Shutdown system at the specified time

Taking a one-time backup.

**Syntax:**

echo “bash backup.sh" | at 9:00 AM

Or

Run first: at 9:00 AM then enter and give the cmd or script to run and press ctrl+D.

**Commands used with at:**

at : execute commands at specified time.

atq : lists the pending jobs of users.

atrm : delete jobs by their job number

|  |
| --- |
| **at time AM**  **bash scriptname.sh then click on enter**  **ctrl+d =====> to save job** |

♣ Ex-1: Schedule task at coming 10:00 AM.

♣ at 10:00 AM

♣ Ex-2: Schedule task at 10:00 AM on coming Sunday.

♣ at 10:00 AM Sun

♣ Ex-3: Schedule task at 10:00 AM on coming 25’th July.

♣ at 10:00 AM July 25

♣ Ex-4: Schedule task at 10:00 AM on coming 22’nd June 2025.

♣ at 10:00 AM 6/22/2015

♣ at 10:00 AM 6.22.2015

♣ Ex-5: Schedule task at 10:00 AM on the same date at next month.

♣ at 10:00 AM next month

♣ Ex-6: Schedule task at 10:00 AM tomorrow.

♣ at 10:00 AM tomorrow

♣Ex-7: Schedule task to execute just after 1 hour.

♣ at now + 1 hour

♣ Ex-8: Schedule task to execute just after 30 minutes.

♣ at now + 30 minutes

♣ Ex-9: Schedule task to execute just after 1 and 2 weeks.

♣ at now + 1 week ♣ at now + 2 weeks

♣ Ex-10: Schedule task to execute just after 1 and 2 years.

♣ at now + 1 year ♣ at now + 2 years

♣ Ex-11: Schedule task to execute at midnight.

♣ at midnight

**Scheduling jobs with crontab (crontab.guru)**

The crontab is used for running specific tasks on a regular interval.

♣ Each user can schedule jobs using crontab.

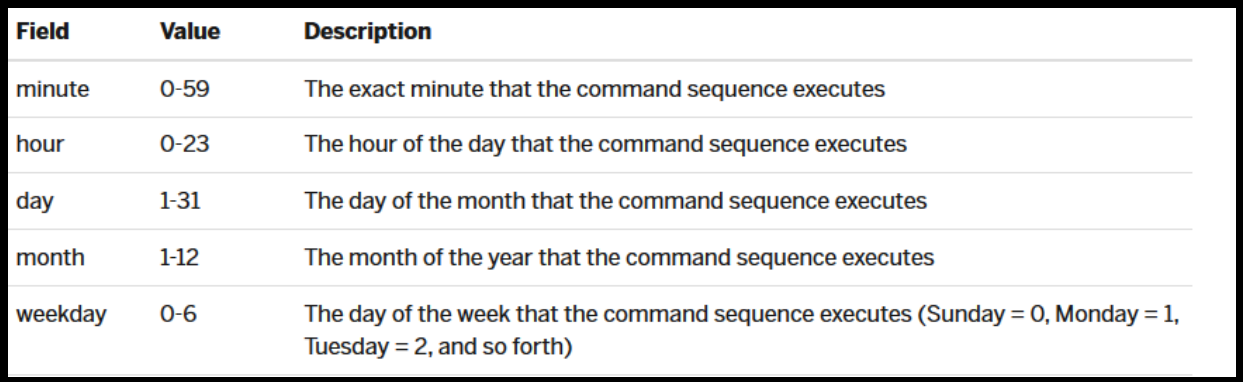
♣ Syntax:

♣ minute(s) hour(s) day(s) month(s) weekday(s) command/script

♣ Each scheduled job has six fields

♣ Don’t change the order and six fields are separated by space

♣ The first five are integer patterns and the sixth is the command/script to execute.

****

**Useful crontab commands**

♣ Use crontab -e to schedule a job.

♣ Use crontab -l to list the jobs (crontab -u user\_name -l )

♣ Use crontab -r to remove jobs

**Scheduling jobs with crontab:**

♣ 30 9 15 11 6 /root/my\_backup.sh

♣ 30 9 15 \* 6 /root/my\_backup.sh

♣ 30 9 15 \* \* /root/my\_backup.sh

♣ 30 9 \* \* \* /root/my\_backup.sh

♣ 30 \* \* \* \* /root/my\_backup.sh

♣ \* \* \* \* /root/my\_backup.sh

♣ Schedule a crontab to execute on every Sunday at 5 PM.

♣ 0 17 \* \* 0 /root/my\_backup.sh

♣ Schedule a crontab to execute on every Sunday at 5 AM and 5 PM

♣ 0 5,17 \* \* 0 /root/my\_backup.sh

♣ Schedule a crontab to execute on every two hours.

♣ 0 \*/2 \* \* \* /root/my\_backup.sh

♣ Yearly once:

♣ 0 0 1 1 \* /root/my\_backup.sh

♣ @yearly /root/my\_backup.sh

♣ @monthly

♣ @weekly

♣ @daily

♣ @hourly

♣ @reboot It useful for those tasks which you want to run on your system startup.

Q. Schedule a cron to execute at 2am daily.

0 2 \* \* \* /bin/sh backup.sh

Q.Schedule a cron to execute on every minutes.

\* \* \* \* \* /scripts/script.sh

Q.Schedule a cron to execute on every 10 minutes.

\*/10 \* \* \* \* /scripts/monitor.sh

Q.Schedule a cron to execute on Jan,march,dec.

\* \* \* 1,3,12 \* /script/script.sh

Q.Schedule a cron to execute on first sunday of every month.

0 2 \* \* sun [[ $(date '+%d') -le 07 ]] && /script/script.sh

Concept: command1 && command2 (command2 will execute if command1 is success)

**Shell script to send Automatic Mail alert when RAM Memory gets low**

Free ==> you can see RAM size in kb

Free -m ==> you can see RAM size in mb

Free -mt ==> you can see Total used memory & currently available memory

Free -g ==> you can see RAM size in gb

(Q) When ever my RAM size is lessthan 400mb.we can say that my RAM is getting low.

And schedule job for every min, hour, day , month , year

**pavankumar@pavankumar:~/pavan$ cat send\_au\_mail.sh**

**#!/bin/bash**

**To="ppasupuleti50@gmail.com"**

**TH\_L= 400**

**free\_RAM= $(free -mt | grep -E "Total:" | awk '{print $4}')**

**if [[ $free\_RAM -lt TH\_L ]]**

**then**

**echo -e "server is running with low RAM size\nAvaliable RAM is : $free\_RAM" | mail -s "RAM INFO $(date)" “$To”**

**fi**

Note: to execute new line in echo command we must use -e command

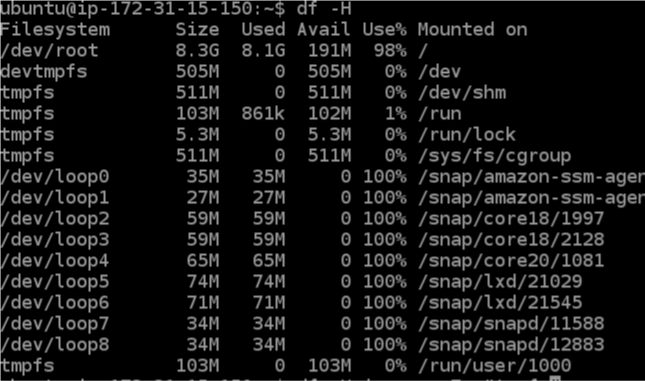
**\* \* \* \* \* /bin/bash /home/ubuntu/ send\_au\_mail.sh**

**Shell script to monitor file system utilization with mail alerts**

Shell script to monitor Disk space usage with email alert

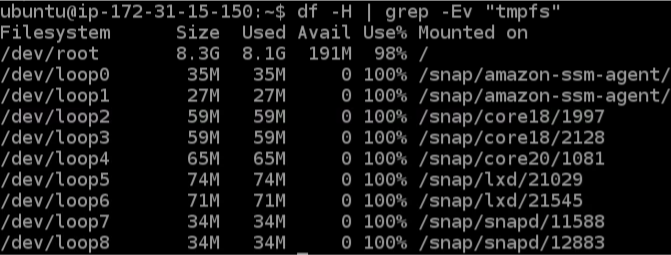
Ex:

df -H ==> you will get info about file system utilization



(Q1) Suppose if you don’t want to get tmpfs files then we need to use regular “v“exp near -E

df -H |grep -Ev “tmpfs”



**#!/usr/bin/env bash**

**mailId=”**[**ppasupuleti50@gmail.com**](mailto:ppasupuleti50@gmail.com)**”**

**echo -e “The file system utilization on $(hostname) is : \n $( df -H |grep -Ev “tmpfs”) ” | /usr/bin/mail -s “file system utilization” “$mailId”**

**\* \* \* \* \* /bin/bash /home/ubuntu/filename.sh**

**(Q2) Suppose if you want to monitor only /dev/root file system then**

**df -H | grep -E /dev/root |awk ‘{print $5}’ | sed ‘s/%//’ ===> substitute percentile with nothing**

**[or]**

**df -H | grep -E /dev/root |awk ‘{print $5}’ | tr -d “%”**

**From the above command you will get only 98 value**

**#!/usr/bin/env bash**

**mailId=”**[**ppasupuleti50@gmail.com**](mailto:ppasupuleti50@gmail.com)**”**

**Fu= df -H | grep -E /dev/root |awk ‘{print $5}’ | tr -d “%”**

**If [[ $Fu -ge 80 ]]**

**echo “The file system /dev/root usage is : $Fu”| /usr/bin/mail -s “file system utilization alert” “$mailId”**

**we need to schedule this job at every 10 min then**

**\*/10 \* \* \* \* /home/ubuntu/filename.sh**

**SECTION: 15**

**Arrays for Bash Shell Scripting:**

Concepts of Arrays:

1. What is an Array ?
2. How to define array ?
3. How to access Array Values ?
4. Different Types of Arrays
5. How to store the command output into an array ?
6. How to delete and update exiting array with new values ?
7. How to read array using read command ?

**What is an array ?**

An Array is the data structure of the bash shell, which is used to store multiple data’ s.

Simple array:

myarray=( ls pwd date 2 5.6 ) #No limit for length of an array

**How to Define/declare an array ?**

There are different ways to define an array in bash shell scripting.

myArray=() =====> Empty Array

mycmds=( ls pwd date 2 5.6 )

myNewArray=( ls -lrt hostname -s )

myNewArray=( “ls –lrt” “hostname –s" )

declare -a NewArray

NewArray=( 1 3 4 5 bash scripting)

**How to access Array values/elements ?**

Basically, Bash Shell Array is the zero-based Array (i.e., indexing start with 0)

Then what is an index ?

myarray=(23 4 6 15 5 7 )

myarray

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 23 | 4 | 6 | 15 | 5 | 7 |

0 1 2 3 4 5 ===> Positive Index Values or Positive Indices

-6 -5 -4 -3 -2 -1 ===> Negative Index Values or Negative Indices

echo “$myarray”

echo “${myarray}”

echo “${myarray[\*]}” ===> it prints all the values

echo “${myarray[@]” ====> it also prints all the values

echo “${myarray[0]” ====> Prints first value

echo “${myarray[-1]}” ===> Prints last value

echo “${myarray[\*]:0}” ===> Prints all the values starting from index-0

echo “${myarray[\*]:1}” ===> Prints all the values starting from index-1

echo “${myarray[\*]:0:2}” ===>Prints two values starting from index-o

echo “${myarray[\*]:1:2}” ===>Prints two values starting from index-1

echo "${!myarray[\*]}" ===> Prints index values of array

echo “${#myarray[\*]}” ===>Find the length (number elements) of array

We can also customize index numbers:

newarray[5]=“bash”

newarray[9]=“shell scriting”

[Or]

newarray=([5]=“bash” [9]=“shell scripting”)

Note: We can also take indices as strings and that array is called Associative Array.

**Different Types of Arrays ?**

We have two types of arrays in Bash Shell Scripting.

They are:

* 1. Index Based Arrays or Arrays

2. Associative Arrays

**How to store the command output into an array ?**

Storing the output of a command into array:

arraywithcmd=( $(command) )

**How to delete and update an exiting array ?**

Delete an array or even normal variable:

unset variable/arrayvariable

Updating an exiting array:

myarray=(1,2,3)

myarray+=(4,5,6)

**How to read an array using read command ?**

Syntax:

read -a myarray

read –p “Enter your array” -a myarray

**Associative Arrays:**

* We already know about Normal arrays or Indexed Arrays or Index Based Arrays and for these arrays Index values or Indices are Numbers (Integer Number)
* Associative arrays are the arrays with index values as strings.
* Generally no need to declare normal arrays before using them but we have to declare Associative arrays before using them.
  + declare -A myassarray
* Defining Associative Arrays:
  + - myassarray=([name]=“bash shell scripting” [version]=4.4)
    - Or
    - myassarray[name]=“bash shell scripting”
    - myassarray[version]=4.4
* Simply Associative Arrays are called key-value pair representation.

**Differences between Indexed Arrays and Associative Arrays**

* Associative Arrays should be declare before using them (declare –A myassarray).
* echo “${indexarray}” 🡪 will print first value by default===>In index array
* echo “${myassarray}” 🡪 wont print any value ==>Associate array
* Indexed Array Indices are numbers
* Associative Array Indices are strings

**SECTION: 16 (PART-1)**

**Introduction to loops:**

Most languages have the concept of loops and they are very useful to execute series of commands for n number of times.

♣ Types of loops:

* for loop
* while loop
* until loop
* select loop

*for loop syntaxes:*

Different ways to use for loop

Basic For loop:

for variables in list\_of\_values

do

command1

command2

done

C-language type for loop:

for ((initialization;condition;increment/decrement))

do

command1

command2

done

infinity for loop: it will print infinity number of times. To stop this press ctrl+c

for ((;;))

do

command1

command2

done

basic\_for\_loop.sh

-----------------

#!/bin/bash

#for each\_value in 1 2 3

for each\_file in $(ls)

do

echo "this is a loop"

echo "for this iteration each\_value is: $each\_file"

done

=======================================================

c\_type\_for\_loop.sh

------------------

#!/bin/bash

clear

echo "we are working with loops"

for((cnt=1;cnt<=20;cnt++))

do

echo "$cnt"

done

echo "we complete for loop"

==================================================

infi\_for\_loop.sh

----------------

#!/bin/bash

echo "This is a for loop"

cnt=1

for((;;))

do

echo "this is infi loop"

((cnt++))

sleep 1

if [[ $cnt -eq 10 ]]

then

break

fi

done

echo "out of loop"

====================================================

Q1 . Check the given file having execution permission or not using command chaining operator.

#!/usr/bin/env bash

[[ -x loopEx1.sh ]] && echo "your loopEx1.sh file have execution permission" || echo "your loopEx1.sh file doesn't have execution permission"

Q2 . Check the given two files having execution permission or not using if-else condition

#!/usr/bin/env bash

if [[ -x loopEx1.sh ]]

then

echo "your loopEx1.sh file have execution permission"

else

echo "your loopEx1.sh file doesn't have execution permission"

fi

if [[ -x loopEx2.sh ]]

then

echo "your loopEx2.sh file have execution permission"

else

echo "your loopEx2.sh file doesn't have execution permission"

fi

Q3 . Check the given two files having execution permission or not using for loop.

#!/usr/bin/env bash

for A in loopEx1.sh loopEx2.sh

do

if [[ -x $A ]]

then

echo "your "$A" file have execution permission"

else

echo "your "$A" file doesn't have execution permission"

fi

done

Q4 . without placing filenames manually give command to find all files having execution permission or not .

#!/usr/bin/env bash

for A in $(ls)

do

if [[ -x $A ]]

then

echo "your "$A" file have execution permission"

else

echo "your "$A" file doesn't have execution permission"

fi

done

Note: it works only in the current folder .

Q5 . without placing filenames manually give the input as path to find the files having execution permission not , using command line argument.

#!/usr/bin/env bash

if [[ $# -ne 1 ]]

then

echo "usage of this script : $0 <ProvideAnyPath>"

exit 1

fi

given\_path=$1

for A in $(ls $given\_path)

do

if [[ -x $A ]]

then

echo "your "$A" file have execution permission"

else

echo "your "$A" file doesn't have execution permission"

fi

done

Note: please pass arguments while working with commandline arguments and also provide some message then others will understand.

Q6. Print the given text 4 times using for loop.

#!/usr/bin/env bash

for A in 1 2 3 4

do

echo " Hi pavan welcome to Bash shell scripting"

echo “This is the $A iteration.”

done

Note: variable is optional we can use based on requirement.

(Q) Script about installing packages (vim,apache2 ,nginx)

Run only from the root user otherwise ask to run from root user/use sudo privileges.

Install vim package .if it is already installed then show success message and if the package is not available show the message package is not available.

Step 1: creating if block, if the user is not root then asking the normal user to run from the root /provide sudo privileges to run the script and stop the operation

Step 2: if vim is installed then say installed message otherwise ask to install the package. If the package is installed they show installed message other say unable to install package.

Step 3: creating for loop for multiple packages to install . in this we are manually providing packages (means its fix).so we are going for command line orguments.

Step 4: creating command line arguments there we can pass our arguments based on our wish.

Must provide some usage message then user will understand.

If we are not providing any arguments then show message and stop operation.

!/usr/bin/env bash

#Author :p pavankumar

#usage : installing packages using command line arguments

if [[ $(id -u) -ne 0 ]]

then

echo "please switch into root user otherwise make sure that you have sudo privilages"

exit 1

else

if [[ $# -eq 0 ]]

then

echo "usage : $0 <pkg1> <pkg2> ....."

exit 2

else

for A in $@

do

which $A &>/dev/null

if [[ $? -eq 0 ]]

then

echo "Already $A installed in your machine"

else

echo "installing $A in your machine just a min"

apt-get install $A -y &>/dev/null

if [[ $? -eq 0 ]]

then

echo "$A installed successfully"

else

echo "unable to install $A package “

fi

fi

done

fi

fi

#!/bin/bash

#Author: Narendra

#Installing mutliple pkags

if [[ $# -eq 0 ]]

then

echo "Usage: $0 pkg1 pkg2 ...."

exit 1

fi

if [[ $(id -u) -ne 0 ]]

then

echo "Please run from root user or with sudo privilage"

exit 2

fi

for each\_pkg in $@

do

if which $each\_pkg &> /dev/null

then

echo "Already $each\_pkg is installed"

else

echo "Installing $each\_pkg ......"

yum install $each\_pkg -y &> /dev/null

if [[ $? -eq 0 ]]

then

echo "Successfully installed $each\_pkg pkg"

else

echo "Unable to install vim $each\_pkg"

fi

fi

done

**Difference\_between\_$@\_and$\*.sh**

!/bin/bash

echo "The below output is for \$\*"

for each in "$\*"

do

echo $each

done

echo "The below output is for \$@"

for each in "$@"

do

echo $each

done

Note: in for loop we are using command line arguments as “$\*” and “$#”. Why we are giving double quotations means whenever we are giving string as command line argument then it will take as string.if you don’t give double quotes to $\* and $# although you giving i/p as string argument it treats single argument.

***Loop Control Commands :***

Break & Continue commands are used to control the execution of loops.

**Break**: Break command is used to terminate/exit current loop completely before the actual ending of loop.

**Continue**:

Continue command is used in script to skip current iteration of loop & continue to next iteration of the loop.

Q1. Display all the files using for loop.

#!/bin/bash

#ls

<< mcom

for each\_file in $(ls)

do

echo "$each\_file"

done

mcom

Q2. Display only the text files using for loop (count)

cnt=1

for each\_file in $(ls \*.txt)

do

if [[ $cnt -eq 1 ]]

then

echo "$each\_file"

((cnt++))

fi

done

Q3. Display only the text files using for loop (break)

echo "starting for loop"

cnt=1

for each\_file in $(ls \*.txt)

do

if [[ $cnt -eq 1 ]]

then

echo "$each\_file"

break

fi

done

echo "for loop is over"

Q3. Giving i/p as 1 to 10 then

Display numbers from only 1 to 5 using for loop.

#!/bin/bash

echo "starting for loop"

for each in $(seq 1 10)

do

if [[ $each -gt 5 ]]

then

break

fi

echo "$each"

done

Q3. Giving i/p as 1 to 10 then

Display numbers from only 1 to 5 using for loop.

for each in $(seq 1 10)

do

if [[ $each -ne 5 ]]

then

echo "$each"

fi

done

Q4. Display number from 1 to 10 by excluding 5

for each in $(seq 1 10)

do

if [[ $each -eq 5 ]]

then

continue

fi

echo "$each"

done

echo "for loop is over"

***For loop with arrays :***

Normal for loop

for eachValue in docker nginx

do echo “The service is: $eachValue

done

For loop with arrays

myServices=(docker nginx)

for eachValue in ${myServices[@]}

do echo “The service is: $eachValue

done

**SECTION: 17**

**Login into remote server from local server using ssh**

**Using Passwrod:**

ssh user\_name@remote\_ip

[or]

ssh user\_name@servername

ssh remote\_ip (here remote user name is same as local terminal user)

Note:-

vi /etc/ssh/sshd\_config (Make it; PasswordAuthentication yes in remote server)

========================================================================

***Using passwordless: By sharing exchange keys***

Step1: On local server generate keys using ssh-keygen

Step2: go to user\_home/.ssh then here you will find two files

id\_rsa (private key, it should be safe)

id\_rsa.pub (public, this has to share with remote servers)

Step3: use below command to share public key with remtoe server, it will ask password

ssh-copy-id username@remote\_server\_ip

Step4: if step3 is success then use below command to login with remote server, it wont ask password now

ssh username@remote\_server\_ip

**Executing commands on remote server without logging into remote server:**

Two ways (for both password and password less Authentication)

♣ First way:

♣ ssh user\_name@remote\_server

♣ Provide the password if it is password authentication connection.

♣ Now run the command and see the result

♣ Run exit command to close remote session

♣ Note: This is not good for automation

♣ Second way:

♣ ssh user\_name@remote\_server “command”

♣ Provide the password if it is password authentication connection.

♣ This is good for automation, if the connection is password less authentication

**Note:**

ssh -t user\_name@remote\_server “command”

ssh -t -o StrictHostKeyChecking=No user\_name@remote\_server “command”

-o StrictHostKeyChecking=No ====> If you give this it won’t ask everytime for fingerprint like (yes/no)

-t ====> for some commands like top, …etc o/p will showing without stopping console it will be running . to stop the console we need to press ctrl+c. In the same way when we try to execute this type of commands we need to give -t otherwise it throws error .

**Executing command on remote server without logging into remote server:**

♣ ssh -t -o StrictHostKeyChecking=No user\_name@remote\_server “command”

**Executing multiple commands on remote server without logging into remote server**

♣ ssh -t -o StrictHostKeyChecking=No user\_name@remote\_server “cmd1;cmd2;cmd3”

Note:

Command will run in remote server but output will shows in local server.

**Providing password for ssh using sshpass utility:**

Basic ssh command to run commands on remote server:

ssh -t -o StrictHostKeyChecking=No automation@54.91.148.241 "date"

**Using sshpass (providing password for ssh):**

sshpass -p "automation@123" ssh -t -o StrictHostKeyChecking=No automation@54.91.148.241 "date"

[or]

sshpass -f path\_for\_password\_file ssh -t -o StrictHostKeyChecking=No automation@54.91.148.241 "date"

[or]

export SSHPASS="automation@123"

sshpass -e ssh -t -o StrictHostKeyChecking=No automation@54.91.148.241 "date"

*Explanation:*

1.providing password directly in command. (not the good practise)

2.saving password in one file and giving that path of the file.

3.exporting password in SSHPASS .it’s a variable derived by the sshpass command.

=====================================================================================

Amazon Linux EPEL:

cd /opt

wget -r --no-parent -A 'epel-release-\*.rpm' http://dl.fedoraproject.org/pub/epel/7/x86\_64/Packages/e/

yum install dl.fedoraproject.org/pub/epel/7/x86\_64/Packages/e/epel-release-7-12.noarch.rpm

RHEL8 EPEL:

yum install https://dl.fedoraproject.org/pub/epel/epel-release-latest-8.noarch.rpm

=====================================================================================

**Executing multiple commands on multiple servers**

#!/bin/bash

Method 1 : (using normal way running commands in one server)

echo "The date command output on the server: 100.26.187.33"

sshpass -f pass ssh -o StrictHostKeyChecking=No automation@100.26.187.33 "date"

echo "-------------------------------------------------------"

echo "The uptime command output on the server: 100.26.187.33"

sshpass -f pass ssh -o StrictHostKeyChecking=No automation@100.26.187.33 "uptime"

echo "-------------------------------------------------------"

echo "The free -m command output on the server: 100.26.187.33"

sshpass -f pass ssh -o StrictHostKeyChecking=No automation@100.26.187.33 "free -m"

echo "-------------------------------------------------------"

Method 2 : (using for loop running commands in one server)

for each\_cmd in date uptime "free -m"

do

echo "The $each\_cmd command output on the server: 100.26.187.33"

sshpass -f pass ssh -o StrictHostKeyChecking=No [automation@100.26.187.33](mailto:automation@100.26.187.33) "$each\_cmd"

echo "-------------------------------------------------------"

done

Note:

ssh -o StrictHostKeyChecking=No automation@100.26.187.33 "$each\_cmd" ====>if you had exchanged password then use the above command

Method 3 : (using for loop running commands on multiple servers)

for each\_ser in 100.26.187.33 100.26.187.33

do

echo "Executing cmds on $each\_ser"

echo "============================"

for each\_cmd in date uptime "free -m"

do

echo "The $each\_cmd command output on the server: $each\_ser"

#sshpass -f pass ssh -o StrictHostKeyChecking=No automation@$each\_ser "$each\_cmd"

ssh -o StrictHostKeyChecking=No automation@$each\_ser "$each\_cmd"

echo "-------------------------------------------------------"

done

Method 4 : (using for loop running commands on multiple servers And server ip’s are stored in text file)

for each\_ser in $(cat remote\_servers.txt)

do

echo "Executing cmds on $each\_ser"

echo "============================"

for each\_cmd in date uptime "free -m"

do

echo "The $each\_cmd command output on the server: $each\_ser"

#sshpass -f pass ssh -o StrictHostKeyChecking=No automation@$each\_ser "$each\_cmd"

ssh -o StrictHostKeyChecking=No automation@$each\_ser "$each\_cmd"

echo "-------------------------------------------------------"

done

done

**Shell script to execute different commands on different servers with different users and different passwords**

servers\_info.txt

----------------

18.212.27.210 automation automation@123 date

18.212.185.2 tomcat tomcat123 whoami

1.we have four commands in each line and these are stored in a text file.

2.if you try to read these four commands in each line then it will print as 8 lines as o/p. it won’t print in two lines as it is whatever in text file.

3.At that time we are using while loop.

Ex:

#!/bin/bash

for A in $(cat servers\_info.txt)

do

echo " $A"

done

OUTPUT: IT IS NOT THE EXCEPTED OUTPUT

18.212.27.210

Automation

automation@123

date

18.212.185.2

tomcat

tomcat123

whoami

Reason:

**this is because for loop will read values by spaces**

Ex:

#!/bin/bash

while read A

do

echo "$A"

done < servers\_info.txt

OUTPUT: IT IS THE EXCEPTED OUTPUT

18.212.27.210 automation automation@123 date

18.212.185.2 tomcat tomcat123 whoami

[OR]

Ex:

#!/bin/bash

while read A B C

do

echo "$A $B $C"

done < servers\_info.txt

OUTPUT: IT IS THE EXCEPTED OUTPUT

18.212.27.210 automation automation@123

18.212.185.2 tomcat tomcat123

Reason:

**this is because, while loop will read values line by line**

#!/bin/bash

while read ser user pass cmd1 cmd2

do

echo "Executing $cmd1 on $ser with user as $user and password $pass"

sshpass -p $pass ssh -n -o StrictHostKeyChecking=No $user@$ser "$cmd1"

echo "Executing $cmd2 on $ser with user as $user and password $pass"

sshpass -p $pass ssh -n -o StrictHostKeyChecking=No $user@$ser "$cmd2"

echo "---------------------------------------------------------"

done < servers\_info.txt

Note:

-n =======> don’t except any i/p from std in (if you don’t give this then it will print only for first user o/p it won’t print second user o/p)

**SECTION: 18 (PART-2)**

**Loops and loop control statements:**

**Infinity while loop:**

while true while :

do do

statements/commands statements/commands

done done

**while loop with commands:**

while command while [[ 3 -gt 5]]

do do

statements/commands statements/commands

done done

**Reading a file content:**

while read line

do

statements/commands

done

**Reading command output:**

command |while read line

do

statements/commands

done

Ex 1: **Infinity while loop (**prints infinity times ok )

#!/bin/bash

while :

do

echo “ok”

done

Ex 2: **Infinity while loop (**prints infinity times ok )

#!/bin/bash

while true

do

echo “ok”

done

Ex 3: **while loop with commands** (date command o/p) & ok

#!/bin/bash

while date

do

echo “ok”

done

Ex 4:

#!/bin/bash **while loop with commands (**prints infinity times ok )

while date &> /dev/null

do

echo “ok”

done

Ex 5: **while loop with test commands (**prints 1 to 10 numbers )

#!/bin/bash

A=1

while [[ $A -le 10 ]]

do

echo “$A”

((A++))

done

Ex 6: **Reading a file content:**

#!/bin/bash

while read A

do

echo “$A”

done < servers\_info.txt

Ex 7: **Reading command output:**

#!/bin/bash

Cat servers\_info.txt |while read A

do

echo “$A”

done

**While loop with IFS**

servers\_info.txt

----------------

18.212.27.210 ,automation,automation@123,date

18.212.185.2,tomcat,tomcat123,whoami

***NOTE:***

By default while loop will splits the lines based on space

To devide the lines into fields ,we use IFS=”field\_separator”

EX 1:

#!/bin/bash

while IFS=’,’ read f1 f2 f3 f4

do

echo “$f2”

done < servers\_info.txt

OUTPUT:

automation

tomcat

EX 1:

#!/bin/bash

while IFS=’,’ read f1 f2

do

echo “$f2”

done < servers\_info.txt

OUTPUT

automation,automation@123,date

tomcat,tomcat123,whoami

Note: **VVVVVIMP**

suppose if you are going to give exactly the no.of variables which are there in your file then first variable will assign to first field and second variable will assign to second field ………….etc.

But if you are not passing exact no.of variables then only the first variable will assign to the first field and for the last variable the remaining all fields will assign to that

**SECTION: 19**

**Simple Introduction to functions:**

A Function is a block of code that performs a specific task and which is reusable.

▪ Functions concept reduces the code length.

▪ Two ways to define a function

▪ Calling a function is exactly same as calling any command

**METHOD 1:**

function function\_name

{

commands/statements

}

**METHOD 2:**

function\_name ()

{

commands/statements

}

EX 1: Without using functions concept

#!/bin/bash

clear

echo "--------------------------------"

echo "Welcome to Arithmetic Calculator"

echo "--------------------------------"

echo -e "[a]ddition\n[b]Subtraction\n[c]Multiplication\n[d]Division\n"

read -p "Enter your choice: " choice

case $choice in

[aA])

read -p "Enter first number: " num1

read -p "Enter second number: " num2

result=$((num1+num2))

echo "The result for your choice is: $result"

;;

[bB])

read -p "Enter first number: " num1

read -p "Enter second number: " num2

result=$((num1-num2))

echo "The result for your choice is: $result"

;;

[cC])

read -p "Enter first number: " num1

read -p "Enter second number: " num2

result=$((num1\*num2))

echo "The result for your choice is: $result"

;;

[dD])

read -p "Enter first number: " num1

read -p "Enter second number: " num2

result=$((num1/num2))

echo "The result for your choice is: $result"

;;

\*)

echo "Wrong choice"

;;

esac

EX 2: With using functions concept

#!/bin/bash

mycode()

{

read -p "Enter first number: " num1

read -p "Enter second number: " num2

}

clear

echo "--------------------------------"

echo "Welcome to Arithmetic Calculator"

echo "--------------------------------"

echo -e "[a]ddition\n[b]Subtraction\n[c]Multiplication\n[d]Division\n"

read -p "Enter your choice: " choice

case $choice in

[aA])

mycode

result=$((num1+num2))

echo "The result for your choice is: $result"

;;

[bB])

mycode

result=$((num1-num2))

echo "The result for your choice is: $result"

;;

[cC])

mycode

result=$((num1\*num2))

echo "The result for your choice is: $result"

;;

[dD])

mycode

result=$((num1/num2))

echo "The result for your choice is: $result"

;;

\*)

echo "Wrong choice"

;;

esac

**Defining a function and calling a function**

#!/bin/bash

read\_inputs()

{

read -p "Enter first num: " num1

read -p "Enter second num: " num2

}

addition()

{

sum=$((num1+num2))

echo "The addition of $num1 and $num2 is: $sum"

}

subtraction()

{

sub=$((num1-num2))

echo "The sub of $Num1 and $num2 is: $sub"

}

read\_inputs

addition

subtraction

**Scope of the variables in functions and Returning variable value from a Function**

**Global variable:**

Suppose if you define a variable inside a function and that variable you can access it from outside of the function.

Suppose if you define a variable outside of a function and that variable you can access it from inside a function.

If you define any variable that’s by default global variable.

**Local variables:**

Local variables can only define in a function

We can access local variables only inside a functions.

If you want to access local variables from outside then we need to assign function o/p to a variable.

variable\_Name = $(function\_name)

echo “$variable\_name”

EX 1: Understanding Global and local variables

#!/bin/bash

display()

{

y=55

echo "The variable value of x is: $x"

local x=67

local p=78

}

x=5

display

echo "The value of a variable y is: $y"

echo "The x value after calling display function: $x"

echo "The value of p is: $p"

EX 2: To Access local variables from outside of a function.so we are assigning the function o/p to the variable and then we can print that variable.

#!/bin/bash

define\_variables()

{

local x=6

echo "$x"

}

y=$(define\_variables)

echo "The y value is: $y"

EX 3: To access the local variable from outside of a function using return command.

#!/bin/bash

return\_variable()

{

local x="5"

return $x

}

return\_variable

echo "The value of y is: $?"

**Note:**

Return command returns only integer values .it won’t return strings

EX 4: To access the local variable from outside of a function using return command.

#!/bin/bash

return\_variable()

{

local x="shell\_script"

return $x

}

return\_variable

echo "The value of y is: $?"

OUTPUT:

ERROR

**Passing parameters to a function:**

***Syntax:***

Function\_name $var1 $var2 …….

We can access passed parameters inside a functions using $1 $2 …….

Ex 1:

#!/bin/bash

addition()

{

#echo "The \$0 value is: $0"

m=$1

n=$2

result=$((m+n))

echo "The addition of $m and $n is: $result"

}

x=6

y=2

addition $x $y

p=3

q=7

addition $p $q

addition 4 9

**SECTION: 20**

**Printf command:**

▪Both echo and printf commands are used to display string or value of a variable.  
▪ The difference is that echo sends a newline at the end of its output, there is no way to "send" an EOF in printf command.

▪ The advantage of printf command:

▪ We can format the output

▪ Useful in awk command/scripting as well

▪ **Syntax:**

▪ printf “format\n” “arguments”

▪ printf “format\_with\_modifiers\n” “arguments”

▪ Note: format/format\_with\_modifiers is an optional and we can omit it.

**Ex 1: print entire row as “-“ symbol using printf & echo commands.**

**METHOD 1: Using printf**

Printf “%203s” “ “ | tr “ “ “-“

**METHOD 2: Using echo**

#!/bin/bash

columns=$(tput cols)

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

do

if [[ $i -eq columns ]]

then

echo “-”

break

fi

echo -n “-”

done

**Printf command with only format:**

Different types of formats are:

%d = For signed decimal numbers

%i = For signed decimal numbers

%u = For unsigned decimal numbers

%o = For unsigned octal numbers

%x = For unsigned hexadecimal numbers with lower case letters (a-f)

%X = For unsigned hexadecimal numbers with upper case letters (A-F)

%f = For floating point numbers

%s = For string

%% = For percent % symbol

X=5

Y=1.045

z=pavankumar

Printf “my\_name=%s my\_num=%d my\_float=%f\n” “$z” “$x” “$y”

Printf “%d\n” “$x”

Printf “%5d\n” “$x” ===> x value will print after 4 spaces i.e 5th position. So total 5

Printf “%05d\n” “$x” ====>0’s prints upto four positions after x value will print at 5th position .so total 5spaces.

Printf “%0.2f\n” “$y” ===> we will get only two fractional parts

Printf “%20.2f\n” “$y” ===> taking 20 spaces . we will get these two fractional parts value among those 20 spaces using right alignment.

Printf “%020.2f\n” “$y” ===> taking 20 spaces . we will get these two fractional parts value among those 20 spaces using right alignment.0’s will print at 20 spaces where there is spaces

Printf command format with modifiers:

Different types of format modifiers are

N = This specifies the width of the field for output

\* = This is the placeholder for the width

- = To left align output in the field. (Default: Right align)

0 = Pas result with leading 0s.

+ = To put + sign before positive numbers and – sign for negative numbers.

Printf “%40f\n” “$x”

[or]

P=40

Printf “%\*f\n” “$p” “$x”

[or]

P=40

Printf “%${p}f\n” “$x”

**Ex 1:**

#!/bin/bash

title=”welcome to git installation through shell script”

columns=$(tput cols)

title\_size=${#title}

span=$((($columns + $title\_size) / 2) )

printf “%${columns}s” “ “ | tr “ “ “\*”

printf “%${span}s\n” “$title “

printf “%${columns}s” “ “ | tr “ “ “\*”

**OUTPUT:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

welcome to git installation through shell script

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

tput cols ===>we will get no.of columns

tput lines ===>we will get no.of lines

tput rows ===>we will get no.of rows

#variable ===>will get size of the variable vaue

$@ and $\* How to get all command line arguments at a time

$? ===> whatever the previous we execute will give the status of that command.

$0 ===> shellscriptname

$# ==> we can give n.no of arguments

0 it is success

Non-zero it is failure

/passwordAuth ------------🡪 to search for string in editors.

Systemctl restart sshd ----🡪 restart ssh

Service sshd restart -🡪 restart ssh

One.txt |awk ‘NR!=1 {print}’ ===========>it won’t print the first line(no.of records not equal to 1 )

Seq 1 20 -------🡪 it will print 1 to 20 sequentially

Wc -l -🡪 it will count the total number of files

ls \*.txt ---->it displays all files which ends with .txt

exit --->if you want to terminate shell script then we will use exit command.

Break --🡪 if you want to terminate only loop then we will use break command

Continue -🡪 current iteration will skip and executes another iterations

\$A -🡪 we are using escape function to print as it is $A. if you don’t give \ then it will print o/p of $A

${ A[@] } --🡪 means all values from array

sleep 1

**We can run the file in two ways**

sh filename.sh

./filename.sh

If any command you are unable to install then check for epl for ubuntu repository in google and run the command