

LINUX-SHELL-SCRIPTING-AUTOMATION

---Introduction to Shell Scripting

- Shell Scripting is a great way to automate repetitive tasks in our Linux/Unix Environment.

-Shell Script is a sequence of system commands pasted in a text file.

****E.G** send some XYZ report to compliance team @every day.

Report: Version Info of Docker and Nginx and OS Uname

-We can enhance the shell scripts by using the below concepts:

****Variables**

****Conditional Statements**

****Loops**

****Functions**

****Job Scheduling and Many more**

-When we are running a command, it is executing through shell. There are different types of shells. We can see that by below--

```
#cat /etc/shells
```

```
/bin/sh
```

```
/bin/bash
```

```
/usr/bin/sh
```

```
/usr/bin/bash
```

```
/bin/tcsh
```

```
/bin/csh
```

```
/bin/zsh
```

We can see the current shell with

```
#echo $SHELL
```

After that if we need to change the shell to bash shell, we need to check the path of the shell.

```
#cat /etc/shells
```

```
/bin/sh
```

```
/bin/bash
```

```
/usr/bin/sh
```

```
/usr/bin/bash
```

```
/bin/tcsh
```

```
/bin/csh
```

```
/bin/zsh
```

Then we can change shell for that user with below command--

```
#chsh
```

---Before writing a shell script we have to write shebang line.

```
#!/bin/bash <=====This is the line what will tell us what shell we are using.
```

-Some commands will help to find the informations

```
**whatis java
```

```
**whatis pwd
```

```
**man java
```

```
**man pwd
```

--8. Redirection Operators and STDIN, STDOUT & STDERR

-Based on input redirection or output redirection. We have different types of operators.

****Output redirection operators**

****Input redirection operators**

****Combining redirection operators**

-Output redirection operators

> To create a new file

```
ls -l > test.txt
```

If we will do the same redirection with any other command, then the output will be replaced with single redirection.

```
cat test.txt
```

Filesystem	Size	Used	Avail	Capacity	iused	ifree	%iused	Mounted on
/dev/disk1s5	466Gi	10Gi	197Gi	6%	488378	4881964502	0%	/
devfs	195Ki	195Ki	0Bi	100%	681	0	100%	/dev
/dev/disk1s1	466Gi	257Gi	197Gi	57%	1240069	4881212811	0%	
/System/Volumes/Data								
/dev/disk1s4	466Gi	1.0Gi	197Gi	1%	1	4882452879	0%	/private/var/vm
map auto_home	0Bi	0Bi	0Bi	100%	0	0	100%	/System/Volumes/Data/home

If we will give double redirection then it will append the file with new output.

df >> test.txt

cat test.txt

Filesystem	Size	Used	Avail	Capacity	iused	ifree	%iused	Mounted on
/dev/disk1s5	466Gi	10Gi	197Gi	6%	488378	4881964502	0%	/
devfs	195Ki	195Ki	0Bi	100%	681	0	100%	/dev
/dev/disk1s1	466Gi	257Gi	197Gi	57%	1240069	4881212811	0%	/System/Volumes/Data
/dev/disk1s4	466Gi	1.0Gi	197Gi	1%	1	4882452879	0%	/private/var/vm
map auto_home	0Bi	0Bi	0Bi	100%	0	0	100%	/System/Volumes/Data/home

Filesystem	512-blocks	Used	Available	Capacity	iused	ifree	%iused	Mounted on
/dev/disk1s5	976490576	21947376	412606352	6%	488378	4881964502	0%	/
devfs	390	390	0	100%	681	0	100%	/dev
/dev/disk1s1	976490576	538336848	412606352	57%	1240080	4881212800	0%	/System/Volumes/Data
/dev/disk1s4	976490576	2097192	412606352	1%	1	4882452879	0%	/private/var/vm
map auto_home	0	0	0	100%	0	0	100%	/System/Volumes/Data/home

**This will not redirect the error output to a file.

#ll 1> test.out

-bash: ll: command not found

**But this will redirect the error output to the file.

#ll 2> test.out

-Input redirection Format ----- To provide the input

cat < test.txt

Filesystem	Size	Used	Avail	Capacity	iused	ifree	%iused	Mounted on
/dev/disk1s5	466Gi	10Gi	197Gi	6%	488378	4881964502	0%	/
devfs	195Ki	195Ki	0Bi	100%	681	0	100%	/dev
/dev/disk1s1	466Gi	257Gi	197Gi	57%	1240069	4881212811	0%	/System/Volumes/Data
/dev/disk1s4	466Gi	1.0Gi	197Gi	1%	1	4882452879	0%	/private/var/vm
map auto_home	0Bi	0Bi	0Bi	100%	0	0	100%	/System/Volumes/Data/home

Filesystem	512-blocks	Used	Available	Capacity	iused	ifree	%iused	Mounted on
/dev/disk1s5	976490576	21947376	412606352	6%	488378	4881964502	0%	/
devfs	390	390	0	100%	681	0	100%	/dev
/dev/disk1s1	976490576	538336848	412606352	57%	1240080	4881212800	0%	/System/Volumes/Data
/dev/disk1s4	976490576	2097192	412606352	1%	1	4882452879	0%	/private/var/vm
map auto_home	0	0	0	100%	0	0	100%	/System/Volumes/Data/home

-Combining redirection

-To send the standard output of one command to another command as a standard input.

For example----

```
ls -lrt
total 0
drwxr-xr-x+ 27 ASUTOSH  staff  864 Jan  3  2020 ASUTOSH-OLD
drwxr-xr-x  16 root    admin  512 Jan 25  2021 test-function
drwxrwxrwt  15 root    wheel  480 Jun 28 15:39 Shared
drwxr-xr-x 143 ASUTOSH  admin  4576 Jul 19 21:18 MY DATA-OLD
drwxr-xr-x  78 ASUTOSH  admin  2496 Sep 14 01:47 ASUTOSH

(base) ASUTOSH-MACBOOK-PRO:Users ASUTOSH$ ls -lrt | awk {'print $1'}
```

```
total
drwxr-xr-x+
drwxr-xr-x
drwxrwxrwt
drwxr-xr-x
drwxr-xr-x
```

```
(base) ASUTOSH-MACBOOK-PRO:Users ASUTOSH$ ls -lrt | awk {'print $1'} | grep t
total
drwxrwxrwt
```

---Store java version into a file using redirection operators

****How to separate STDOUT and STDERR**

***Solution - Using file descriptors**

*******A file descriptor is an integer to identify STDIN, STDOUT and STDERR.**

***0 ---->>> STDIN**

***1 ----->>> STDOUT**

***2 ----->>> STDERR**

ls >1 test.out, or ls > test.out is the same command, if the the command is successful.

Now if we run

ls >2 test.out <<< It will store only the error message in the file and success output in the cmd

```
ls 2> test.out
.localized  2      ASUTOSH    ASUTOSH-OLD  MY DATA-OLD  Shared
test-function test.out
```

```
--
ls 1> test.txt 2> err.txt
```

```
sh-3.2# cat test.txt
```

```
.localized
```

```
2
```

```
ASUTOSH
```

```
ASUTOSH-OLD
```

```
MY DATA-OLD
```

```
Shared
```

```
err.txt
```

```
test-function
```

```
test.out
```

```
test.txt
```

```
sh-3.2# cat err.txt
```

```
<blank>
```

```
-----
java -version 2> java.txt
```

```
sh-3.2# cat java.txt
```

```
java version "16.0.2" 2021-07-20
```

```
Java(TM) SE Runtime Environment (build 16.0.2+7-67)
```

```
Java HotSpot(TM) 64-Bit Server VM (build 16.0.2+7-67, mixed mode, sharing)
```

```
-----
-We can also specify to store error and success output into a single file.
```

```
java -version &> java.txt
```

```
sh-3.2# cat java.txt
```

```
java version "16.0.2" 2021-07-20
```

```
Java(TM) SE Runtime Environment (build 16.0.2+7-67)
```

```
Java HotSpot(TM) 64-Bit Server VM (build 16.0.2+7-67, mixed mode, sharing)
```

--Commands to read a file content

****Read a file content with opening it**
we have vi/nano etc

****Read a file content without opening it**
We have cat/less/more etc

****If we want to read the contents of the file with line number.**

#cat -n <file-name>

```
#cat -n swapuse.sh
```

```
1 #!/bin/bash
```

```
2 SUM=0
```

```
3 OVERALL=0
```

```
4 for DIR in `find /proc/ -maxdepth 1 -type d -regex "^/proc/[0-9]+"`
```

```
5 do
```

```
6 PID=`echo $DIR | cut -d / -f 3`
```

```
7 PROGNAME=`ps -p $PID -o comm --no-headers`
```

```
8 for SWAP in `grep VmSwap $DIR/status 2>/dev/null | awk '{ print $2 }`
```

```
9 do
```

```
10 let SUM=$SUM+$SWAP
```



```
11 done
12 if (( $SUM > 100000 )); then
13 echo "PID=$PID swapped $SUM KB ($PROGNAME)"
14 fi
15 let OVERALL=$OVERALL+$SUM
16 SUM=0
17 done
18 echo "Overall swap used: $OVERALL KB"
```

**Read file content with conditions

using more,head,tail,grep,awk,sed

**more -n <file-name> <====This will show first n line and then it will wait.

**more +n <file-name> <====This will show the file from the nth line and then it will wait

**head <file-name> <===== by default it will show the first 10 line

**head -n <file-name> <<===== it will show the top n line

cat -n swapuse.sh | head -10

```
1 #!/bin/bash
2 SUM=0
3 OVERALL=0
4 for DIR in `find /proc/ -maxdepth 1 -type d -regex "^/proc/[0-9]+"`
5 do
6 PID=`echo $DIR | cut -d / -f 3`
7 PROGNAME=`ps -p $PID -o comm --no-headers`
8 for SWAP in `grep VmSwap $DIR/status 2>/dev/null | awk '{ print $2 }`
9 do
10 let SUM=$SUM+$SWAP
```

****tail <file-name> <<<===** By default it will show the last 10 lines of the file

****tail -n <file-name> <=====** It will show the last n lines

--How to read required range of lines from a given file

****Need to show line 6 to line 12 of the file**

cat test.out

LINE 1
LINE 2
LINE 3
LINE 4
LINE 5
LINE 6
LINE 7
LINE 8
LINE 9
LINE 10
LINE 11
LINE 12
LINE 13
LINE 14
LINE 15
LINE 16
LINE 17
LINE 18
LINE 19
LINE 20

```
cat test.out | head -12 | tail -7
```

LINE 6
LINE 7
LINE 8
LINE 9
LINE 10
LINE 11
LINE 12

```
cat test.out | awk 'NR>=6 && NR<=12 {print}' #Line Number greater than equals to 6 && line number <= 12
```

LINE 6
LINE 7
LINE 8
LINE 9
LINE 10
LINE 11
LINE 12

```
cat test.out | sed -n '6,12p' #starting from 6th line to 12 th line print.
```

LINE 6
LINE 7
LINE 8
LINE 9
LINE 10
LINE 11
LINE 12

```
--Basic usage of grep command
```

-grep command is use to search in files, multiple files, directory, and text also.

****Simple grep command syntax:**

grep [options] "string/pattern" file/files

cat file | grep [options] "string/pattern"

echo "some text" | grep [options] "string/pattern"

Basicoptions: -i -w -v -o -n -c -A -B -C -r -l -h

Advanced Options: -e -f and -E

****Basicoptions: -i -w -v -o -n -c -A -B -C -r -l -h**

grep "string/pattern" file/files

grep [options] "string/pattern" file/files

-i - To ignore case for matching/searching

-w - To match a whole word

-v - To display the lines which are not having given string or text

-o - To print/display only matched parts from matched lines

-n - To display the matched line numbers

-c - To display matched number of lines

-A - To display N lines after match (grep -A 3 "string" file)

-B - To display N lines before match

-C - To display N lines around match

-r - To search under current directory and its sub-directory

-l - To display only file names

-h - To hide file names

--Advanced grep option

grep command syntax:

grep [options] "string/pattern" file/files

Basic options: -i -w -v -o -n -c -A -B -C -r -l -h

---Advanced Options: -f -e and -E

-f Takes search string/pattern from a file, one per line

-e To search multiple strings/patterns

Pattern is a string and it represents more than one string.

-E To work with patterns

grep -E [options] "pattern" file/files

----- Rules to create patterns:

xy|pq Matches for xy or pq

^xyz Matches for the lines which are starting with "xyz"

xyz\$ Matches for the lines which are ending with "xyz"

^\$ Matches for the lines which are empty

\ To remove the special purpose of any symbol. Ex: \^ \\$

. Matches any character

\. Matches exactly with .

\b Match the empty string at the edge of the word

Rules to create patterns:

`[:alnum:]` `[:alpha:]` `[:blank:]` `[:digit:]` `[:lower:]` `[:space:]` `[:upper:]`

- Alphanumeric characters.
- Alphabetic characters
- Blank characters: space and tab.
- Digits: '0 1 2 3 4 5 6 7 8 9'.
- Lower-case letters: 'a b c d e f g h i j k l m n o p q r s t u v w x y z'.
- Space characters: tab, newline, vertical tab, form feed, carriage return, and space.
- Upper-case letters: 'A B C D E F G H I J K L M N O P Q R S T U V W X Y Z'.

--Simple grep command with example

-Write the grep command to list only the directories.

for example--

```
-rw-r--r--@ 1 ASUTOSH admin 38916 Sep 18 02:58 .DS_Store
drwxr-xr-x@ 43 ASUTOSH admin 1376 Jun 3 09:26 Ansible Essential Training
drwxr-xr-x@ 26 ASUTOSH admin 832 Aug 18 04:01 Complete Python Scripting
drwxr-xr-x 27 ASUTOSH admin 864 Jun 8 2020 DEVOPS-PREREQUISITES
drwxr-xr-x 3 ASUTOSH admin 96 May 23 2020 Docker
drwxrwxr-x@ 3 ASUTOSH admin 96 Jun 10 2016 Getting familiar with the command line
drwxr-xr-x 3 ASUTOSH admin 96 Jul 17 2020 Kubernetes-Basics
drwxr-xr-x 5 ASUTOSH admin 160 Jul 11 2020 LINUX-BASIC-COURSE
drwxr-xr-x@ 24 ASUTOSH admin 768 Jun 3 09:24 Learning Ansible
drwxr-xr-x@ 36 ASUTOSH admin 1152 Sep 6 2020 Learning Docker
drwxr-xr-x@ 19 ASUTOSH admin 608 Sep 6 2020 Learning Git and GitHub
drwxr-xr-x@ 37 ASUTOSH admin 1184 Sep 7 2020 Learning Kubernetes
drwxr-xr-x@ 45 ASUTOSH admin 1440 Sep 15 2020 Learning Linux Shell Scripting
```

****I want to list only the directories.**

NOte - Directories starts with drwxr

ll | grep ^d <=====It will list ot the files/directories starts with d. So it will print only directories.

```
drwxr-xr-x@ 43 ASUTOSH admin 1376 Jun 3 09:26 Ansible Essential Training
drwxr-xr-x@ 26 ASUTOSH admin 832 Aug 18 04:01 Complete Python Scripting
drwxr-xr-x 27 ASUTOSH admin 864 Jun 8 2020 DEVOPS-PREREQUISITES
drwxr-xr-x 3 ASUTOSH admin 96 May 23 2020 Docker
drwxrwxr-x@ 3 ASUTOSH admin 96 Jun 10 2016 Getting familiar with the command line
drwxr-xr-x 3 ASUTOSH admin 96 Jul 17 2020 Kubernetes-Basics
drwxr-xr-x 5 ASUTOSH admin 160 Jul 11 2020 LINUX-BASIC-COURSE
drwxr-xr-x@ 24 ASUTOSH admin 768 Jun 3 09:24 Learning Ansible
drwxr-xr-x@ 36 ASUTOSH admin 1152 Sep 6 2020 Learning Docker
drwxr-xr-x@ 19 ASUTOSH admin 608 Sep 6 2020 Learning Git and GitHub
drwxr-xr-x@ 37 ASUTOSH admin 1184 Sep 7 2020 Learning Kubernetes
drwxr-xr-x@ 45 ASUTOSH admin 1440 Sep 15 2020 Learning Linux Shell Scripting
drwxr-xr-x 28 ASUTOSH admin 896 Sep 1 01:33 Linux-Bash-Scripts
drwxr-xr-x 16 ASUTOSH admin 512 Sep 18 03:03 Linux-Shell-Automation
drwxr-xr-x@ 21 ASUTOSH admin 672 Apr 25 21:35 Python Course with Notes
drwxr-xr-x 52 ASUTOSH admin 1664 May 24 22:06 Python-new
```

-Write grep command to list only files.

If it is a file, then it will start withb "-"

```
sh-3.2# ll | grep ^- <=====It will list the files start with "-"  
-rw-r--r--@ 1 ASUTOSH admin 38916 Sep 18 02:58 .DS_Store
```

```
-rw-r--r--@ 1 ASUTOSH admin 28965 Aug 25 2020 RAILS-ARCHITECTURE-.pdf
-rw-r--r-- 1 ASUTOSH admin 120 Jul 3 22:52 my-workspace.code-workspace
```

--Find the server's ipv4 info from a file

```
cat ip.txt | grep -E "\b[0-9](1,3)\.[0-9](1,3)\.[0-9](1,3)\.[0-9](1,3)\b"
```

```
[0-9](1,3)\.
```

0-9 means letters from 0-9 in the field.

1,3 means from 1 to 3 digit number in the field

\. means it will match with the .

\b is the space before

--CUT command

--Complete cut command in one video

****The 'cut' command is a powerful tool to extract parts of each line from a file.**

****It is based on**

Byte Position

Character Position

Fields based on delimiter (by default delimiter is the tab)

Cut command syntax:

cut [options] <positions(fields) /range of positions(fields)> <input_file>

cat file | cut [options] <positions(fields) /range of positions(fields)>

Options: -b -c and -f

Rages:

2 only second byte/character/field

2- second byte/character/field to last

-7 first to seven

3,5 third and fifth

Cut command for Byte/Character Position:

To cut out a section of a line by specifying a byte/character position use the -b/-c option.

*Syntax:

cut -b <position's/range of position's> file

cut -c <position's/range of position's> file

Position's: 3,5,10

Range of Position's: 3-7, 6-10

Ex: mytext.txt

cut -b 2 mytext.txt

cut -b 3,7 mytext.txt

cut -b 5-9 mytext.txt

cut -b 5- mytext.txt

cut -b -7, 9 mytext.txt

Use --complement to complement the output

****Cut command for field position**

To cut out a section of a line by specifying a field position use the -f option.

Assume fields are like columns, by default cut command will separate columns based on tab(delimiter).

If we want to use different field separator use -d (delimiter).

****Syntax:**

cut -f <position's/range of position's> file

cut -f <position's/range of position's> [-d ':'] [--output-delimiter='**'] file

-d is a delimiter like @ , : / etc....

Position's: 3,5,2

Range of Position's: 3-7, 6-10

Ex: mytext.txt

cut -f 2 mytext.txt

cut -f 3,7 mytext.txt

cut -f 5-9 mytext.txt

cut -f 5- mytext.txt

cut -f -7, 9 --output-delimiter=" " mytext.txt

ls -lrt | cut -c 1

t
d
d
d
d
d
d
d
d
d

ls -lrt | cut -c 4

a
x
x
x
x
x
x
x
x

ls -lrt | cut -c 1-4

tota
drwx
drwx
drwx
drwx
drwx
drwx
drwx
drwx
-rw-
drwx

ls -lrt | cut -c 1-9,10

total 160
drwxrwxr-x
drwxr-xr-x
drwxr-xr-x
drwxr-xr-x
drwxr-xr-x

```
drwxr-xr-x
drwxr-xr-x
drwxrwxr-x
```

lets say to get 4th charcter to the last character from each and evry line

```
ls -lrt | cut -c 4-
al 160
```

```
xrwxr-x@ 3 ASUTOSH admin 96 Jun 10 2016 Getting familiar with the command line
xr-xr-x 3 ASUTOSH admin 96 May 23 2020 Docker
xr-xr-x 27 ASUTOSH admin 864 Jun 8 2020 DEVOPS-PREREQUISITES
xr-xr-x 5 ASUTOSH admin 160 Jul 11 2020 LINUX-BASIC-COURSE
xr-xr-x 5 ASUTOSH admin 160 Jul 11 2020 SHELL-SCRPTING-BASIC
xr-xr-x 3 ASUTOSH admin 96 Jul 17 2020 Kubernetes-Basics
```

--Cut commands based on field. Fields means coulumns.

**Field separator is the delemeter.(by -default tab)

--I want to get the second field from this file.

```
cat /etc/passwd | cut -f 2-5 -d":"
```

```
##
```

```
# User Database
```

```
#
```

```
# Note that this file is consulted directly only when the system is running
```

```
# in single-user mode. At other times this information is provided by
```

```
# Open Directory.
```

```
#
```

```
# See the opendirectoryd(8) man page for additional information about
```

```
# Open Directory.
```

```
##
```

*:2:2:Unprivileged User
*:0:0:System Administrator
*:1:1:System Services
*:4:4:Unix to Unix Copy Protocol
*:13:13:Task Gate Daemon
*:24:24:Network Services
*:25:25:Install Assistant
*:26:26:Printing Services

**Here : as the delemeter/field separator

--Use the cut command to get the exact version number

for example---

httpd -v

Server version: Apache/2.4.46 (Unix)

Server built: Apr 12 2021 01:44:06

**If we will grep for version , we will get the first line

httpd -v | grep version

Server version: Apache/2.4.46 (Unix)

httpd -v | grep version | cut -d "/" -f 2 | cut -d "(" -f 1
2.4.46

Then first we can cut the output with "/"

httpd -v | grep version | cut -d "/" -f 2

2.4.46 (Unix)

Then we can cut the output with "(" and print the first field

We can also separate the values with " " (space)

```
httpd -v | grep version | cut -d "/" -f 2 | cut -d " " -f 1  
2.4.46
```

--We can also store the above value into a variable

```
httpd_version=$(httpd -v | grep version | cut -d "/" -f 2 | cut -d "(" -f 1)
```

```
sh-3.2# echo $httpd_version  
2.4.46
```

--AWK command

The awk command is a powerful method for processing or analyzing text or data files , which are organized by lines (rows or records) and columns(fields).

we can use awk as a linux command and also as a scripting language like bash shell scripting.

****Simple awk command syntax:**

```
awk [options] '[selection_criteria] {action }' input-file  
cat input-file | awk [options] '[selection_criteria] {action }' input-file
```

****Awk can take the following options:**

- F fs To specify a field separator. (Default separator is tab and space)
- f file To specify a file that contains awk script.
- v var=value To declare a variable.

Selection criteria: pattern/condition

Action: It is a logic to perform action on each row/record

***Simple awk command syntax:

awk ' {action }' input-file

Action: Action is a logic to perform action on each record. Example: print \$1 print first field from each line

Some of the default variables for awk:

\$0 -->Entire file

\$1 -->First field from each line/record

\$2 --->Second field from each line/record

NR ---->It will print line or record number <=====Thsi will print the line with the number

NF ---> It will print number of filed from each line/record <=====This will give us the last record of each line

-SO we are using cut command as below to find out the exact version number

```
sh-3.2# httpd -v | grep version | cut -d "/" -f 2 | cut -d " " -f 1
2.4.46
```

--We can also use AWK command.

```
httpd -v
Server version: Apache/2.4.46 (Unix)
Server built: Apr 12 2021 01:44:06
```

cat input-file | awk [options] '[selection_criteria] {action }' input-file

```
httpd -v | awk '/version/ {print}'
Server version: Apache/2.4.46 (Unix)
```

```
httpd -v | awk -F '[ /]' '/version/ {print $4}'
2.4.46
```

*****Here -F is the field separator with space and / [/] , so we can assign two field separators at a time here.

Note - With cut command we can get characters, but when we need to get the field then we can use AWK command.

```
sh-3.2# cat /etc/passwd |awk -F ':' 'NR==13 {print}'
daemon*:1:1:System Services:/var/root:/usr/bin/false
```

If we are taking "\$0" or only print , that means it will print the entire file.

By default the output field separator is also "-" space

This will show the line number of each line and the number of fields on each line

```
#cat /etc/passwd |awk -F ':' '{print NR,$0,NF}'
```

```
12 root*:0:0:System Administrator:/var/root:/bin/sh 7
```

--tr command for shell scripting

tr: short for translate

tr is useful to translate or delete given set of characters from the input.

Syntax:

tr [options] [SET1] [SET2] <inputFile

Some Command | tr [options] [SET1] [SET2]

No Option: For translation

Examples for SET1/SET2: [:lower:], [a-z], [:upper:] [A-Z] , [:digit:] , [0-9], [:space:]

-d : deletes given set of characters.

It is used to translate files

cat xyz.txt

Line 1

Line 2

Line 3

Line 4

Line 5

Line 6

Line 7

Line 8

Line 9

Line 10

Line 11

Line 12

tr '[:upper:]' '[:lower:]' <xyz.txt

line 1

line 2
line 3
line 4
line 5
line 6
line 7
line 8
line 9
line 10
line 11
line 12

tr '[:lower:]' '[:upper:]' <xyz.txt

LINE 1
LINE 2
LINE 3
LINE 4
LINE 5
LINE 6
LINE 7
LINE 8
LINE 9
LINE 10
LINE 11
LINE 12

-Only i

tr 'i' 'I' <xyz.txt

LIne 1
LIne 2

LIne 3
LIne 4
LIne 5
LIne 6
LIne 7
LIne 8
LIne 9
LIne 10
LIne 11
LIne 12

--tee command for shell scripting

-tee command is used to display the output and also to store the output into a file(it does both the tasks simultaneoulsy)

- It is useful to create logs for shell scripting

- syntax

command | tee outputfile.txt

I want to dsisplay the output of the command and also to store the output into a file.

ls -lrt | tee /tmp/abc.txt

total 2992

-rw-r--r--@ 1 ASUTOSH admin 241882 Sep 18 02:57 1.Document-grep-command-part-1.pdf

-rw-r--r--@ 1 ASUTOSH admin 263618 Sep 18 02:59 2.Document-grep-command-part-2.pdf

-rw-r--r--@ 1 ASUTOSH admin 306874 Sep 18 03:01

1.Document-complete-cut-command.pdf

```
-rw-r--r--@ 1 ASUTOSH admin 230130 Sep 18 03:01 1.Document-awk+command+part-1.pdf
-rw-r--r--@ 1 ASUTOSH admin 67123 Sep 18 03:01 1.+tr+command.pdf
-rw-r--r--@ 1 ASUTOSH admin 261 Sep 18 03:01 practice_on_variables.sh.txt
-rw-r--r--@ 1 ASUTOSH admin 73695 Sep 18 03:01
4.+Advanced+usage+of+echo+command.pdf
-rw-r--r--@ 1 ASUTOSH admin 2133 Sep 18 03:01 4.color_codes.txt
```

If you want to append the output everytime you run the command to the same file, use

```
#command | tee -a abc.txt
```

```
ls | tee -a /tmp/abc.txt
```

```
1.+Basic+String+Operations.pdf
1.+tr+command.pdf
1.Document-awk+command+part-1.pdf
1.Document-complete-cut-command.pdf
1.Document-grep-command-part-1.pdf
1.Document-input-and-out-commands.sh
2.+String+Operations+on+Paths.pdf
2.Document-command-line-arguments.sh
2.Document-grep-command-part-2.pdf
4.+Advanced+usage+of+echo+command.pdf
4.color_codes.txt
5.+Here+Document+for+Multi-lines+or+Multi-line+block.pdf
9.+Debugging+a+Bash+Shell+Script.pdf
practice_on_variables.sh.txt
xyz.txt
```

--

```
cat /tmp/abc.txt
```

```
total 2992
```

-rw-r--r--@ 1 ASUTOSH admin 241882 Sep 18 02:57 1.Document-grep-command-part-1.pdf
-rw-r--r--@ 1 ASUTOSH admin 263618 Sep 18 02:59 2.Document-grep-command-part-2.pdf
-rw-r--r--@ 1 ASUTOSH admin 306874 Sep 18 03:01
1.Document-complete-cut-command.pdf
-rw-r--r--@ 1 ASUTOSH admin 230130 Sep 18 03:01 1.Document-awk+command+part-1.pdf
-rw-r--r--@ 1 ASUTOSH admin 67123 Sep 18 03:01 1.+tr+command.pdf
-rw-r--r--@ 1 ASUTOSH admin 261 Sep 18 03:01 practice_on_variables.sh.txt
-rw-r--r--@ 1 ASUTOSH admin 73695 Sep 18 03:01
4.+Advanced+usage+of+echo+command.pdf
-rw-r--r--@ 1 ASUTOSH admin 2133 Sep 18 03:01 4.color_codes.txt
-rw-r--r--@ 1 ASUTOSH admin 78140 Sep 18 03:01
5.+Here+Document+for+Multi-lines+or+Multi-line+block.pdf
-rw-r--r--@ 1 ASUTOSH admin 93284 Sep 18 03:01 9.+Debugging+a+Bash+Shell+Script.pdf
-rw-r--r--@ 1 ASUTOSH admin 73347 Sep 18 03:01 1.+Basic+String+Operations.pdf
-rw-r--r--@ 1 ASUTOSH admin 63347 Sep 18 03:01 2.+String+Operations+on+Paths.pdf
-rw-r--r--@ 1 ASUTOSH admin 191 Sep 18 03:02 1.Document-input-and-out-commands.sh
-rw-r--r--@ 1 ASUTOSH admin 226 Sep 18 03:02 2.Document-command-line-arguments.sh
-rw-r--r-- 1 root admin 87 Sep 25 04:33 xyz.txt
1.+Basic+String+Operations.pdf
1.+tr+command.pdf
1.Document-awk+command+part-1.pdf
1.Document-complete-cut-command.pdf
1.Document-grep-command-part-1.pdf
1.Document-input-and-out-commands.sh

--Basics of bash shell scripting

--Hello-world script file

cat hello-world.sh

#!/bin/bash

--Introduction to variables

```
cat hello-world.sh
```

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

```
echo "Hello World"
```

```
#Lets take an example of variable
```

```
s="Bash Shell Scripting"
```

```
echo "Welcome to $s"
```

```
echo "$s is powerful in linux Env"
```

```
echo "Now we wre working with variables concept of $s"
```

```
chmod +x hello-world.sh
```

```
sh-3.2# ./hello-world.sh
```

```
Hello World
```

```
Welcome to Bash Shell Scripting
```

```
Bash Shell Scripting is powerful in linux Env
```

```
Now we wre working with variables concept of Bash Shell Scripting
```

-Variables are useful to store data in shell scripts and later we can use them if they required.

e.g - X=4

```
sh-3.2# x=4
```

```
sh-3.2# echo $x
```

```
4
```

```
sh-3.2# echo "$x"
```

```
4
```

```
sh-3.2# echo '$x'
```

\$x #<===== If we will take single quote, then it will not substitute the value of x. It will take the entire value.

-Default value of a string is empty/nothing

```
echo $y
```

```
--
```

```
sh-3.2# echo $BASH
```

```
/bin/sh
```

```
sh-3.2# echo $USER
```

```
root
```

```
-----
```

-In linux shell scripting, there are two types of variables----

****System variables**

Created and maintained by Operating system itself

This type of variables are defined in CAPITAL LETTERS

*****We can see them by set command.

```
#set
```

e.g---

```
OSTYPE=darwin19
```

```
PATH='/Library/Frameworks/Python.framework/Versions/3.8/bin:/Users/ASUTOSH/Anaconda/anaconda3/bin:/Users/ASUTOSH/Anaconda/anaconda3/condabin:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/Applications/VMware Fusion.app/Contents/Public'
```

```
PIPESTATUS=([0]="0")
```

```
POSIXLY_CORRECT=y
```

PPID=2956
PS1='\s-\v\\$ '
PS2='> '
PS4='+ '
PWD=/Users/ASUTOSH/COURSES/Linux-Shell-Automation
SHELL=/bin/sh
SHELLOPTS=braceexpand:emacs:hashall:histexpand:history:interactive-comments:monitor:posix
SHLV=1
SSH_AUTH_SOCK=/private/tmp/com.apple.launchd.Tgqp3lph6h/Listeners
SUDO_COMMAND=/usr/bin/su
SUDO_GID=20
SUDO_UID=501
SUDO_USER=ASUTOSH
TERM=xterm
UID=0
USER=root
_root
__CF_USER_TEXT_ENCODING=0x0:0:0
x=4
e.g-
HOME,USER

****User defined variables**

Created and maintained by the users

This type of variables are defined in lower letters

We can also take the combination of lower and upper case letters.

-Rules to define User Defined variables

-Variable name should contain only a-z or A-Z or 0-9 and _ characters

- Variable length should be less than or equals to 20 characters
 - Variable names are case sensitive.
 - Do not provide space on either sides of equal symbol
 - no need to declare variable type, it will take automatically, while executing the command/script.
 - use quotes for the data, if data consists spaces.
 - We can store the output of a command into a variables as follows---
- ```
#val=$(command)
#my_val=`command`
```

e.g-

```
#value=$(date)
#echo $value
Wed Sep 29 03:11:21 IST 2021
```

If I want to see only time

```
#echo $value | awk -F '[' '{print $4}'
03:11:21
```

- We can assign one variable value/data into another using---
- ```
Name="Shell Scripting"
NewName=$Name
or
NewName=${Name}
-----
```

```
sh-3.2# Name="Asutosh"
```

```
sh-3.2# NewNamae=$Name
```

```
sh-3.2# echo $NewNamae
```

```
Asutosh
```

```
sh-3.2# NewName=${Name}
```

```
sh-3.2# echo $NewNam
```

```
sh-3.2# echo $NewName
```

```
Asutosh
```

-Simple shell script to know the usage of variables(print docker status and version)

-If I want see the status of docker command.

I want to see the output as running/stopped

```
systemctl status docker | grep Active
```

```
Active: active (running) since Tue 2021-09-28 15:19:31 PDT; 2min 8s ago
```

We can use awk command to get that

```
#systemctl status docker |awk '/Active/ {print $3}'  
(running)
```

Now I want get only running. Then we can remove the paranthesis through "tr" command.
This will translate the output

```
#systemctl status docker |awk '/Active/ {print $3}' | tr -d "[]"  
running
```

---Now in docker -v, we want only the exact version.

-We can print the value with awk.

```
[root@m2-maprts-vm248-172 ~]# docker -v | awk '{print $3}'  
1.13.1,
```

But we want only the version number

Either we can use here "cut -d" or "tr -d", both will give us the same result.

```
#docker -v | awk '{print $3}' | cut -d "," -f 1  
1.13.1
```

```
#docker -v | awk '{print $3}' | tr -d "[,]"  
1.13.1
```

-----Now we can write a simple shell script---

```
cat docker-status.sh  
#!/bin/bash  
dockerstatus=$(systemctl status docker |awk '/Active/ {print $3}' | tr -d "[()]" )  
dockerversion=$(docker -v | awk '{print $3}' | tr -d "[,]" )  
  
echo "The docker status is $dockerstatus"  
  
echo "The docker version is $dockerversion"
```

```
-----  
./docker-status.sh  
The docker status is running  
The docker version is 1.13.1
```

--Advanced usage of echo command

echo command is used to display string/message or variable value or command result.

Simple syntax:

echo message/string

echo "message/string"

echo "message/string with some variable \$xyz"

echo "message/string/\$variable/\$(command)"

Advanced usage (to execute escape characters):

echo -e "Message/String or variable"

Escape Characters:

\n <--- New Line

\t <--- Horizontal Tab

\v <--- Vertical Tab

\b <--- Backspace

\r <--- Carriage Return etc...

To display message in colors.

=====

Reset

Color_Off='\033[0m' # Text Reset

Regular Colors

Black='\033[0;30m' # Black

Red='\033[0;31m' # Red

Green='\033[0;32m' # Green

Yellow='\033[0;33m' # Yellow

Blue='\033[0;34m' # Blue

Purple='\033[0;35m' # Purple

Cyan='\033[0;36m' # Cyan
White='\033[0;37m' # White

Bold
BBlack='\033[1;30m' # Black
BRed='\033[1;31m' # Red
BGreen='\033[1;32m' # Green
BYellow='\033[1;33m' # Yellow
BBlue='\033[1;34m' # Blue
BPurple='\033[1;35m' # Purple
BCyan='\033[1;36m' # Cyan
BWhite='\033[1;37m' # White

Underline
UBlack='\033[4;30m' # Black
URed='\033[4;31m' # Red
UGreen='\033[4;32m' # Green
UYellow='\033[4;33m' # Yellow
UBlue='\033[4;34m' # Blue
UPurple='\033[4;35m' # Purple
UCyan='\033[4;36m' # Cyan
UWhite='\033[4;37m' # White

Background
On_Black='\033[40m' # Black
On_Red='\033[41m' # Red
On_Green='\033[42m' # Green
On_Yellow='\033[43m' # Yellow
On_Blue='\033[44m' # Blue
On_Purple='\033[45m' # Purple
On_Cyan='\033[46m' # Cyan
On_White='\033[47m' # White

High Intensity

IBlack='\033[0;90m' # Black
IRed='\033[0;91m' # Red
IGreen='\033[0;92m' # Green
IYellow='\033[0;93m' # Yellow
IBlue='\033[0;94m' # Blue
IPurple='\033[0;95m' # Purple
ICyan='\033[0;96m' # Cyan
IWhite='\033[0;97m' # White

Bold High Intensity

BIBlack='\033[1;90m' # Black
BIRed='\033[1;91m' # Red
BIGreen='\033[1;92m' # Green
BIYellow='\033[1;93m' # Yellow
BIBlue='\033[1;94m' # Blue
BIPurple='\033[1;95m' # Purple
BICyan='\033[1;96m' # Cyan
BIWhite='\033[1;97m' # White

High Intensity backgrounds

On_IBlack='\033[0;100m' # Black
On_IRed='\033[0;101m' # Red
On_IGreen='\033[0;102m' # Green
On_IYellow='\033[0;103m' # Yellow
On_IBlue='\033[0;104m' # Blue
On_IPurple='\033[0;105m' # Purple
On_ICyan='\033[0;106m' # Cyan
On_IWhite='\033[0;107m' # White

=====

echo -n "message/string/\$variable/\${command}"

We can use echo commands like this---

#echo "The currently loggedin user is: \$(whoami)"

The currently loggedin user is: root

#!/bin/bash

echo "This is first line"

echo "This is the second line"

#We can write both the lines in one echo command

echo -e "This is the first line\nThis is the second line"

sh-3.2# bash Advanced-usage-echo.sh

This is first line

This is the second line

This is the first line

This is the second line

cat Advanced-usage-echo.sh

#!/bin/bash

echo "This is first line"

echo "This is the second line"

#We can write both the lines in one echo command
echo -e "This is the first line\nThis is the second line"

echo -e "This is the first line\tThis is the second line"

echo -e "This is the first line\bThis is the second line"

echo -e "This is the first line\rThis is the second line"

#Now to escape the characters
echo -e "This is the \"bash\" line"

sh-3.2# ./Advanced-usage-echo.sh
This is first line
This is the second line
This is the first line
This is the second line
This is the first line This is the second line
This is the first linThis is the second line
This is the second line
This is the "bash" line

-To escape the escape charactors we need to add two times "\"

echo -e "This is the first line\\rThis is the second line"

This is the first line\rThis is the second line

Whenever we are using echo -n "" It will not send the cursor to the next line, and the consecutive messages will print in one line.

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

```
echo "This is first line"  
echo -n "This is the third line"  
echo " this is the fourth line"
```

```
-----  
sh-3.2# ./Advanced-usage-echo.sh  
This is first line  
This is the second line  
This is the third line this is the fourth line
```

```
-----  
--Here Document for multilines or multi-line block  
-----
```

Heredoc is very useful to write multi-lines or multiline block.

```
Syntax--  
command << DELIMITER  
Line-1  
Line-2  
Line-3  
DELIMITER
```

Note - Here DELIMITER can be any string.

Heredoc is mostly used with the combination of cat command.

Display multi-lines using cat command.

```
cat << DELIMITER
```

```
Line-1
```

```
Line-2
```

```
Line-3
```

```
DELIMITER
```

```
-----
```

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

```
echo "The user is: $USER"
```

```
echo "The home for this user is: $HOME"
```

```
#So with echo we can display multiline block
```

```
echo "
```

```
The user is: $USER
```

```
The home for this user is: $HOME"
```

```
-----
```

```
sh-3.2# ./usage-of-here-doc.sh
```

```
The user is: root
```

```
The home for this user is: /var/root
```

```
The user is: root
```

```
The home for this user is: /var/root
```

```
-----
```

```
Same we can use with heredoc
```

```
-----
```

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

```
cat << EOF
```

```
The user is $USER
```

```
The home for this $USER is $HOME
```

```
EOF
```

```
-----
```

```
sh-3.2# ./usage-of-here-doc.sh
```

```
The user is root
```

```
The home for this root is /var/root
```

```
-----
```

****We can also use any name with the combination of capital and small**

```
-----
```

```
cat << Asutosh
```

```
The user is: $USER
```

```
The home for this user $USER is: $HOME
```

```
Asutosh
```

```
-----
```

```
sh-3.2# ./usage-of-here-doc.sh
```

```
The user is: root
```

```
The home for this user root is: /var/root
```

```
-----
```

We can also redirect this heredoc result into a file or as a input for another command.

#we can also redirect the output to any file

```
cat << EOF > /tmp/demo.txt
```

```
The user is $USER
```

```
The home for this $USER is $HOME
```

EOF

#We can also use grep command

```
cat << EOF | grep -i user
```

The user is \$USER

The home for this \$USER is \$HOME

EOF

```
sh-3.2# ./usage-of-here-doc.sh
```

The user is root

The home for this root is /var/root

The user is: root

The home for this user root is: /var/root

The user is root

--Here String usage

-Here string is like here document only but with one line

Syntax:

```
command <<<string
```

--e.g

By using here string we can easily convert the string to capital, no need to use echo command.

```
#tr [a-z] [A-Z] <<<"Welcome to my world"
```

WELCOME TO MY WORLD

--Writing comments for a shell script

**Single line comment

**Multi line comment

cat comment-example.sh

#!/bin/bash

#name :Asutosh

#Purpose :Inventory Script

echo "This is an inventory script"

bash comment-example.sh

This is an inventory script

**Multi line comment

at comment-example.sh

#!/bin/bash

#name :Asutosh

#Purpose :Inventory Script

echo "This is an inventory script"

#If I want to add multi line comment

<< MyComment

name :Asutosh

Purpose :Inventory Script

echo "This is an inventory script"

If I want to add multi line comment

Mycomment

sh-3.2# bash comment-example.sh

This is an inventory script

--What is #!/usr/bin/env bash

-What is Shebang line?

Shebang line means : which shell we are using to execute our shell scripts.

Suppose in our case it is bash shell and we are using shebang as #!/bin/bash

In case of other os it could be different path.

cat /etc/shells

/bin/sh

/bin/bash

/usr/bin/sh

/usr/bin/bash <=====

So it may not work on the other os.

-We can do that simply with env

which env

/usr/bin/env

path of env command is same on all the os

cat comment-example.sh

#!/usr/bin/env bash <=====

#name :Asutosh

#Purpose :Inventory Script

```
echo "This is an inventory script"
```

**Then env command will take care of the path of the bash. So on all the os we can run it.

*****We can make our shell scripts portable with--

```
#!/usr/bin/env bash
```

--Debugging a bash shell script

Debugging is determining the cause which fails the script.

Why script fails ?

Because of some errors.

This is because of two type of errors.

Syntax Errors

Runtime Errors

Syntax Errors stops script execution and run time errors don't stop script.

e.g -

```
cat debugging.sh
```

```
#!/usr/bin/env bash
```

```
pwd
```

```
date
```

```
yiff;fyfyig;iyug
```

```
user=$(whoami
```

```
echo "The user name is $user"
```

```
-----  
./debugging.sh  
/Users/ASUTOSH/COURSES/Linux-Shell-Automation  
Tue Oct 5 02:49:04 IST 2021  
./debugging.sh: line 4: yiff: command not found  
./debugging.sh: line 4: fyfyig: command not found  
./debugging.sh: line 4: iyug: command not found  
./debugging.sh: line 5: unexpected EOF while looking for matching `)`  
./debugging.sh: line 8: syntax error: unexpected end of file  
-----
```

Actually we don't have good debugging procedures with shell scripting, but we can try with some commands.

And there are different commands for debugging and we will work with set command.

One more thing our bash is an interpreter.

Interpreter will read the code line by line and execute the code line byu line

But compiler , compile the whole code and then exceute the code.

```
-----  
We can go with set command and We have different options with set command. Syntax:  
set [options]
```

No Options: To list system defined variables(set)

set -n No Execution, Purely for syntax check.


```
----  
e.g-  
cat debugging.sh  
#!/usr/bin/env bash  
set -n      <=====
```

pwd
date
yiff;fyfyig;iyug
user=\$(whoami
echo "The user name is \$user"

```
-----  
sh-3.2# ./debugging.sh  
./debugging.sh: line 6: unexpected EOF while looking for matching `)`  
./debugging.sh: line 9: syntax error: unexpected end of file
```

It will not execute the shell script, only check the syntax.

set -x Prints the command before executing it in script

```
----  
e.g-  
cat debugging.sh  
#!/usr/bin/env bash  
#set -n  
set -x  
pwd  
date  
yiff;fyfyig;iyug  
user=$(whoami  
echo "The user name is $user"
```

```
./debugging.sh
+ pwd
/Users/ASUTOSH/COURSES/Linux-Shell-Automation
+ date
Tue Oct 5 03:12:38 IST 2021
+ yiff
./debugging.sh: line 6: yiff: command not found
+ fyfyig
./debugging.sh: line 6: fyfyig: command not found
+ iyug
./debugging.sh: line 6: iyug: command not found
./debugging.sh: line 7: unexpected EOF while looking for matching `)'
./debugging.sh: line 10: syntax error: unexpected end of file
```

```
set -e Exit Script if any command fails
```

e.g-

```
cat debugging.sh
#!/usr/bin/env bash
#set -n
#set -x
set -e
pwd
date
yiff;fyfyig;iyug
user=$(whoami
echo "The user name is $user"
```

```
sh-3.2# ./debugging.sh
/Users/ASUTOSH/COURSES/Linux-Shell-Automation
Tue Oct 5 03:14:08 IST 2021
./debugging.sh: line 7: yiff: command not found
```

** We can also use debug like this-----

```
sh-3.2# bash -x debugging.sh <=====
+ bash -x debugging.sh
+ set -e
+ pwd
/Users/ASUTOSH/COURSES/Linux-Shell-Automation
+ date
Tue Oct 5 03:18:58 IST 2021
+ yiff
debugging.sh: line 7: yiff: command not found
```

--Exit status of a command(very important)

-Each linux command returns a status when it is executed.

- We will get (zero) if the command executed successfully. And we will get non zero value, if the command faced any error.

**echo \$?

e.g-

```
echo $?
+ echo 0
0
```

```
sh-3.2# ll
+ ll
sh: ll: command not found
```

```
sh-3.2# echo $?
```

```
+ echo 127
```

```
127
```

-We can store the value of (\$?) in to a variable.

```
sh-3.2# ll
```

```
+ ll
```

```
sh: ll: command not found
```

```
sh-3.2# command_rv=$?
```

```
+ command_rv=127
```

```
sh-3.2# echo $command_rv
```

```
+ echo 127
```

```
127
```

**Non zero values can be 1-255.

example-

127 - Command not found

1 - Command failed during execution

2 - Incorrect command usage.

Special bash parameters and their meaning

Special bash parameter Meaning

\$! **\$!** bash script parameter is used to reference the process ID of the most recently executed command in background.

\$\$ **\$\$** is used to reference the process ID of bash shell itself

\$# **\$#** is quite a special bash parameter and it expands to a number of positional parameters in decimal.

\$0 **\$0** bash parameter is used to reference the name of the shell or shell script. so you can use this if you want to print the name of shell script.

\$- **\$-** (dollar hyphen) bash parameter is used to get current option flags specified during the invocation, by the set built-in command or set by the bash shell itself. Though this bash parameter is rarely used.

\$? **\$?** is one of the most used bash parameters and used to get the exit status of the most recently executed command in the foreground. By using this you can check whether your bash script is completed successfully or not.

\$_ **\$_** (dollar underscore) is another special bash parameter and used to reference the absolute file name of the shell or bash script which is being executed as specified in the argument list. This bash parameter is also used to hold the name of mail file while checking emails.

\$@ **\$@** (dollar at the rate) bash parameter is used to expand into positional parameters starting from one. When expansion occurs inside double-quotes, every parameter expands into separate words.

\$* **\$*** (dollar star) this is similar to **\$@** special bash parameter only difference is when expansion occurs with double quotes, it expands to a single word with the value of each bash parameter separated by the first character of the IFS special environment variable.

--Basic string operations

Defining a string variable

```
x=shell / y="Shell scripting" / cmdOut=$(date)
```

Displaying the string variable value

```
echo $x / echo ${x}
```

Finding the length of a string

```
xLength=${#x}
```

Concatenation of strings

```
xyResult=$x$y
```

Convert Strings into lower/upper case

```
xU=${x^^}, yL=${y,,}
```

Replacing the part of the string using variable

```
newY=${y/Shell/Bash Shell} or we can also use sed command
```

Slicing the string/sub-string

```
${variable_name:start_position:length}
```

```
sh-3.2# x="shell"
```

```
sh-3.2# y="shell scripting"
```

```
sh-3.2# z=$(date)
```

```
sh-3.2# echo $z
```

```
Tue Oct 5 03:50:35 IST 2021
```

```
sh-3.2# echo $(date)
Tue Oct 5 03:51:20 IST 2021
```

****Length**

Print how many characters

```
z="bash scripting"
```

```
echo ${#z}
```

```
14
```

***Concatination**

```
z=$x$y
```

```
sh-3.2# echo $z
```

```
shellshell scripting
```

****Convert strings to upper/lower case**

```
x=shell
```

```
[root@m2-maprts-vm248-172 ~]# y="Shell scripting"
```

```
[root@m2-maprts-vm248-172 ~]# echo "${x^^}"
```

```
> "
```

```
SHELL
```

```
[root@m2-maprts-vm248-172 ~]# echo "${x^^}"
```

```
SHELL
```

```
[root@m2-maprts-vm248-172 ~]# xU=${x^^}
```

```
[root@m2-maprts-vm248-172 ~]# echo $xU
```

```
SHELL
```

```
[root@m2-maprts-vm248-172 ~]# echo "${y,,}"
```

```
> "
```

shell scripting

```
[root@m2-maprts-vm248-172 ~]# echo "${y,,}"
```

shell scripting

****We can also do this with translate.**

```
echo "$x" | tr [a-z] [A-Z]
```

SHELL

```
[root@m2-maprts-vm248-172 ~]# echo "$y" | tr [a-z] [A-Z]
```

SHELL SCRIPTING

```
echo "$x" | tr [a-z] [A-Z]
```

SHELL

```
[root@m2-maprts-vm248-172 ~]# echo "$y" | tr [a-z] [A-Z]
```

SHELL SCRIPTING

=====

*******awk,cut and tr*******

```
[root@m2-maprts-vm248-172 ~]# httpd -v | awk -F " " '/version/ {print $3}'
```

Apache/2.4.6

```
[root@m2-maprts-vm248-172 ~]# httpd -v | awk -F " " '/version/ {print $3}' | cut -d "/" -f 2
```

2.4.6

```
[root@m2-maprts-vm248-172 ~]# httpd -v | awk -F "^C"/version/ {print $3}'
```

```
[root@m2-maprts-vm248-172 ~]# docker -v
```

Docker version 1.13.1, build 7d71120/1.13.1

```
[root@m2-maprts-vm248-172 ~]# docker -v |awk '{print $3}'
```

1.13.1,

```
[root@m2-maprts-vm248-172 ~]# docker -v |awk '{print $3}'| tr -d [,]
```

1.13.1

```
[root@m2-maprts-vm248-172 ~]# docker -v |awk '{print $3}'| cut -d ","
```

cut: you must specify a list of bytes, characters, or fields

Try 'cut --help' for more information.


```
[root@m2-maprts-vm248-172 ~]# docker -v |awk '{print $3}'| cut -d "," -f 1  
1.13.1
```

****Replacing the part of the string using variable**

```
echo $y  
Shell scripting
```

```
echo "${y/Shell/bash shell}"  
bash shell scripting
```

****We can also use sed command**

```
echo "$y" | sed 's/Shell/BAsh Shell/'  
BAsh Shell scripting
```

--String operations on paths|Useful for real-time

realpath : Converts each filename argument to an absolute pathname but it do not validate the path.

basename:
Strips directory information
Strips suffixes from file names

dirname : It will delete any suffix beginning with the last slash character and return the result

realpath variables.tf

/root/terraform/variables.tf

--Input and Output command for bash shell scripting

```
#!/usr/bin/env bash
my_name="Asutosh"
#I want to convert the string into upper case and print that string
Name_Upper=$(echo "$my_name" | tr [a-z] [A-Z])
echo "The Name in Upper case is: $Name_Upper" #echo is called the output command for the
shell script
```

sh-3.2# ./input-output.sh
The Name in Upper case is: ASUTOSH

Now I want to give the name as an input to the script.

```
#!/usr/bin/env bash
#my_name="Asutosh"
read -p "Enter the name: " my_name
#If we will not provide any variable to the read command, then the value will be automatically
stored in "REPLY"
#I want to convert the string into upper case and print that string
Name_Upper=$(echo "$my_name" | tr [a-z] [A-Z])
echo "The Name in Upper case is: $Name_Upper" #echo is called the output command for the
shell script
```

sh-3.2# ./input-output.sh
Enter the name: ankita

The Name in Upper case is: ANKITA

If we will not provide any variable to the read command, then the value will be automatically stored in "REPLY"

```
read -p "enter the name"
```

```
enter the name asutosh
```

```
sh-3.2# echo $REPLY
```

```
asutosh
```

--Read with command line arguments

```
echo $0 <=====This is the script name itself.
```

```
echo $1
```

```
echo $2
```

```
echo $3
```

```
./input-output.sh 12 13 14
```

```
12
```

```
13
```

```
14
```

```
./input-output.sh 12 13 14
```

```
./input-output.sh
```

```
12
```

```
13
```

```
14
```

---If we want to print the command line argument for the two digit number, then take curly braces.

```
echo $0
echo $1
echo $2
echo $3
echo $10 #It will consider as $1 and 0
echo ${12}
```

```
-----
sh-3.2# ./input-output.sh 1 2 3 4 5 6 7 8 9 0 12 13 12 1 3 1 3 1 3 1 3 1 3
./input-output.sh
1
2
3
10
13
-----
```

**IF I want to see how many command line arguments are getting passed---

```
echo $0
echo $1
echo $2
echo $3
echo $10 #It will consider as $1 and 0
echo ${12}
```

```
echo "The command line arguments are getting passed $#"
```

```
-----
```

```
sh-3.2# ./input-output.sh 1 2 3 4 5 6 7 8 9 0 12 13 12 1 3 1 3 1 3 1 3
```

```
./input-output.sh
```

```
1
```

```
2
```

```
3
```

```
10
```

```
13
```

```
The command line arguments are getting passed 23
```

```
sh-3.2# ./input-output.sh
```

```
./input-output.sh
```

```
0
```

```
The command line arguments are getting passed 0
```

```
sh-3.2# ./input-output.sh 1 2
```

```
./input-output.sh
```

```
1
```

```
2
```

```
10
```

```
The command line arguments are getting passed 2
```

```
-----
```

```
**We want to see all the passing arguments
```

```
#!/usr/bin/env bash
```

```
#To comment multiple lines at a time.
```

```
<<mycode
```

```
#my_name="Asutosh"
```

```
read -p "Enter the name: " my_name
#If we will not provide any variable to the read command, then the value will be automatically
stored in "REPLY"
#I want to convert the string into upper case and print that string
Name_Upper=$(echo "$my_name" | tr [a-z] [A-Z])
echo "The Name in Upper case is: $Name_Upper" #echo is called the output command for the
shell script
mycode
```

```
echo $0
echo $1
echo $2
echo $3
echo $10 #It will consider as $1 and 0
echo ${12}
```

```
echo "The command line arguments are getting passed $#"
```

```
echo "All the command line arguments are $@"
```

```
#or
echo "All the command line arguments are $*"
```

```
sh-3.2# ./input-output.sh 1 2
```

```
./input-output.sh
```

```
1
```

```
2
```

```
10
```

```
The command line arguments are getting passed 2
```

All the command line arguments are 1 2

All the command line arguments are 1 2

sh-3.2# ./input-output.sh 1 2 3 4 5 6 7 8 9 0 12 13 12 1 3 1 3 1 3 1 3 1 3

./input-output.sh

1

2

3

10

13

The command line arguments are getting passed 23

All the command line arguments are 1 2 3 4 5 6 7 8 9 0 12 13 12 1 3 1 3 1 3 1 3 1 3

All the command line arguments are 1 2 3 4 5 6 7 8 9 0 12 13 12 1 3 1 3 1 3 1 3 1 3

--Arithmetic Operators for Bash Shell Scripting

-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)

e.g--

sh-3.2# x=89

```
sh-3.2# y=96
sh-3.2# MUL=$((x*y))
sh: syntax error near unexpected token `('
sh-3.2# ((mul=x*y))
sh-3.2# echo "$mul"
8544
sh-3.2# ((sub=x-y))
sh-3.2# echo "$sub"
-7
```

We can also use below commands to directly print the result--

```
sh-3.2# echo $((x*y))
8544
sh-3.2# echo $((x/y))
0
sh-3.2# echo $((x-y))
-7
sh-3.2# echo $((x+y))
185
```

*Using bc (For integer and float numbers)

We can use bash calculator(bc) to do the arithmetic operations for float and integer numbers.

e.g---

```
sh-3.2# x=236
```



```
sh-3.2# y=678
sh-3.2# bc<<<$x/$y
0
sh-3.2# bc<<<$y/$x
2
sh-3.2# bc<<<"$y/$x"
2
sh-3.2# bc<<<"$x*$y"
160008
sh-3.2# bc<<<"$x+$y"
914
```

--Simple case statement

```
--Syntax:  case $opt in
            opt1)
                statements
                ;;
            opt2)
                statements
                ;;
            *)
                statements
                ;;
        esac
```

For example I want to design a simple calculator--

```
#!/usr/bin/env bash
```

```
read -p "Enter the number num1: " a
read -p "Enter the number num2: " b
read -p "Enter the option(1.Addition,2.Subtraction,3.Multiplication,4.Division) " opt
case $opt in
    1)
        echo "You have selected addition"
        echo "The addition of $a and $b is $((a+b))"
        ;;
    2)
        echo "You have selected Subtraction"
        echo "The Subtraction of $a and $b is $((a-b))"
        ;;
    3)
        echo "You have selected Multiplication"
        echo "The multiplication of $a and $b is $((a*b))"
        ;;
    4)
        echo "Yopu have selected division"
        echo "The division of $a and $b is $((a/b))"
        ;;
    *)
        echo "You have entered the wrong option"
        ;;
esac
```

```
sh-3.2# ./calculator-switch.sh
Enter the number num1: 980
Enter the number num2: 890
```

Enter the option(1.Addition,2.Subtraction,3.Multiplication,4.Division) 3

You have selected Multiplication

The multiplication of 980 and 890 is 872200

--test command, commands changing and conditional statements

--Test command and it's usage| comparision and file test operators

-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 false if they are not equal else true

`[[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 non-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

For example--

```
sh-3.2# test 4 eq 43
sh: test: eq: binary operator expected
sh-3.2# test 4 -eq 43
sh-3.2# echo $?
1
sh-3.2# [ 4 eq 45 ]
sh: [: eq: binary operator expected
sh-3.2# [ 4 -eq 45 ]
sh-3.2# echo $?
1
sh-3.2# [[ 4 -eq 45 ]]
sh-3.2# echo $?
1
```

```
sh-3.2# x=89
sh-3.2# y=90
sh-3.2# [[ $x -gt $y ]]
sh-3.2# echo $?
1
sh-3.2# [[ $x -lt $y ]]
sh-3.2# echo $?
0 <===== Zero means success
```

--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:

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

cmd1 && cmd2 || cmd3 – Run cmd2 if cmd1 is success else run cmd3

For example----

which docker;docker -v

/usr/bin/docker

Docker version 1.13.1, build 7d71120/1.13.1

which docker && docker -v

/usr/bin/docker

Docker version 1.13.1, build 7d71120/1.13.1

lll && docker -v
-bash: lll: command not found

lll || docker -v
-bash: lll: command not found
Docker version 1.13.1, build 7d71120/1.13.1

ll || docker -v
total 20
-rw-----. 1 root root 1911 Jul 6 2020 anaconda-ks.cfg
-rwxr-xr-x 1 root root 0 Aug 15 16:08 cats.txt
-rw-r--r--. 1 root root 9 Jul 6 2020 disk.txt
-rwxr-xr-x 1 root root 243 Sep 28 15:40 docker-status.sh
-rwxr-xr-x 1 root root 0 Aug 15 16:08 dogs.txt
drwxr-xr-x 2 root root 6 Sep 14 07:21 jenkins
drwxr-xr-x 2 root root 6 Jul 3 10:10 linux-shell-script
drwxr-xr-x 2 root root 31 Aug 31 14:47 python-scripts
-rw-r--r-- 1 root root 423 Jun 22 07:49 swapuse.sh
drwxr-xr-x 5 root root 294 Aug 15 16:07 terraform
-rw-r--r-- 1 root root 151 Sep 17 14:07 test.out
-rw-r--r-- 1 root root 0 Aug 31 14:48 test.txt

ll && docker -v || lll
total 20
-rw-----. 1 root root 1911 Jul 6 2020 anaconda-ks.cfg
-rwxr-xr-x 1 root root 0 Aug 15 16:08 cats.txt

```
-rw-r--r--. 1 root root  9 Jul 6 2020 disk.txt
-rwxr-xr-x 1 root root 243 Sep 28 15:40 docker-status.sh
-rwxr-xr-x 1 root root  0 Aug 15 16:08 dogs.txt
drwxr-xr-x 2 root root  6 Sep 14 07:21 jenkins
drwxr-xr-x 2 root root  6 Jul 3 10:10 linux-shell-script
drwxr-xr-x 2 root root 31 Aug 31 14:47 python-scripts
-rw-r--r-- 1 root root 423 Jun 22 07:49 swapuse.sh
drwxr-xr-x 5 root root 294 Aug 15 16:07 terraform
-rw-r--r-- 1 root root 151 Sep 17 14:07 test.out
-rw-r--r-- 1 root root  0 Aug 31 14:48 test.txt
Docker version 1.13.1, build 7d71120/1.13.1
```

```
-----
lll && docker -v || ll
-bash: lll: command not found
total 20
```

```
-rw-----. 1 root root 1911 Jul 6 2020 anaconda-ks.cfg
-rwxr-xr-x 1 root root  0 Aug 15 16:08 cats.txt
-rw-r--r--. 1 root root  9 Jul 6 2020 disk.txt
-rwxr-xr-x 1 root root 243 Sep 28 15:40 docker-status.sh
-rwxr-xr-x 1 root root  0 Aug 15 16:08 dogs.txt
drwxr-xr-x 2 root root  6 Sep 14 07:21 jenkins
drwxr-xr-x 2 root root  6 Jul 3 10:10 linux-shell-script
drwxr-xr-x 2 root root 31 Aug 31 14:47 python-scripts
-rw-r--r-- 1 root root 423 Jun 22 07:49 swapuse.sh
drwxr-xr-x 5 root root 294 Aug 15 16:07 terraform
-rw-r--r-- 1 root root 151 Sep 17 14:07 test.out
-rw-r--r-- 1 root root  0 Aug 31 14:48 test.txt
```

```
-----
--Executing block of code using {}
```

```
-----
- If we are putting different commands inside a curly bracket, we can call them a block of code.
```


e.g-

```
cat block.sh
#!/usr/bin/env bash
```

```
{
pwd
ls
date
}
```

-If all the commands are executing independently then I can write them with semicolon

```
cat block.sh
#!/usr/bin/env bash
```

```
{ ls; pwd; date; }
```

**Now I want to write the code, if the first command will run then only the second command/code block will execute

```
[root@m2-maprts-vm248-172 ~]# cat block.sh
#!/usr/bin/env bash
```

```
# { ls; pwd; date; }
```

#Now I want to write the code, if the first command will run then only the second command/code block will execute

```
which docker && { echo "Docker is installed on this node" ; echo "The docker version is  
$(docker -v)" ; }
```

```
[root@m2-maprts-vm248-172 ~]# ./block.sh
```

```
/usr/bin/docker
```

```
Docker is installed on this node
```

```
The docker version is Docker version 1.13.1, build 7d71120/1.13.1
```

```
--Conditional Statements | Simple if | if else | if elif elif else
```

```
--simple if and if-else conditional statement
```

Syntax:

```
Cmd1 && Cmd2
```

```
if Cmd1
```

```
then
```

```
    Cmd2
```

```
fi
```

```
Cmd1&&{Cmd2; Cmd3;}
```

```
if Cmd1
```

```
then
```

```
    Cmd2
```

```
    Cmd3
```

```
fi
```

Simple if-else Stement

Syntax:

Cmd1 && Cmd2 || Cmd3

```
if Cmd1
then
  Cmd2
else
  cmd3
fi
```

```
Cmd1 && { Cmd2 ; Cmd3 ; } || Cmd4
if Cmd1
then
  Cmd2
  Cmd3
else
  Cmd4
fi
```

-For example--

```
cat if-condition.sh
#!/usr/bin/env bash
```

```
if which docker
then
  echo "Docker is installed on this node"
  echo "Docker version installed $(docker -v)"
```

fi

./if-condition.sh

/usr/bin/docker

Docker is installed on this node

Docker version installed Docker version 1.13.1, build 7d71120/1.13.1

--I want to nullify the result of the command.

cat if-condition.sh

#!/usr/bin/env bash

#I want to nullify the output of the command (which docker)

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

then

echo "Docker is installed on this node"

echo "Docker version installed \$(docker -v)"

fi

./if-condition.sh

Docker is installed on this node

Docker version installed Docker version 1.13.1, build 7d71120/1.13.1

***Note - If we are taking/writing comparison operators after the if statement, we need to put two [[]], if it is a command, do not take brackets.

cat if-condition.sh

```
#!/usr/bin/env bash
<<mycode
if which docker 2> /dev/null 1> /dev/null
then
    echo "Docker is installed on this host"
    echo "Docker version is $(docker -v)"
fi
mycode
```

```
which docker 1> /dev/null
```

```
if [[ $? -eq 0 ]]
then
    echo "Docker is present on the host"
    echo "Docker version is $(docker -v)"
else
    echo "Docker is not present on the host"
fi
```

```
[root@m2-maprts-vm248-172 ~]# ./if-condition.sh
Docker is present on the host
Docker version is Docker version 1.13.1, build 7d71120/1.13.1
```

--Simple Shell Script to verify the user is root or not and another script User is having sudo

Here we can use "whoami" to identify the user, but id is the best command to identify the user

```
[root@m2-maprts-vm248-172 ~]# whoami
root
```

```
[root@m2-maprts-vm248-172 ~]# id
```

```
uid=0(root) gid=0(root) groups=0(root)
```

```
[root@m2-maprts-vm248-172 ~]# id -u  
0
```

```
[root@m2-maprts-vm248-172 ~]# id -un  
root
```

```
-----
```

```
cat root-or-not.sh
```

```
#!/usr/bin/env bash
```

```
#Simple Shell Script to verify the user is root or not and
```

```
userid=$(id -u)
```

```
if [[ $userid -eq 0 ]]
```

```
then
```

```
    echo "You are root"
```

```
else
```

```
    echo "You are not root"
```

```
    echo "The user name is $(id -un)"
```

```
fi
```

```
-----
```

```
./root-or-not.sh
```

```
You are root
```

```
-----
```

```
--Lets check if the user is having sudo permission or not
```

```
-----
```

```
sudo -v
```

```
#!/usr/bin/env bash

sudo -v 2> /dev/null 1> /dev/null
if [[ $? -eq 0 ]]
then
    echo "The user $(id -un) is having sudo privileges on this host $(hostname)"
else
    echo "The user $(id -un) is not having sudo privileges on this host $(hostname)"
fi
```

--Shell script to start Docker service

```
[root@m2-maprts-vm248-172 ~]# cat start-docker.sh
#!/usr/bin/env bash
```

```
systemctl status docker 2> /dev/null 1> /dev/null
if [[ $? -eq 0 ]]
then
    echo "Docker is already running"
else
    echo "Starting Docker....."
    systemctl start docker 1> /dev/null 2> /dev/null
    sleep 5
    if [[ $? -eq 0 ]]
    then
        echo "Docker started successfully"
    else
        echo "Docker service failed to start"
    fi
fi
```

```
[root@m2-maprts-vm248-172 ~]# ./start-docker.sh
```

Docker is already running

```
[root@m2-maprts-vm248-172 ~]# systemctl stop docker
```

```
[root@m2-maprts-vm248-172 ~]# ./start-docker.sh
```

Docker started successfully

--if-elif-else Conditional statement

```
#!/usr/bin/env bash
```

```
#####
```

```
# Author: Asutosh #
```

```
# Date: Oct-2021 #
```

```
# Usage is: start,stop,restart and version of docker #
```

```
#####
```

```
read -p "Enter your option" option
```

```
if [[ $option == start ]]
```

```
then
```

```
    echo "Starting docker...."
```

```
    systemctl start docker 1> /dev/null 2> /dev/null
```

```
    sleep 5
```

```
    echo "Docker started successfully"
```

```
    sleep 3
```

```
    echo "Getting status....."
```

```
    sleep 2
```

```
    systemctl status docker
```

```
elif [[ $option == stop ]]
```

```
then
```

```
    echo "Stopping Docker...."
```



```

systemctl stop docker 1> /dev/null 2> /dev/null
sleep 5
echo "Docker stopped"
sleep 3
echo "Getting status....."
sleep 2
systemctl status docker
elif [[ $option == restart ]]
then
echo "Restarting Docker....."
systemctl restart docker 1> /dev/null 2> /dev/null
sleep 5
echo "Docker restarted successfully"
sleep 3
echo "Getting status....."
sleep 2
systemctl status docker
elif [[ $option == version ]]
then
echo "Getting Docker version....."
docker -v
else
echo "Enter valid option"
fi
-----
./actions-on-docker.sh
Enter your option start
Starting docker.....
Docker started successfully
Getting status.....
• docker.service - Docker Application Container Engine

```

Loaded: loaded (/usr/lib/systemd/system/docker.service; disabled; vendor preset: disabled)

Active: active (running) since Thu 2021-10-07 18:01:14 PDT; 1min 41s ago

--How to handle command line arguments

- They are two ways to provide inputs for a shell script.

 **Using read command

 **Using command line arguments

Lets check with command line arguments--

**I want to see the status of any service with the command line argument

#!/usr/bin/env bash

read -p "Enter the service name to perform action: " servicename

read -p "Enter the action name to perform on the \$servicename: " actionname

systemctl \${actionname} \${servicename}

--Shell script to send Automatic Mail Alert when RAM memory gets LOW

-For this first we need to calculate the total value of RAM.

[root@m2-maprts-vm248-172 ~]# free -g

	total	used	free	shared	buff/cache	available
Mem:	19	10	2	0	6	8

```
Swap:      4      0      4
[root@m2-maprts-vm248-172 ~]# free -gt
      total    used    free   shared  buff/cache   available
Mem:      19      10      2       0        6        8
Swap:      4       0       4
Total:     24      10       7
```

```
[root@m2-maprts-vm248-172 ~]# free -mt
      total    used    free   shared  buff/cache   available
Mem:   19914   10910    2289    146    6714    8499
Swap:   5055      0    5055
Total:  24970   10910    7345
```

----Then we can find out the only total value of the RAM, so that we can compare with the threshold value

```
[root@m2-maprts-vm248-172 ~]# free -mt | grep -w Total
Total:   24970   10870    7384
```

```
[root@m2-maprts-vm248-172 ~]# free -mt | grep -w Total | awk '{print $4}'
7357
```

```
-----
[root@m2-maprts-vm248-172 ~]# cat low-ram-alert.sh
#!/usr/bin/env bash
TH_L=9000
free_RAM=$(free -mt | grep -w Total | awk '{print $4}')
if [[ $free_RAM -le $TH_L ]]
then
    echo "The server is running low with available RAM size, Current available RAM is $free_RAM"
fi
```

Now If I want to send a mail alert to a given mail id

```
[root@m2-maprts-vm248-172 ~]# cat ./low-ram-alert.sh
#!/usr/bin/env bash
To=asutoshgec@gmail.com
TH_L=9000
free_RAM=$(free -mt | grep -w Total | awk '{print $4}')
if [[ $free_RAM -le $TH_L ]]
then
    echo "The server is running low with available RAM size, Current available RAM is
$free_RAM" | mail -s "LOW RAM ALERT $(date)" $To
fi
```

If we need to monitor this continuously, we can schedule this with crontab

--Shell script to monitor file system utilization with mail alert

-We can see the filesystem utilization with the df command

```
[root@m2-maprts-vm248-172 ~]# df -h
```

Filesystem	Size	Used	Avail	Use%	Mounted on
devtmpfs	9.8G	0	9.8G	0%	/dev
tmpfs	9.8G	0	9.8G	0%	/dev/shm
tmpfs	9.8G	57M	9.7G	1%	/run
tmpfs	9.8G	0	9.8G	0%	/sys/fs/cgroup
/dev/mapper/centos_m2--maprts--vm40--172-root	94G	15G	80G	16%	/
/dev/sda1	1014M	193M	822M	20%	/boot
localhost:/mapr	444G	77G	367G	18%	/mapr
tmpfs	2.0G	0	2.0G	0%	/run/user/0

-We can use grep -Ev to bypass tmpfs and devtmpfs filesystem outputs.

```
[root@m2-maprts-vm248-172 ~]# df -h | grep -Ev "tmpfs|devtmpfs"
Filesystem                Size  Used Avail Use% Mounted on
/dev/mapper/centos_m2--maprts--vm40--172-root 94G  15G  80G  16% /
/dev/sda1                  1014M  193M  822M  20% /boot
localhost:/mapr            444G   77G  367G  18% /mapr
```

-Now we need to send the output to a mail address with cron job

```
#Now we need to send filesystem utilization mail
Mailid=asutoshgec@gmail.com
FS_util=$(df -h | egrep -v "tmpfs|devtmpfs")
echo -e "The filesystem utilization on $(hostname) is: \n $FS_util " | /usr/bin/mail -s
"Filesystem utilization" "$Mailid"
#To execute the echo command with \n. need to put -e.
```


--Arrays of Bash Shell Scripting

- Concepts of Arrays

What is an Array ?

How to define array ?

How to access Array Values ?

Different Types of Arrays

How to store the command output into an array ?

How to delete and update exiting array with new values ? How to read array using read command ?

What is an Array and How to define or declare it ?

****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.

Empty Array: myArray=()

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

-6 -5 -4 -3 -2 -1

Positive Index Values or Positive Indices

Negative Index Values or Negative Indices

```
-----
echo "$myarray"
echo "${myarray}"
echo "${myarray[*]}"
echo "${myarray[@]}"
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 "${myarray[*]:0:2}" <===== Prints two values starting from index-0
echo "${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 scripting"
Or
newarray=([5]="bash" [9]="shell scripting")
```

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

We have two types of arrays in Bash Shell Scripting.

They are:

Index Based Arrays or Arrays

Associative Arrays.

-We can declare an array like this--

```
myname=(1 2 3 4 5 6 7 8 bash pwd ls)
```

If we will run below

```
echo "${myname}" <===I will get only one value
```

If I need to see all the values, then I need to run

```
echo "${myname[*]}" or "${myname[@]}"
```

```
[root@m2-maprts-vm248-172 ~]# echo ${myname[*]}  
1 2 3 4 5 6 7 8 bash pwd ls
```

```
echo ${myname[8]}
```

```
bash
```

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
```

--Loops and Loop control statements

**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

```
cat for-loop.sh
```

```
#!/usr/bin/env bash
```

```
for i in {1..6}
```

```
do
```

```
    echo "Welcome"
```

```
done
```

```
sh-3.2# ./for-loop.sh
```

```
Welcome
```

```
Welcome
```

```
Welcome
```

```
Welcome
```

```
Welcome
```

```
Welcome
```

-Lets check if the files are having execution permission or not.

```
for i in usage-of-here-doc.sh variable-practice.sh
```

```
do
```

```
if [[ -x $i ]]
```

```
then
```

```
    echo "$i is having execution permission"
```

```
else
```

```
    echo "$i is not having execution permission "
```

```
fi
```

```
done
```

usage-of-here-doc.sh is having execution permission
variable-practice.sh is not having execution permission

-----If I want to look for all the files in the current directory

```
#for i in usage-of-here-doc.sh variable-practice.sh
for i in $(ls)
do
if [[ -x $i ]]
then
echo "$i is having execution permission"
else
echo "$i is not having execution permission "
fi
done
```

sudo-user-or-not.sh is not having execution permission
usage-of-here-doc.sh is having execution permission
variable-practice.sh is not having execution permission
while+loop+with+IFS.pdf is not having execution permission
xyz.txt is not having execution permission

--If I want to check the permissions for the given path

```
#for i in usage-of-here-doc.sh variable-practice.sh
#for i in $(ls) #It will check all the files inside the current directory
#I want to check for all the files inside the directory of a given path
given_path=$1
for i in $(ls $given_path)
```

```
do
  if [[ -x $i ]]
  then
    echo "$i is having execution permission"
  else
    echo "$i is not having execution permission "
  fi
done
```

```
-----
ip.txt is not having execution permission
kubernetes is not having execution permission
my-workspace.code-workspace is not having execution permission
python-scripting is not having execution permission
python-scripting-automation is not having execution permission
```

```
-----
--Different types of for loop syntax's
```

```
-----
```

Different ways to use for loop:

****Basic for loop:**

```
for variable in list_of_values
do
  command1
  command2
done
```

****Infinity for loop**

```
for ((;;))
do
```

```
command1
command2
done
```

--Installing multiple packages with for loop and command line arguments

```
#!/usr/bin/env bash
user=$(id -un)
if [[ $user == root ]]
then
    echo "Performing Installation....."
else
    echo "Please switch to the root user before performing YUM installation"
    exit 1
fi

for i in vim nginx ftp
do
    if which $i 1> /dev/null
    then
        echo "$i package is already installed"
    else
        echo "Installing Package....."
        yum install -y $i 1> /dev/null #&> /dev/null
        if [[ $? -eq 0 ]]
        then
            echo "SUCCESS"
        else
            echo " FAILED"
        fi
    fi
done
```

fi
done

[root@m2-maprts-vm248-172 ~]# ./installation-with-for.sh
Performing Installation.....
vim package is already installed
nginx package is already installed
ftp package is already installed

--Difference between \$@ and \$*

#!/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

./difference-\"\$\"*-\$\"@.sh 1 2 3 4 5 6 7 8 9
The below output is for \$*
1 2 3 4 5 6 7 8 9
The below output is for \$@
1

2
3
4
5
6
7
8
9

--Basic syntaxes of while loop

Different ways to use while loop:

Infinity while loop:

```
while true
do
  command
done
```

```
while:
do
  commands
done
```

while loop with command

```
while command
do
  command/statements
done
```

```
while [[ 3 -gt 5 ]]
do
    statements
done
```

Different ways to use while loop:

Reading a file content:

```
while read line
do
    statements
done
```

Reading command output

```
command | while read line
do
    statements
done
```

example---

```
#!/usr/bin/env bash
<< myloop
while true
do
    echo "SUCCESS"
done
```



```
#This will run for infinity times
myloop
```

```
start=1
while [[ $start -le 10 ]]
do
    echo "SUCCESS"
    ((start++))
done
```

```
-----
sh-3.2# bash while-loop.sh
SUCCESS
SUCCESS
SUCCESS
SUCCESS
SUCCESS
SUCCESS
SUCCESS
SUCCESS
SUCCESS
SUCCESS
SUCCESS
```

```
-----
---Example read each line
```

```
file_name="define_and_calling_a_function.txt"
while read each_line
do
    echo "$each_line"
done < $file_name
```

```
sh-3.2# bash while-loop.sh
```

```
#!/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"
}
```

-If want to read any command output.

```
ls -l | while read each_line
do
echo "$each_line"
done
```

```
bash while-loop.sh
```

```
total 7928
```

```
-rw-r--r--@ 1 ASUTOSH admin 12292 Oct 12 05:17 .DS_Store
```

```
-rw-r--r--@ 1 ASUTOSH admin 73347 Sep 18 03:01 1.+Basic+String+Operations.pdf
```

```
-rw-r--r--@ 1 ASUTOSH admin 456156 Oct 12 03:49 1.+Introduction+to+Arrays.pdf
```

```
-rw-r--r--@ 1 ASUTOSH admin 198828 Oct 12 05:13 1.+Introduction+to+loops.pdf
```

--Functions

Simple Introduction to Functions

- 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:

```
function function_name
```

```
{  
  commands/statements  
}
```

```
function_name()  
{  
  commands/statements  
}
```

---For example--

Here we can create a function and we can call them anytime in the script.

```
#!/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
```

--Define a function and calling a function

```
#!/usr/bin/env bash
```

```
read_inputs()  
{  
    read -p "Enter the first number" num1  
    read -p "Enter the second number" num2  
}
```

```
addition()  
{  
    add=$((num1+num2))  
    echo "The addition of two numbers is: $add"  
}
```

```
subtraction()  
{  
    sub=$((num1-num2))  
    echo "Subtraction is: $sub"  
}
```

#Then we need to call the functions.First we need to call the read_input function

```
read_inputs
```

```
addition
```

subtraction

```
sh-3.2# ./define-function.sh
Enter the first number 67
Enter the second number 89
The addition of two numbers is: 156
Subtraction is: -22
```

--Passing parameters to a function

```
#!/usr/bin/env bash

addition()
{
    m=$1
    n=$2
    result=$((m+n))
    echo "Addition of $m and $n is : $result"
}

x=7
y=8
addition $x $y

p=9
q=10
addition $p $q

addition 4 9
```

sh-3.2# ./function-passing-arguments.sh

Addition of 7 and 8 is : 15

Addition of 9 and 10 is : 19

Addition of 4 and 9 is : 13

--Design simple digital clock

-We need to show the time after every 1 sec

```
#!/usr/bin/env bash
```

```
clear
```

```
while true
```

```
do
```

```
    date | awk '{print $4}'
```

```
    sleep 1
```

```
    clear
```

```
done
```
