



Topic-42: Process Management

We can perform process management by using ps command.

ps → Means Process Status

1) Listing Processes:

1) \$ ps

If we are not passing any argument, then it shows processes related to current session.

```
durgasoft@durgasoft:~$ ps
PID TTY      TIME CMD
20881 pts/0    00:00:00 bash
20890 pts/0    00:00:00 ps
```

2) \$ ps -e

-e means Everything. It shows all processes.

3) \$ ps -ef

It shows full listing of all processes
-f means Full Listing

```
durgasoft@durgasoft:~$ ps -ef
UID    PID  PPID  C STIME TTY      TIME CMD
root     1    0  0 19:41 ?        00:00:02 /sbin/init splash
root     2    0  0 19:41 ?        00:00:00 [kthreadd]
root     3    2  0 19:41 ?        00:00:00 [rcu_gp]
```

UID → UserId/ User Name

PID → Process ID (Every Process has Unique ID)

PPID → Parent Process ID

2) Filtering Processes:

We can filter processes based on UID, PID and PPID.

1) \$ ps -f -u durgasoft

It shows all processes which are running related user durgasoft

2) \$ ps -f -u root

It shows all processes which are running related to root user



3) \$ ps -f -p 20906

It shows process information related to 20906 process id.

4) \$ ps -f --ppid 1266

It shows all process information related to parent process id 1266.

Commonly used other ps Commands:

1) \$ ps -eH

Display a process tree.

```
1 ?      00:00:02  systemd
288 ?    00:00:00  systemd-journal
345 ?    00:00:00  systemd-udev
408 ?    00:00:01  systemd-resolve
559 ?    00:00:00  rsyslogd
560 ?    00:00:00  udisksd
564 ?    00:00:00  cupsd
692 ?    00:00:00  dbus
693 ?    00:00:00  dbus
569 ?    00:00:00  cron
570 ?    00:00:02  dbus-daemon
614 ?    00:00:01  NetworkManager
616 ?    00:00:00  ModemManager
```

2) \$ ps -e --forest

Display proces tree

```
994 ?      00:00:00  VBoxClient
995 ?      00:00:00  \_ VBoxClient
1010 ?     00:00:00  gdm3
1027 ?     00:00:00  \_ gdm-session-wor
1051 tty1   00:00:00  |   \_ gdm-wayland-ses
1055 tty1   00:00:00  |       \_ gnome-session-b
1061 tty1   00:00:13  |           \_ gnome-shell
1078 tty1   00:00:00  |               \_ Xwayland
1107 tty1   00:00:00  |                   \_ ibus-daemon
1110 tty1   00:00:00  |                       \_ ibus-dconf
1230 tty1   00:00:00  |                           \_ ibus-engine-sim
1133 tty1   00:00:00  |                               \_ gsd-xsettings
1138 tty1   00:00:00  |                                   \_ gsd-a11y-settin
1143 tty1   00:00:00  |                                       \_ gsd-clipboard
1146 tty1   00:00:00  |                                           \_ gsd-color
1151 tty1   00:00:00  |                                               \_ gsd-datetime
```



3) \$ pstree

Display processes in a tree format

```
durgasoft@durgasoft:~$ pstree
systemd--ModemManager--2*[{ModemManager}]
      --NetworkManager--dhclient
                        --2*[{NetworkManager}]
      --2*[VBoxClient--VBoxClient]
      --2*[VBoxClient--VBoxClient--{VBoxClient}]
      --VBoxClient--VBoxClient--2*[{VBoxClient}]
      --VBoxService--7*[{VBoxService}]
      --accounts-daemon--2*[{accounts-daemon}]
      --acpid
      --avahi-daemon--avahi-daemon
      --boltd--2*[{boltd}]
      --colord--2*[{colord}]
      --cron
      --cups-browsed--2*[{cups-browsed}]
      --cupsd--2*[dbus]
```

4) \$ top

We can use top command to see processes with consumed resources like memory and cpu utilization. use 'q' to come out of top command.

top - 21:41:31 up 2:00, 2 users, load average: 0.06, 0.08, 0.08

Tasks: 239 total, 1 running, 201 sleeping, 0 stopped, 0 zombie

%Cpu(s): 1.9 us, 0.7 sy, 0.0 ni, 97.5 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st

KiB Mem : 4169364 total, 1617212 free, 1336860 used, 1215292 buff/cache

KiB Swap: 483800 total, 483800 free, 0 used. 2532032 avail Mem

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1477	durgaso+	20	0	3479664	289172	118692	S	3.0	6.9	3:13.07	gnome-shell
1286	durgaso+	20	0	609356	142852	79532	S	1.7	3.4	0:50.24	Xorg
20871	durgaso+	20	0	804408	38456	28308	S	1.0	0.9	0:14.59	gnome-terminal-
21385	durgaso+	20	0	48996	3872	3152	R	1.0	0.1	0:01.51	top
1591	durgaso+	20	0	504684	23460	18144	S	0.7	0.6	0:00.36	gsd-xsettings
644	root	20	0	1165108	24428	13884	S	0.3	0.6	0:02.41	snapped

5) \$ htop

It is similar to top , but output will come in gui style.

We have to install separately.

\$ sudo apt install htop



How to kill the process:

\$ kill -9 procesid

Eg: kill -9 21484

We can kill multiple processes simultaneously.

kill -9 pid1 pid2 pid3 ...



Topic-47: Job Control

All jobs are divided into 2 types

- 1) Foreground Jobs
- 2) Background Jobs

\$ cp file1.txt file2.txt → Foreground Job
\$ cp file1.txt file2.txt & → Background Job

```
durga@durga-VirtualBox:~$ cp a.txt b.txt &
[1] 2972
durga@durga-VirtualBox:~$ cp a.txt b.txt &
[2] 2974
[1] Done          cp a.txt b.txt
durga@durga-VirtualBox:~$ jobs
[2]+ Done          cp a.txt b.txt
```

How to Kill Foreground Job:

ctrl+c

How to suspend foreground job:

ctrl+z

How to Resume Foreground Job:

\$fg

test.sh

```
#!/bin/bash
while [ true ]
do
    echo "Inside while loop"
    sleep 1
done
```

```
durga@durga-VirtualBox:~/scripts$ test.sh
Inside while loop
Inside while loop
Inside while loop
Inside while loop
```



```
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
^C
durga@durga-VirtualBox:~/scripts$ test.sh
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
^Z
[1]+  Stopped                  test.sh
durga@durga-VirtualBox:~/scripts$ fg
test.sh
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
Inside while loop
^C
```

How to Kill Background Job:

```
$ kill -9 pid
```

How to List all Background Jobs:

```
$ jobs
durga@durga-VirtualBox:~/scripts$ jobs
[1]+  Killed                  test.sh
```

How to bring Background Job to Foreground?

```
$ fg jobid
Eg: $fg 1
```



How to send Foreground Job to Background:

First suspend foreground job by using ctrl+z and then use the following command

\$ bg

Note: In the foreground only one can be executed, but in the background multiple jobs can be executed.



Topic-50: Memory related Commands

1) df Command:

df means disk fragmentation.

It displays file system disk space usage. It provides information about the space available on all currently mounted file systems.

```
durga@durga-VirtualBox:~$ df
```

Filesystem	1K-blocks	Used	Available	Use%	Mounted on
udev	1404040	0	1404040	0%	/dev
tmpfs	285584	1584	284000	1%	/run
/dev/sda1	10253588	5512804	4200216	57%	/
tmpfs	1427912	0	1427912	0%	/dev/shm
tmpfs	5120	4	5116	1%	/run/lock
tmpfs	1427912	0	1427912	0%	/sys/fs/cgroup

We can use multiple options with df command.

-h → Human Readable Format

-m → In Mega Bytes

-k → In Kilo Bytes (It is Default)

```
durga@durga-VirtualBox:~$ df -m
```

Filesystem	1M-blocks	Used	Available	Use%	Mounted on
udev	1372	0	1372	0%	/dev
tmpfs	279	2	278	1%	/run
/dev/sda1	10014	5384	4102	57%	/
tmpfs	1395	0	1395	0%	/dev/shm
tmpfs	5	1	5	1%	/run/lock

```
durga@durga-VirtualBox:~$ df -h
```

Filesystem	Size	Used	Avail	Use%	Mounted on
udev	1.4G	0	1.4G	0%	/dev
tmpfs	279M	1.6M	278M	1%	/run
/dev/sda1	9.8G	5.3G	4.1G	57%	/
tmpfs	1.4G	0	1.4G	0%	/dev/shm
tmpfs	5.0M	4.0K	5.0M	1%	/run/lock
tmpfs	1.4G	0	1.4G	0%	/sys/fs/cgroup
/dev/loop1	43M	43M	0	100%	/snap/gtk-common-themes/1313



2) du Command:

du - estimate file space usage

It displays directory wise disk usage in the form of blocks. Each block is 1024 bytes.

```
durga@durga-VirtualBox:~$ du
4      ./dir10
4      ./Downloads/java/corejava
8      ./Downloads/java
12     ./Downloads
du: cannot read directory './.dbus': Permission denied
4      ./dbus
4      ./config/update-notifier
8      ./config/gedit
12     ./config/ibus/bus
```

With du command we can use multiple options

-k in kilobytes(It is the default value)

-h in human readable form

-m in mega bytes

```
durga@durga-VirtualBox:~$ du -k
4      ./dir10
4      ./Downloads/java/corejava
8      ./Downloads/java
12     ./Downloads
du: cannot read directory './.dbus': Permission denied
4      ./dbus
4      ./config/update-notifier
8      ./config/gedit
12     ./config/ibus/bus
16     ./config/ibus
4      ./config/enchant
84     ./config/pulse
4      ./config/goa-1.0
8      ./config/gtk-3.0
8      ./config/evolution/sources
12     ./config/evolution
```

```
durga@durga-VirtualBox:~$ du -h
4.0K   ./dir10
4.0K   ./Downloads/java/corejava
8.0K   ./Downloads/java
12K    ./Downloads
du: cannot read directory './.dbus': Permission denied
4.0K   ./dbus
```



```
4.0K  ./config/update-notifier
8.0K  ./config/gedit
12K   ./config/ibus/bus
16K   ./config/ibus
4.0K  ./config/enchant
84K   ./config/pulse
4.0K  ./config/goa-1.0
8.0K  ./config/gtk-3.0
8.0K  ./config/evolution/sources
12K   ./config/evolution
4.0K  ./config/gnome-session/saved-session
8.0K  ./config/gnome-session
28K   ./config/nautilus
20K   ./config/dconf
212K  ./config
16K   ./output5
260K  ./cache/mozilla/firefox/z901vq89.default-release/OfflineCache
12M   ./cache/mozilla/firefox/z901vq89.default-release/startupCache
```

```
durga@durga-VirtualBox:~$ du -m
```

```
1     ./dir10
1     ./Downloads/java/corejava
1     ./Downloads/java
1     ./Downloads
```

Note: By default du command will always display disk information only about directories. If we want to get files information also then we have to use -a option.
-a means all

```
durga@durga-VirtualBox:~$ du -a -h | head -10
```

```
du: cannot read directory './.dbus': Permission denied
du: cannot read directory './.cache/doc': Permission denied
du: cannot read directory './.cache/dconf': Permission denied
du: cannot read directory './.gvfs': Permission denied
0     ./dir10/c.txt
0     ./dir10/d.txt
0     ./dir10/a.txt
0     ./dir10/b.txt
4.0K  ./dir10
8.0K  ./ICEauthority
0     ./Downloads/java/corejava/collections.txt
0     ./Downloads/java/corejava/arrays.txt
4.0K  ./Downloads/java/corejava
8.0K  ./Downloads/java
```



Note: We can use -c option with du command to find grand total disk space.

```
durga@durga-VirtualBox:~$ du -a -h -c
0      ./dir10/c.txt
0      ./dir10/d.txt
0      ./dir10/a.txt
0      ./dir10/b.txt
4.0K   ./dir10
8.0K   ./ICEauthority
.....
40M    total
```

3) free Command:

It displays amount of free and used memory in the system.

```
durga@durga-VirtualBox:~$ free
              total        used         free       shared    buff/cache   available
Mem:        2855828      1205468       646652         51552        1003708       1434376
Swap:        483800           0         483800
```

Here also we can use -k,-h,-m options.

```
durga@durga-VirtualBox:~$ free -k
              total        used         free       shared    buff/cache   available
Mem:        2855828      1205112       646888         51556        1003828       1434716
Swap:        483800           0         483800
```

```
durga@durga-VirtualBox:~$ free -h
              total        used         free       shared    buff/cache   available
Mem:          2.7G          1.1G          631M           50M           980M           1.4G
Swap:          472M           0B           472M
```

```
durga@durga-VirtualBox:~$ free -m
              total        used         free       shared    buff/cache   available
Mem:           2788          1176           631            50            980           1401
Swap:           472           0            472
```

Note: We can use -l and -t options also.

-l, --lohi

Show detailed low and high memory statistics.

-t, --total

Display a line showing the column totals.

```
durga@durga-VirtualBox:~$ free -l
              total        used         free       shared    buff/cache   available
Mem:        2855828      1205100       646660         51552        1004068       1434736
Low:        2855828      2209168       646660
High:           0           0           0
Swap:        483800           0         483800
```



UNIX/LINUX



```
durga@durga-VirtualBox:~$ free -t
```

	total	used	free	shared	buff/cache	available
Mem:	2855828	1206092	645620	51556	1004116	1433732
Swap:	483800	0	483800			
Total:	3339628	1206092	1129420			



Topic-39: User Management

- 1) How to add a new group?
- 2) How to add a new user?
- 3) Switching from one user to another user
- 4) How to get information about a particular user?
- 5) How to delete user?
- 6) How to delete group ?
- 7) How to change ownership of a file?
- 8) How to change group membership of a file?
- 9) How to change group of a user ?
- 10) Add a User to Multiple Groups
- 11) How to check available groups?
- 12) How to change password of a user?
- 13) What is the difference between adduser and useradd in Linux?
- 14) sudo command
- 15) sudoers file
- 16) sudo command
- 17) sudoers file
- 18) How to check the allowed commands by sudo for a particular user?

1) How to add a New Group?

We have to use addgroup command. For this sudo permission is required.

sudo means superuser do.

```
durga@durga-VirtualBox:~$ sudo addgroup pythongroup
[sudo] password for durga:
Adding group `pythongroup' (GID 1006) ...
Done.
```

Note: We can see all created groups information inside /etc/group file.



2) How to add User:

We have to use adduser command. For this sudo permission is required.

```
durga@durga-VirtualBox:~$ sudo adduser --ingroup pythongroup pyuser1
Adding user `pyuser1' ...
Adding new user `pyuser1' (1005) with group `pythongroup' ...
Creating home directory `/home/pyuser1' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for pyuser1
Enter the new value, or press ENTER for the default
    Full Name []:
    Room Number []:
    Work Phone []:
    Home Phone []:
    Other []:
Is the information correct? [Y/n] y
```

Note: While creating new user, if we are not specifying groupname, then a new group will be created with the same name as username.

```
durga@durga-VirtualBox:~$ sudo adduser pyuser2
Adding user `pyuser2' ...
Adding new group `pyuser2' (1007) ...
Adding new user `pyuser2' (1006) with group `pyuser2' ...
Creating home directory `/home/pyuser2' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for pyuser2
Enter the new value, or press ENTER for the default
    Full Name []:
    Room Number []:
    Work Phone []:
    Home Phone []:
    Other []:
Is the information correct? [Y/n] y
```



Note: We can see all created users information inside /etc/passwd file.

```
durga@durga-VirtualBox:~$ cat /etc/passwd
```

```
...
```

```
pyuser1:x:1005:1006:,,,:/home/pyuser1:/bin/bash
```

```
pyuser2:x:1006:1007:,,,:/home/pyuser2:/bin/bash
```

3) Switching from One User to another User:

We have to use su command.

su means switch user.

syntax: su newuser

```
durga@durga-VirtualBox:~$ whoami
```

```
durga
```

```
durga@durga-VirtualBox:~$ su pyuser1
```

```
Password:
```

```
pyuser1@durga-VirtualBox:/home/durga$ whoami
```

```
pyuser1
```

```
pyuser1@durga-VirtualBox:/home/durga$ exit
```

```
exit
```

```
durga@durga-VirtualBox:~$ whoami
```

```
durga
```

Note: If we use su command without any argument, then it will switch to root user.

Hence the following 2 commands are equal. (But in ubuntu this option is not working)

```
$ su
```

```
$ su root
```

In Ubuntu, to switch to root user , we have to use:

```
durga@durga-VirtualBox:~$ sudo -i
```

su with and without - Option:

If we use su command without - option, then only user will be switched but environment won't be switched.

```
durga@durga-VirtualBox:~$ whoami
```

```
durga
```

```
durga@durga-VirtualBox:~$ pwd
```

```
/home/durga
```

```
durga@durga-VirtualBox:~$ export x=9999
```

```
durga@durga-VirtualBox:~$ echo $x
```

```
9999
```



```
durga@durga-VirtualBox:~$ su pyuser1
Password:
pyuser1@durga-VirtualBox:/home/durga$ whoami
pyuser1
pyuser1@durga-VirtualBox:/home/durga$ pwd
/home/durga
pyuser1@durga-VirtualBox:/home/durga$ echo $x
9999
```

If we use - option with su command, then both user and environment will be switched.

```
durga@durga-VirtualBox:~$ whoami
durga
durga@durga-VirtualBox:~$ pwd
/home/durga
durga@durga-VirtualBox:~$ export x=9999
durga@durga-VirtualBox:~$ echo $x
9999
durga@durga-VirtualBox:~$ su - pyuser1
Password:
pyuser1@durga-VirtualBox:~$ whoami
pyuser1
pyuser1@durga-VirtualBox:~$ pwd
/home/pyuser1
pyuser1@durga-VirtualBox:~$ echo $x
Here x is not available because x is environment variable of durga user.
```

Q) What is the difference between the following?

\$ su pyuser1 → Only user will be switched but not environment.

\$ su - pyuser1 → Both user and environment will be switched.

4) How to get Information about a particular User:

We have to use finger command. But by default this command is not available. we have to install manually.

```
durga@durga-VirtualBox:~$ finger durga
```

Command 'finger' not found, but can be installed with:

```
sudo apt install finger
```

```
durga@durga-VirtualBox:~$ sudo apt install finger
[sudo] password for durga:
Reading package lists... Done
```



Building dependency tree

Reading state information... Done

The following packages were automatically installed and are no longer required:

fonts-liberation2 fonts-opensymbol
gir1.2-geocodeglib-1.0 gir1.2-gst-plugins-base-1.0
gir1.2-gstreamer-1.0 gir1.2-gudev-1.0
gir1.2-udisks-2.0 grilo-plugins-0.3-base
gstreamer1.0-gtk3 libboost-date-time1.65.1
libboost-filesystem1.65.1 libboost-iostreams1.65.1
libboost-locale1.65.1 libcdr-0.1-1
libclucene-contribs1v5 libclucene-core1v5
libcmis-0.5-5v5 libcolamd2 libdazzle-1.0-0
libe-book-0.1-1 libedataserverui-1.2-2 libeot0
libepubgen-0.1-1 libetonyek-0.1-1 libevent-2.1-6
libexiv2-14 libfreerdp-client2-2 libfreerdp2-2
libgee-0.8-2 libgexiv2-2 libgom-1.0-0 libgpgmepp6
libgpod-common libgpod4 liblangtag-common liblangtag1
liblirc-client0 liblua5.3-0 libmediaart-2.0-0
libmsspub-0.1-1 libodfgen-0.1-1 libqqwing2v5 libraw16
librevenge-0.0-0 libsgutils2-2 libssh-4
libsuitesparseconfig5 libvncclient1 libwinpr2-2
libxapian3 libxmlsec1 libxmlsec1-nss lp-solve
media-player-info python3-mako python3-markupsafe
syslinux syslinux-common syslinux-legacy
usb-creator-common

Use 'sudo apt autoremove' to remove them.

The following NEW packages will be installed:

finger

0 upgraded, 1 newly installed, 0 to remove and 235 not upgraded.

Need to get 15.6 kB of archives.

After this operation, 48.1 kB of additional disk space will be used.

Get:1 <http://in.archive.ubuntu.com/ubuntu/bionic/universe/amd64/finger> amd64 0.17-15.1 [15.6 kB]

Fetch: 15.6 kB in 8s (1,977 B/s)

Selecting previously unselected package finger.

(Reading database ... 119576 files and directories currently installed.)

Preparing to unpack .../finger_0.17-15.1_amd64.deb ...

Unpacking finger (0.17-15.1) ...

Setting up finger (0.17-15.1) ...

Processing triggers for man-db (2.8.3-2ubuntu0.1) ...

durga@durga-VirtualBox:~\$ finger durga

Login: durga Name: durga

Directory: /home/durga Shell: /bin/bash

On since Tue Jan 28 16:56 (IST) on :0 from :0 (messages off)



Mail last read Wed Jan 8 20:44 2020 (IST)
No Plan.

5) How to Delete User?

We have to use deluser command.

To delete user compulsory sudo permission must be required.

```
durga@durga-VirtualBox:~$ sudo deluser pyuser2
Removing user `pyuser2' ...
Warning: group `pyuser2' has no more members.
Done.
```

```
durga@durga-VirtualBox:~$ su pyuser2
No passwd entry for user 'pyuser2'
```

Note: We can check whether user deleted or not by using /etc/passwd file.

6) How to Delete Group?

We have to use delgroup command.

For this sudo access must be required and no user associated with the group.

```
durga@durga-VirtualBox:~$ sudo delgroup pythongroup
/usr/sbin/delgroup: `pyuser1' still has `pythongroup' as their primary group!
```

```
durga@durga-VirtualBox:~$ sudo delgroup demogroup1
Removing group `demogroup1' ...
Done.
```

7) How to Change Ownership of a File:

We can change by using chown command.

But for this sudo permission required.

```
durga@durga-VirtualBox:~$ touch ddd.txt
durga@durga-VirtualBox:~$ ls -l ddd.txt
-rw-r--r-- 1 durga durga 0 Jan 28 17:16 ddd.txt
durga@durga-VirtualBox:~$ chown root ddd.txt
chown: changing ownership of 'ddd.txt': Operation not permitted
durga@durga-VirtualBox:~$ sudo chown root ddd.txt
durga@durga-VirtualBox:~$ ls -l ddd.txt
-rw-r--r-- 1 root durga 0 Jan 28 17:16 ddd.txt
```



8) How to Change Group Membership of a File?

We have to use chgrp command.

chgrp means change group.

To use this command compulsory sudo permission must be required.

```
durga@durga-VirtualBox:~$ ls -l ddd.txt
-rw-r--r-- 1 root durga 0 Jan 28 17:16 ddd.txt
durga@durga-VirtualBox:~$ sudo chgrp root ddd.txt
durga@durga-VirtualBox:~$ ls -l ddd.txt
-rw-r--r-- 1 root root 0 Jan 28 17:16 ddd.txt
```

9) How to Change Group of a User:

We have to use usermod command.

Syntax: \$ usermod -g groupname username

```
durga@durga-VirtualBox:~$ sudo usermod -g durga pyuser7
durga@durga-VirtualBox:~$ su - pyuser7
Password:
pyuser7@durga-VirtualBox:~$ groups
durga
```

Note: We can find groups associated with particular user by using groups command as follows

```
$ groups username
```

10) Add a User to Multiple Groups:

While assigning the secondary groups to a user account, we can easily assign multiple groups at once by separating the list with a comma.

```
$ usermod -a -G group1,group2,group3 username
```

```
durga@durga-VirtualBox:~$ groups uiuser1
uiuser1 : uiuser1
durga@durga-VirtualBox:~$ sudo usermod -a -G uiuser2,uiuser3 uiuser1
durga@durga-VirtualBox:~$ groups uiuser1
uiuser1 : uiuser1 uiuser2 uiuser3
```



11) How to Check available Groups:

To check available groups:

```
$ groups
```

This command lists all the groups that you belong to.

To check which group a certain user belongs to:

```
$ groups [username]
```

12) How to Change Password of User:

To change a password for user named durga:

```
$ sudo passwd durga
```

To change a password for root user:

```
$ sudo passwd root
```

To change your own password:

```
$ passwd
```

13) What is the difference between adduser and useradd in Linux?

Useradd is built-in Linux command that can be found on any Linux system.

However, creating new users with this low-level is a difficult and lengthy task. First we have to create user and then we have to provide password and extra information separately.

Adduser is not a standard Linux command. It is essentially a Perl script that uses the useradd command in the background. This high-level utility is more efficient in properly creating new users on Linux. Default parameters for all new users can also be set through the adduser command.

```
durgasoft@durgasoft:/etc$ sudo adduser durga5
Adding user `durga5' ...
Adding new group `durga5' (1004) ...
Adding new user `durga5' (1004) with group `durga5' ...
Creating home directory `/home/durga5' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for durga5
Enter the new value, or press ENTER for the default
    Full Name []:
    Room Number []:
```



Work Phone []:

Home Phone []:

Other []:

Is the information correct? [Y/n]

```
durgasoft@durgasoft:/etc$ sudo useradd durga98
```

14) sudo Command:

sudo means Super User Do.

We can use sudo command to execute commands as another user, mostly root user.

Eg: \$ sudo adduser durga2

We are executing adduser command as root.

```
$ sudo -u durga1 command
```

We are using executing the command as durga1 user.

Note: If we want to use sudo command for any user other than root then we have to use -u option.

The following two commands are equal.

```
$ sudo command
```

```
$ sudo -u root command
```

15) sudoers File:

Which commands can be executed by using sudo of a particular user, information is configured in sudoers file, which is present in /etc directory.

System Administrators are responsible to configure this file.

```
durgasoft@durgasoft:~$ ls -l /etc/sudoers
```

```
-r--r----- 1 root root 755 Jan 18 2018 /etc/sudoers
```

```
durgasoft@durgasoft:~$ cat /etc/sudoers
```

```
cat: /etc/sudoers: Permission denied
```

```
durgasoft@durgasoft:~$ sudo cat /etc/sudoers
```

```
#
```

```
# This file MUST be edited with the 'visudo' command as root.
```

```
#
```

```
# Please consider adding local content in /etc/sudoers.d/ instead of  
# directly modifying this file.
```

```
#
```

```
# See the man page for details on how to write a sudoers file.
```

```
#
```

```
Defaults    env_reset
```



```
Defaults      mail_badpass
Defaults
    secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/
snap/bin"

# Host alias specification

# User alias specification

# Cmnd alias specification

# User privilege specification
root    ALL=(ALL:ALL) ALL

# Members of the admin group may gain root privileges
%admin  ALL=(ALL) ALL

# Allow members of group sudo to execute any command
%sudo  ALL=(ALL:ALL) ALL

# See sudoers(5) for more information on "#include" directives:

#includedir /etc/sudoers.d
```

16) How to Check the allowed Commands by sudo for a particular User?

\$ sudo -l

It will display allowed commands of current user by sudo.

\$ sudo -l -u username

It will display allowed commands of provided user by sudo.

durgasoft@durgasoft:~\$ sudo -l

Matching Defaults entries for durgasoft on durgasoft:

env_reset, mail_badpass,

secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

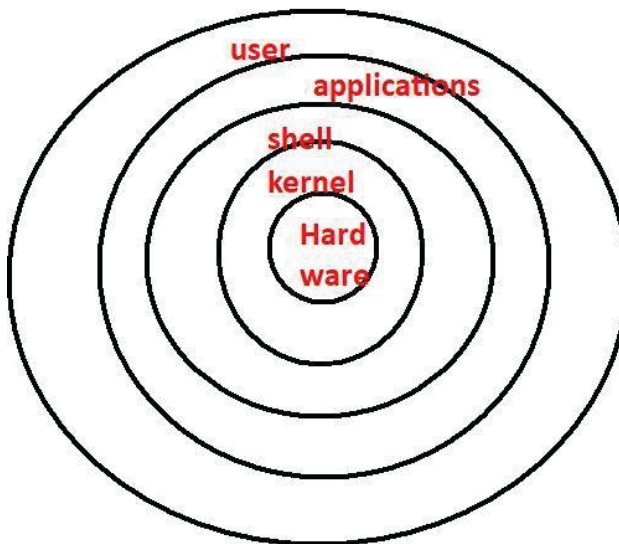
User durgasoft may run the following commands on durgasoft:

(ALL : ALL) ALL



Topic-24: What is Shell and Types of Shells

What is Shell?



- 1) Shell is responsible to read commands/applications provided by user.
- 2) Shell will check whether command is valid or not and whether it is properly used or not. If everything is proper then shell interpretes (converts) that command into kernel understandable form. That interpreted command will be handover to kernel.
- 3) Kernel is responsible to execute that command with the help of hardware.

Shell acts as interface between user and kernel.

shell+kernel is nothing but operating system.

Types of Shells:

There are multiple types of shells are available.

1) Bourne Shell:

- It is developed by Stephen Bourne.
- This is the first shell which is developed for UNIX.
- By using sh command we can access this shell.



2) BASH Shell:

- BASH → Bourne Again SHell.
- It is advanced version of Bourne Shell.
- This is the default shell for most of the linux flavours.
- By using bsh command we can access this shell.

3) Korn Shell:

- It is developed by David Korn.
- Mostly this shell used in IBM AIX operating system.
- By using ksh command, we can access this shell.

4) CShell:

- Developed by Bill Joy.
- C meant for California University.
- It is also by default available with UNIX.
- By using csh command, we can access this shell.

5) TShell:

- T means Terminal.
- It is the advanced version of CShell.
- This is the most commonly used shell in HP UNIX.
- By using tcsh command, we can access this shell.

6) Z Shell:

- Developed by Paul.
- By using zsh command we can access this shell.

Note: The most commonly used shell in linux environment is BASH. It is more powerful than remaining shells.

How to Check Default Shell in our System?

```
$ echo $0
bash
$ echo $SHELL
/bin/bash
```

We can also check the default shell information inside /etc/passwd file

```
$ cat /etc/passwd
```

```
durgasoft:x:1000:1000:durgasoft,,,:/home/durgasoft:/bin/bash
```



How to check all available Shells in our System?

/etc/shells file contains all available shells information.

```
$ cat /etc/shells
# /etc/shells: valid login shells
/bin/sh
/bin/bash
/bin/rbash
/bin/dash
```

How to Switch to other Shells?

Based on our requirement we can switch from one shell to another shell.

```
durgasoft@durgasoft:~/Desktop$ sh
$ echo $0
sh
$ exit
durgasoft@durgasoft:~/Desktop$ echo $0
bash
durgasoft@durgasoft:~/Desktop$ rbash
durgasoft@durgasoft:~/Desktop$ echo $0
rbash
durgasoft@durgasoft:~/Desktop$ exit
exit
durgasoft@durgasoft:~/Desktop$ dash
$ echo $0
dash
$ exit
```



Topic-25: What is Shell Script, Sha-Bang and First Script

What is Shell Script:

A sequence of commands saved to a file and this file is nothing but shell script.

Inside shell script, we can also use programming features like conditional statements, loops, functions etc. Hence we can write scripts very easily for complex requirements also.

Best suitable for automated tasks.

How to write and run Shell Script:

Step - 1: Write script

demo.sh:

```
echo "Welcome to shell script"
date
cal
```

Step - 2: Provide execute permissions to the script:

```
$ chmod a+x demo.sh
```

Step - 3: Run the script

We can run the script in multiple ways

```
$ /bin/bash ./demo.sh
$ bash ./demo.sh
$ /bin/bash /home/durgasoft/scripts/demo.sh
$ ./demo.sh # default shell is bash
```

Note: The default shell is bash. Hence bash is responsible to execute our script. Instead of bash, if we want to use Bourne shell then we have to use the following command

```
$ /bin/sh ./demo.sh
$ sh ./demo.sh
```



Importance of Sha-Bang:

By using sha-bang, we can specify the interpreter which is responsible to execute the script.

→ Sharp

! → Bang

#! → Sharp Bang or Shabang or Shebang

#!/bin/bash → It means the script should be executed by bash

#!/bin/sh → It means the script should be executed by Bourne Shell

#!/usr/bin/python3 → It means the script should be executed by Python3 interpreter

If we write shabang in our script at the time of execution, we are not required to provide command to execute and we can execute script directly.

Q1) Write a Python Script and execute without shabang and with shabang?

test.py

```
#!/usr/bin/python3
```

```
import random
```

```
name = input("Enter Your Name:")
```

```
l=["Sunny","Katrina","Kareena","Mallika","Malaika"]
```

```
print("Hello:",name)
```

```
print("Congratulations..You are going to marry:",random.choice(l))
```

Without Shabang:

```
$ python3 ./test.py
```

```
$ python3 /home/durgasoft/scripts/test.py
```

With Shabang:

```
$ ./test.py
```

```
$ /home/durgasoft/scripts/test.py
```

Q2) Consider the folloiwng Script:

demo.sh

```
#!/bin/rm
```

```
echo "It is beautiful script"
```

If we execute this script what will happend?

```
$ ./demo.sh
```



It is equivalent to

```
$ rm ./demo.sh
```

demo.sh will be removed as this script executed by rm command.

Q3) Write and Run Shell Script that Prints Current System Date and Time

date.sh

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

```
echo "The current System Date and Time:"
```

```
date
```

Provide execute permissions for this script

```
$ chmod a+x date.sh
```

```
$/date.sh
```

Q4) How to Run Our Script from any where in our System?

- For any command or script, by default shell will check locations specified by PATH variable.
- If we add our script location to PATH Variable value, then we can run our script from anywhere in our system. We can do this in the following two ways:

1) Session Level:

```
durgasoft@durgasoft:~/scripts$ echo $PATH
```

```
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:  
/snap/bin
```

```
$ export PATH=$PATH:/home/durgasoft/scripts
```

```
durgasoft@durgasoft:~/scripts$ echo $PATH
```

```
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:  
/snap/bin:/home/durgasoft/scripts
```

Now we can run our script from anywhere. We are not required to provide either absolute path or relative path.

```
$date.sh
```

```
The current System Date and Time:
```

```
Wed Dec 4 21:01:49 IST 2019
```



Note: This way of setting PATH variable is applicable only for current session. Once we close terminal then automatically these changes will be lost.

2) Setting PATH permanently at User Level:

- In our user home directory there is a file named with .bashrc. This file will be executed every time automatically whenever we open the terminal. If we define PATH variable in this file then that PATH value will become permanent and we are not required to set every time.
- Add the following line at bottom of this file
- `export PATH=$PATH:/home/durgasoft/scripts`

Q5) What is the meaning of Startup File?

- .bashrc is the startup file and will be executed automatically at the time of terminal starting.
- Hence if we want to perform any activity while starting the terminal we have to define that activity in this file.
- eg: creating aliases and updating PATH variable value etc

Q6) What is meant by logout File?

- .bash_logout is logout file and will be executed automatically at the time of terminal exit.
- Hence if we want to perform any activity at terminal exit, then we have to define that activity inside this file.



Topic-26: Shell Variables

Variables are place holders to hold values.

Variables are key-value pairs.

In Shell programming, there are no data types. Every value is treated as text type/ String type.

All variables are divided into 2 types

- 1) Environment variables / predefined variables
- 2) User defined variables

1) Environment Variables:

These are predefined variables and mostly used internally by the system. Hence these variables also known as System variables.

But based on our requirement, we can use these variables in our scripts.

We can get all environment variables information by using either env command or set command.

```
durgasoft@durgasoft:~$ env
LANG=en_IN
USERNAME=durgasoft
USER=durgasoft
PWD=/home/durgasoft
HOME=/home/durgasoft
SHELL=/bin/bash
LOGNAME=durgasoft
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/home/durgasoft/scripts
...

durgasoft@durgasoft:~$ set
BASH=/bin/bash
```



```
LANG=en_IN
LOGNAME=durgasoft
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/
games:/snap/bin:/home/durgasoft/scripts
PS1='\[\e]0;\u@\h:
\w\a\]${debian_chroot:+($debian_chroot)}\[\033[01;32m\]\u@\h\[\033[00m\]:\[\033[0
1;34m\]\w\[\033[00m\]\$ '
PWD=/home/durgasoft
SHELL=/bin/bash
```

How to Change Prompt:

Internally PS1 Environment variable is responsible to display terminal prompt. By reassigning the value we can change prompt.

```
durgasoft@durgasoft:~$
durgasoft@durgasoft:~$ PS1=DURGA$
DURGA$ls
abc.txt Documents d.txt Pictures scripts Videos y
Desktop Downloads Music Public Templates x
DURGA$cat > abc.txt
asdfjkasjfdsa
asfdkasklfdjlads
DURGA$PS1=Sunny#
Sunny#ls -l
total 52
-rw-r--r-- 1 durgasoft durgasoft 31 Dec 5 20:58 abc.txt
drwxr-xr-x 2 durgasoft durgasoft 4096 Dec 5 18:04 Desktop
drwxr-xr-x 2 durgasoft durgasoft 4096 Nov 21 17:06 Documents
drwxr-xr-x 2 durgasoft durgasoft 4096 Nov 21 17:06 Downloads
```

Demo Script to Use some Environment Variables:

env.sh

```
#!/bin/bash
echo "User Name: $USER"
echo "User Home Directory: $HOME"
echo "Default Shell Name: $SHELL"
echo "PATH: $PATH"

$chmod 755 env.sh
$./env.sh
User Name: durgasoft
User Home Directory: /home/durgasoft
```



Default Shell Name: /bin/bash

PATH:

```
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:  
/snap/bin:/home/durgasoft/scripts
```

2) User defined Variables:

Based on our programming requirement, we can define our own variables also.

```
durgasoft@durgasoft:~/scripts$ GUEST=Dhoni
```

```
durgasoft@durgasoft:~/scripts$ echo "Hello $GUEST You are my Hero"
```

```
Hello Dhoni You are my Hero
```

Rules to define Variables:

- 1) It is recommended to use only UPPER CASE characters.
- 2) If variable contains multiple words, then these words should be separated with _ symbol.
- 3) Variable names should not start with digit.

```
durgasoft@durgasoft:~/scripts$ 123A=40  
123A=40: command not found
```
- 4) We should not use special symbols like -, @, # etc

How to define Readonly Variables:

We can define read only variables by using readonly keyword.

```
$A=100
```

```
$readonly A
```

```
$A=300
```

```
bash: A: readonly variable
```

If the variable is readonly then we cannot perform reassignment for that variable. It will become constant.



Variable Scopes:

There are 3 scopes available for variables.

- 1) Session Scope
- 2) User Scope
- 3) System Scope

1) Session Scope:

The variables which are declared in the terminal are said to be in session scope. Once we close the terminal (ie exit session) automatically all variables will be gone.

```
$ X=10
$ Y=10
```

2) User Scope:

- The variables which are declared inside .bashrc file, are said to be in user scope.
- These variables are available for all sessions related to current user. These variables cannot be accessed by other users.

```
.bashrc
....
export GUEST=Dhoni
export FRIEND=Sunny
```

3) System Scope:

If the variable available for all users and for all sessions, such type of variables are said to be in system scope.

We have to declare these variables inside /etc/profile file.
But to edit this file compulsory root permissions must be required.

```
$ sudo gedit /etc/profile

...
export NAME=DURGA
export COURSE=PYTHON
```



Topic-27: Variable Substitution and Command Substitution

Variable Substitution:

Accessing the value of a variable by using \$ symbol is called variable substitution.

Syntax:

\$variablename
\${variablename}

Recommended to use \${variablename}.

test.sh

```
#!/bin/bash
a=10
b=20
COURSE="linux"
ACTION="SLEEP"
echo "Values of a and b are: $a and $b"
echo "My Course is: ${COURSE}"
echo "My Favourite Action: $ACTIONING"
echo "My Favourite Action: ${ACTION}ING"
```

Output:

```
Values of a and b are: 10 and 20
My Course is: linux
My Favourite Action:
My Favourite Action: SLEEPING
```

Q1) Consider the following variable declaration NAME="durga"
Which of the following is valid way to print value of NAME?

- A) echo NAME
- B) echo \$NAME
- C) echo '\$NAME'
- D) echo "\$NAME"

Ans: B, D



Note: If we use single quotes then variable substitution won't be happen. But we can use double quotes.

```
durgasoft@durgasoft:~/scripts$ NAME='durga'
durgasoft@durgasoft:~/scripts$ echo NAME
NAME
durgasoft@durgasoft:~/scripts$ echo $NAME
durga
durgasoft@durgasoft:~/scripts$ echo '$NAME'
$NAME
durgasoft@durgasoft:~/scripts$ echo "$NAME"
durga
```

Command Substitution:

We can execute command and we can substitute its result based on our requirement by using command substitution.

Syntax:

Old style: `command` These are backquotes butnot single quotes

New Style: \$(command)

Eg 1:

```
durgasoft@durgasoft:~/scripts$ echo "Today Date is: `date +%D` "
Today Date is: 12/06/19
durgasoft@durgasoft:~/scripts$ echo "Today Date is: $(date +%D) "
Today Date is: 12/06/19
```

Eg 2:

```
durgasoft@durgasoft:~/scripts$ echo "Number of files in Current Working Directory: `ls |
wc -l` "
Number of files in Current Working Directory: 5
durgasoft@durgasoft:~/scripts$ echo "Number of files in Current Working Directory: $(ls |
wc -l) "
Number of files in Current Working Directory: 5
```

Eg 3:

```
durgasoft@durgasoft:~/scripts$ echo "The No of Lines in test.sh: `cat test.sh | wc -l` "
The No of Lines in test.sh: 9
durgasoft@durgasoft:~/scripts$ echo "The No of Lines in test.sh: $(wc -l test.sh) "
The No of Lines in test.sh: 9 test.sh
```



Eg 4:

```
durgasoft@durgasoft:~/scripts$ echo "My Current Working Directory: `pwd` "
```

```
My Current Working Directory: /home/durgasoft/scripts
```

```
durgasoft@durgasoft:~/scripts$ echo "My Current Working Directory: ${pwd} "
```

```
My Current Working Directory: /home/durgasoft/scripts
```



Topic-28: Command Line Arguments

The arguments which are passing from the command prompt at the time of executing our script, are called command line arguments.

```
$ ./test.sh learning linux is very easy
```

The command line arguments are learning, linux, is, very, easy.

Inside script we can access command line arguments as follows:

```
$# → Number of Arguments (5)
$0 → Script Name (./test.sh)
$1 → 1st Argument (learning)
$2 → 2nd Argument (linux)
$3 → 3rd Argument (is)
$4 → 4th Argument (very)
$5 → 5th Argument (easy)
$* → All Arguments (learning Linux is very easy)
@$ → All Arguments (learning Linux is very easy)
 $? → Represents exit code of previously executed command or script.
```

Eg: test.sh

```
#!/bin/bash
echo "Number of arguments: $#"
```

```
echo "Script Name: $0"
```

```
echo "First argument: $1"
```

```
echo "Second argument: $2"
```

```
echo "Third argument: $3"
```

```
echo "Fourth argument: $4"
```

```
echo "Fifth argument: $5"
```

```
echo "Total arguments: $*"
```

```
durgasoft@durgasoft:~/scripts$ ./test.sh learning linux is very easy
Number of arguments: 5
Script Name: ./test.sh
First argument: learning
Second argument: linux
```



Third argument: is
Fourth argument: very
Fifth argument: easy
Total arguments: learning linux is very easy

Difference between \$@ and \$*:

\$@ → All command line arguments with space separator
"\$1" "\$2" "\$3" ...

\$* → All command line arguments as single string
"\$1c\$2c\$3c.."

Where c is the first character of the Internal Field Separator (IFS).
The default first character is space.

How to Check Default IFS:

```
$ set | grep "IFS"
IFS=$' \t\n'
```

How to Change IFS:

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

```
IFS="-"
```

```
echo "Number of arguments: $#"  
echo "Script Name: $0"  
echo "First argument: $1"  
echo "Second argument: $2"  
echo "Third argument: $3"  
echo "Fourth argument: $4"  
echo "Fifth argument: $5"  
echo "Total arguments: $@"  
echo "Total arguments: $*"
```

```
durgasoft@durgasoft:~/scripts$ test.sh learning unix is very easy  
Number of arguments: 5  
Script Name: /home/durgasoft/scripts/test.sh  
First argument: learning  
Second argument: unix  
Third argument: is  
Fourth argument: very  
Fifth argument: easy  
Total arguments: learning unix is very easy  
Total arguments: learning-unix-is-very-easy
```



Note: The main purpose of command line arguments is to customize behaviour of the script.

test.sh

```
#!/bin/bash
l=$(echo -n "DURGA" | wc -c)
echo "The Length of given String: $l"
```

This script will work only for string: DURGA

test.sh

```
#!/bin/bash
len=$(echo -n "$1" | wc -c)
echo "The Length of given string $1 : $len"
```

This script will work for any string provided from command prompt.

```
durgasoft@durgasoft:~/scripts$ test.sh
The Length of given string : 0
durgasoft@durgasoft:~/scripts$ test.sh durgasoft
The Length of given durgasoft : 9
durgasoft@durgasoft:~/scripts$ test.sh adsfkjshfdjkhsfkjhksfhk
The Length of given string adsfkjshfdjkhsfkjhksfhk : 24
```

Q1) Write Script to Create Log File with Timestamp

test.sh

```
#!/bin/bash
timestamp=$(date +%d_%m_%Y_%H_%M_%S)
echo "This is data to log file" >> ${timestamp}.log
echo "This is extra data to log file" >> ${timestamp}.log
date >> ${timestamp}.log
echo >> ${timestamp}.log
echo "Data Written to log file successfully"
```

If we execute this script, every time a new file will be created.

If we take

```
timestamp=$(date +%d_%m_%Y_%H_%M)
```

Then log file will be created for every new minute.

For every hour if want a new log file: `timestamp=$(date +%d_%m_%Y_%H)`

For every day if want a new log file: `timestamp=$(date +%d_%m_%Y)`



Topic-29: How to Read Dynamic Data from the User

By using read keyword we can read dynamic data from the end user.

Without Prompt Message:

```
durgasoft@durgasoft:~/scripts$ read a b
100 200
durgasoft@durgasoft:~/scripts$ echo "The values of a and b are:$a and $b"
The values of a and b are:100 and 200
```

With Prompt Message:

Approach-1

test.sh

```
#!/bin/bash
echo "Enter A Value:"
read A
```

```
echo "Enter B Value:"
read B
```

```
echo "A Value: $A"
echo "B Value: $B"
```

```
durgasoft@durgasoft:~/scripts$ test.sh
Enter A Value:
100
Enter B Value:
200
A Value: 100
B Value: 200
```

test.sh

```
#!/bin/bash
echo -n "Enter A Value:"
read A
```



```
echo -n "Enter B Value:"
read B
echo "A Value: $A"
echo "B Value: $B"
durgasoft@durgasoft:~/scripts$ test.sh
Enter A Value:100
Enter B Value:200
A Value: 100
B Value: 200
```

Approach-2

```
#!/bin/bash
read -p "Enter A Value:" A
read -p "Enter B Value:" B
```

```
echo "A Value: $A"
echo "B Value: $B"
```

```
durgasoft@durgasoft:~/scripts$ test.sh
Enter A Value:100
Enter B Value:200
A Value: 100
B Value: 200
```

```
#!/bin/bash
read -p "Enter User Name:" user
read -p "Enter Password:" password
```

```
echo "$user thanks for your information"
```

```
durgasoft@durgasoft:~/scripts$ test.sh
Enter User Name:durga
Enter Password:anushka123
durga thanks for your information
```

Note:

read -p : Just to display prompt message
read -s : It hides input on screen which is provided by end user.

```
#!/bin/bash
read -p "Enter User Name:" user
read -s -p "Enter Password:" password
echo
echo "Hello $user , thanks for your information"
```



```
durgasoft@durgasoft:~/scripts$ test.sh
Enter User Name:Durga
Enter Password:
Hello Durga , thanks for your information
```

Q2) Write a Script to Read Student Data and display to the Console for Confirmation?

```
#!/bin/bash
read -p "Enter Student Name:" name
read -p "Enter Student RollNo:" rollno
read -p "Enter Student Age:" age
read -p "Enter Student Marks:" marks
echo "Please confirm your details"
echo "-----"
echo "Student Name: $name"
echo "Student Rollno: $rollno"
echo "Student Age: $age"
echo "Student Marks: $marks"
```

```
durgasoft@durgasoft:~/scripts$ test.sh
Enter Student Name:Durga
Enter Student RollNo:101
Enter Student Age:40
Enter Student Marks:89
Please confirm your details
-----
Student Name: Durga
Student Rollno: 101
Student Age: 40
Student Marks: 89
```

Q3) Write a Script to Read Employee Details and Save to emp.txt File?

```
#!/bin/bash
read -p "Enter Employee Number:" eno
read -p "Enter Employee Name:" ename
read -p "Enter Employee Salary:" esal
read -p "Enter Employee Address:" eaddr

echo "Below details are saved to the file"
echo "$eno:$ename:$esal:$eaddr"
echo "$eno:$ename:$esal:$eaddr" >> emp.txt
```



```
durgasoft@durgasoft:~/scripts$ test.sh
Enter Employee Number:100
Enter Employee Name:Sunny
Enter Employee Salary:1000
Enter Employee Address:Mumbai
Below details are saved to the file
100:Sunny:1000:Mumbai
durgasoft@durgasoft:~/scripts$ test.sh
Enter Employee Number:200
Enter Employee Name:Bunny
Enter Employee Salary:2000
Enter Employee Address:Hyderabad
Below details are saved to the file
200:Bunny:2000:Hyderabad
durgasoft@durgasoft:~/scripts$ test.sh
Enter Employee Number:300
Enter Employee Name:Chinny
Enter Employee Salary:3000
Enter Employee Address:Chennai
Below details are saved to the file
300:Chinny:3000:Chennai
durgasoft@durgasoft:~/scripts$
durgasoft@durgasoft:~/scripts$ cat emp.txt
100:Sunny:1000:Mumbai
200:Bunny:2000:Hyderabad
300:Chinny:3000:Chennai
```

emp.txt

```
durgasoft@durgasoft:~/scripts$ cat emp.txt
100:Sunny:1000:Mumbai
200:Bunny:2000:Hyderabad
300:Chinny:3000:Chennai
```

Q4) Write a Script that takes a String from the End User and Print its Length?

```
#!/bin/bash
read -p "Enter any string to find length:" str
len=$(echo -n $str | wc -c)
echo "Length of $str : $len"
```

```
durgasoft@durgasoft:~/scripts$ test.sh
Enter any string to find length:apple
Length of apple : 5
durgasoft@durgasoft:~/scripts$ test.sh
```



Enter any string to find length:durgasoftwaresolutions
Length of durgasoftwaresolutions : 22

Q5) Write a Script to Read File Name from the End User and display its Content?

```
#!/bin/bash

read -p "Enter any File name to display its content:" fname
echo "-----"
cat $fname
echo "-----"
durgasoft@durgasoft:~/scripts$ test.sh
Enter any File name to display its content:emp.txt
-----
100:Sunny:1000:Mumbai
200:Bunny:2000:Hyderabad
300:Chinny:3000:Chennai
-----
```

Q6) Write a Script to Read File Name from the End User and Remove Blank Lines Present in that File?

```
#!/bin/bash
read -p "Enter any File name to remove blank lines:" fname
grep -v "^$" $fname > temp.txt
mv temp.txt $fname
echo "In $fname all blank lines deleted"
```

Q7) Write a Script to Read File Name from the End User and Remove Duplicate Lines Present in that File?

```
#!/bin/bash
read -p "Enter any File name to remove duplicate lines:" fname

sort -u $fname > temp.txt
mv temp.txt $fname
echo "In $fname all duplicate lines deleted"
```



Topic-30: Operators

1) Arithmetic Operators

- + → Addition
- → Subtraction
- * → Multiplication (we should use * as it is wild card character)
- / → Division
- % → Modulo Operator

Relational Operators: (Numeric Comparison Operators)

- gt → Greater than
- ge → Greater than or equal to
- lt → Less than
- le → Less than or equal to
- eq → Is equal to
- ne → Not equal to

These operators return boolean value (true/false)

Logical Operators:

- a → Logical AND
- o → Logical OR
- ! → Logical Not

Assignment operator =

Note: Except assignment operator, for all operators we have to provide space before and after operator.

How to perform Mathematical Operations:

There are multiple ways

- 1) By using expr keyword
- 2) By using let keyword
- 3) By using (())
- 4) By using []



1) By using expr Keyword:

expr means expression and it is a keyword.

```
#!/bin/bash
read -p "Enter First Number:" a
read -p "Enter First Number:" b
```

```
sum=`expr $a + $b`
echo "The Sum: $sum"
```

```
sum=$(expr $a + $b)
echo "The Sum: $sum"
```

Note: While using expr keyword, \$ symbol is mandatory.
Space must be required before and after + symbol.

2) By using let Keyword:

```
let sum=a+b
echo "The Sum: $sum"
```

```
let sum=$a+$b
echo "The Sum: $sum"
```

Here \$ symbol is optional. But we should not provide space before and after +.

3) By using (()):

```
sum=$((a+b))
echo "The Sum: $sum"
```

```
sum=$(( $a+$b ))
echo "The Sum: $sum"
```

Here space and \$ symbol, both are optional.

4) By using []:

```
sum=$((a+b))
echo "The Sum: $sum"
```

```
sum=$(( $a+$b ))
echo "The Sum: $sum"
```

Here also space and \$ symbol, both are optional.



Note: All the above 4 approaches will work only for integral arithmetic (only for integer numbers).
If we want to perform floating point arithmetic then we should use bc command.

bc Command:

bc means binary calculator.

We can start binary calculator by using bc command.

```
durgasoft@durgasoft:~/scripts$ bc
bc 1.07.1
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006, 2008, 2012-2017 Free Software
Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
10.5+30.6
41.1
10.2^2
104.0
10.5*3.6
37.8
ctrl+d → To exit bc
```

Note: bc can perform both integral and floating point arithmetic.

Script for floating Point Arithmetic:

```
#!/bin/bash

read -p "Enter First Number:" a
read -p "Enter Second Number:" b

sum=$( echo $a+$b | bc)
echo "The Sum:$sum"

echo "The Difference: $( echo $a-$b | bc)"
echo "The Product: $( echo $a*$b | bc)"
```

Output:

```
durgasoft@durgasoft:~/scripts$ test.sh
Enter First Number:10.5
Enter Second Number:5.3
The Sum:15.8
The Difference: 5.2
The Product: 55.6
```



Q1) Write a Script to Read 2 Integer Numbers and Print

Sum?

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

```
read -p "Enter First Number:" a
read -p "Enter Second Number:" b
sum=$((a+b))
echo "The Sum:$sum"
```

Q2) Write a Script to Read 4 Digit Integer Number and Print the Sum of Digits Present in that Number?

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

```
read -p "Enter Any 4-digit Integer Number:" n
```

```
a=$(( echo $n | cut -c 1 ))
b=$(( echo $n | cut -c 2 ))
c=$(( echo $n | cut -c 3 ))
d=$(( echo $n | cut -c 4 ))
```

```
echo "The Sum of 4 digits : $((a+b+c+d))"
```

```
durgasoft@durgasoft:~/scripts$ test.sh
Enter Any 4-digit Integer Number:1234
The Sum of 4 digits : 10
durgasoft@durgasoft:~/scripts$ test.sh
Enter Any 4-digit Integer Number:3456
The Sum of 4 digits : 18
```

Q3) Write a Script to Read Employee Monthly Salary and Print his Bonus. The Bonus should be 25% of Annual Salary

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

```
read -p "Enter Employee Monthly Salary:" salary
```

```
annual_salary=$((salary*12))
bonus=$((annual_salary*25/100))
echo "The Bonus:$bonus"
```

```
durgasoft@durgasoft:~/scripts$ test.sh
Enter Employee Monthly Salary:10000
The Bonus:30000
```



Topic-31: Control Statements

- 1) if statement
- 2) case statement
- 3) while loop
- 4) for loop
- 5) until loop
- 6) break
- 7) continue
- 8) exit

1) if Statement:

There are 4 types of if statements

- 1) simple if
- 2) if-else
- 3) nested if
- 4) ladder if

1) simple if:

```
if [ condition ]  
then  
    action  
fi
```

If condition is true then only action will be executed.

Q1) WAS to Read Name from the End User and if Name is Sunny then Display some Special Message?

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

```
read -p "Enter Your Name:" name
```

```
if [ $name = "sunny" ]  
then  
    echo "Hello Sunny Very Good Evening"  
fi  
echo "How are you"
```

```
durgasoft@durgasoft:~/scripts$ test.sh
```

```
Enter Your Name:durga
```



```
How are you
durgasoft@durgasoft:~/scripts$ test.sh
Enter Your Name:sunny
Hello Sunny Very Good Evening
How are you
```

Note:

x=10 → Assignment
x = 10 → Comparison

2) if -else:

```
if [ condition ]
then
    action if condition is true
else
    action if condition is false
fi
```

3) Nested if:

```
if [ condition ]
then
    .....
    .....
    if [ condition ]
    then
        .....
        .....
    else
        .....
    fi
else
    .....
fi
```

4) ladder -if:

```
if [condition]
then
    action-1
elif [ condition ]
then
    action-2
elif [ condition ]
then
    action-3
```



```
else
    default action
fi
```

Q2) Write a Script to find Greater of 2 Numbers?

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

```
read -p "Enter First Number:" a
read -p "Enter Second Number:" b
```

```
if [ $a -gt $b ]
then
    echo "Greater Number is:$a"
else
    echo "Greater Number is:$b"
fi
```

```
durgasoft@durgasoft:~/scripts$ test.sh
Enter First Number:10
Enter Second Number:20
Greater Number is:20
durgasoft@durgasoft:~/scripts$ test.sh
Enter First Number:20
Enter Second Number:10
Greater Number is:20
durgasoft@durgasoft:~/scripts$ test.sh
Enter First Number:-10
Enter Second Number:-20
Greater Number is:-10
```

Q3) Write a Script to Check whether Numbers OR Equal OR Not. **If the Numbers are not Equal then Print Greater Number?**

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

```
read -p "Enter First Number:" a
read -p "Enter Second Number:" b
```

```
if [ $a -eq $b ]
then
    echo "Both Numbers are equal"
elif [ $a -gt $b ]
then
    echo "First Number is greater than Second Number"
else
    echo "First Number is less than Second Number"
fi
```



Q4) Write a Script to find Greater of 3 Numbers?

1st Way:

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

```
read -p "Enter First Number:" a
read -p "Enter Second Number:" b
read -p "Enter Third Number:" c

if [ $a -gt $b ]
then
    if [ $a -gt $c ]
    then
        echo "The Greater Number:$a"
    else
        echo "The Greater Number:$c"
    fi
elif [ $b -gt $c ]
then
    echo "The Greater Number:$b"
else
    echo "The Greater Number:$c"
fi
```

2nd Way:

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

```
read -p "Enter First Number:" a
read -p "Enter Second Number:" b
read -p "Enter Third Number:" c

if [ $a -gt $b -a $a -gt $c ]
then
    echo "The Greater Number: $a"
elif [ $b -gt $c ]
then
    echo "The Greater Number: $b"
else
    echo "The Greater Number: $c"
fi
```

WAS to check whether student is pass or fail based on given 3 subjects marks.
In any subject if marks less than 35 then status is fail.



1st Way:

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

```
read -p "Enter First Subject Marks:" a
read -p "Enter Second Subject Marks:" b
read -p "Enter Third Subject Marks:" c
```

```
if [ $a -lt 35 ]
then
    echo "Student Failed"
elif [ $b -lt 35 ]
then
    echo "Student Failed"
elif [ $c -lt 35 ]
then
    echo "Student Failed"
else
    echo "Result is Pass"
fi
```

2nd Way:

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

```
read -p "Enter First Subject Marks:" a
read -p "Enter Second Subject Marks:" b
read -p "Enter Third Subject Marks:" c
```

```
if [ $a -ge 35 -a $b -ge 35 -a $c -ge 35 ]
then
    echo "Result is Pass"
else
    echo "Result is Fail"
fi
```

The Funniest Example with if-else

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

```
read -p "Enter Your Favourite Brand:" brand
```

```
if [ $brand = "KF" ]
then
    echo "It is Childrens Brand"
```

```
elif [ $brand = "KO" ]
```



```
then
    echo "It is not that much Kick"

elif [ $brand = "RC" ]
then
    echo "It is too light"

elif [ $brand = "FO" ]
then
    echo "Buy One Get One FREE"
else
    echo "Other Brands are not recommended"
fi
```

```
durgasoft@durgasoft:~/scripts$ ./test.sh
Enter Your Favourite Brand:KF
It is Childrens Brand
durgasoft@durgasoft:~/scripts$ ./test.sh
Enter Your Favourite Brand:KO
It is not that much Kick
durgasoft@durgasoft:~/scripts$ ./test.sh
Enter Your Favourite Brand:RC
It is too light
durgasoft@durgasoft:~/scripts$ ./test.sh
Enter Your Favourite Brand:FO
Buy One Get One FREE
durgasoft@durgasoft:~/scripts$
durgasoft@durgasoft:~/scripts$ ./test.sh
Enter Your Favourite Brand:KALYANI
Other Brands are not recommended
```

Exit Codes/ Status Codes:

Every command and script return some value after execution, which indicates that whether it is successfully executed or not. This return value is called exit code or status code.

We can find exit code by using "\$?".

zero means command/script executed successfully.
non-zero means command/script not executed successfully.

```
durgasoft@durgasoft:~/scripts$ echo "Hello Friends"
Hello Friends
durgasoft@durgasoft:~/scripts$ echo "$?"
0
```



```
durgasoft@durgasoft:~/scripts$ rm sdfjkslajfdksajfjdsakjfdksajfkd
rm: cannot remove 'sdfjkslajfdksajfjdsakjfdksajfkd': No such file or
directory
durgasoft@durgasoft:~/scripts$ echo "$?"
1
```

exit Command:

In the script to stop execution in the middle, we can use exit command.

Syntax:

exit status_code

The allowed values for status_code are 0 to 255.

256 → 0

257 → 1

258 → 2

259 → 3

....

Q5) Write a Script that takes 2 Integer Numbers as Command Line Arguments and Prints Sum

If Number of Arguments is not 2, then we have to get Message saying "You should provide only 2 Arguments"

If the Arguments are not Integer Numbers then we have to get Message saying "You should provide Integer Numbers only"

test.sh

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

```
if [ $# -ne 2 ]
then
    echo "You Should provide exactly two arguments"
    exit 1
fi
x=$1
y=$2
sum=`expr $x + $y`
if [ $? -ne 0 ]
then
    echo "You should provide integer numbers only"
    exit 2
else
    echo "The Sum:$sum"
fi
```

**Output:**

```
durgasoft@durgasoft:~/scripts$ test.sh 10 20
The Sum:30
durgasoft@durgasoft:~/scripts$ test.sh
You Should provide exactly two arguments
durgasoft@durgasoft:~/scripts$ test.sh 10
You Should provide exactly two arguments
durgasoft@durgasoft:~/scripts$ test.sh 10 20 30
You Should provide exactly two arguments
durgasoft@durgasoft:~/scripts$ test.sh 10 durga
expr: non-integer argument
You should provide integer numbers only
```

Q6) Write a Script that Reads an Integer Number and Check whether the given Number is +ve Number OR -ve Number?

```
#!/bin/bash

read -p "Enter integer number to check:" n
if [ $n -gt 0 ]; then
    echo "$n is +ve number"
else
    echo "$n is -ve number"
fi
```

Output:

```
durgasoft@durgasoft:~/scripts$ test.sh
Enter integer number to check:10
10 is +ve number
durgasoft@durgasoft:~/scripts$ test.sh
Enter integer number to check:-10
-10 is -ve number
```

Note: If we want to take then in the same line then we should use ;

if [\$n -gt 0]; then
The advantage is we can reduce the number of lines.

Q7) Write a Script that Reads an Integer Number and Checks whether it is Even Number OR not?

```
#!/bin/bash

read -p "Enter integer number to check:" n
if [ ${n%2} -eq 0 ]; then
    echo "$n is even number"
```



```
else
    echo "$n is odd number"
fi
```

Output:

```
durgasoft@durgasoft:~/scripts$ test.sh
Enter integer number to check:10
10 is even number
durgasoft@durgasoft:~/scripts$ test.sh
Enter integer number to check:15
15 is odd number
```

Q8) Write a Script that Reads an Integer Number and Checks whether it is 3 Digit Number OR not?

```
#!/bin/bash

read -p "Enter integer number to check:" n

if [ $n -ge 100 -a $n -le 999 ]; then
    echo "$n is 3-digit number"
else
    echo "$n is not 3-digit number"
fi
```

Output:

```
durgasoft@durgasoft:~/scripts$ test.sh
Enter integer number to check:123
123 is 3-digit number
durgasoft@durgasoft:~/scripts$ test.sh
Enter integer number to check:45
45 is not 3-digit number
durgasoft@durgasoft:~/scripts$ test.sh
Enter integer number to check:1234
1234 is not 3-digit number
```

File Test Options:

- e → Returns true if file/directory exists
- s → Returns true if the file is non-empty
- f → Returns true if the file is a regular file
- d → Returns true if the file is a directory
- l → Returns true if the file is link file
- b → Returns true if the file is block special file
- c → Returns true if the file is character special file



-r → Returns true if current user has read permission on the file
-w → Returns true if current user has write permission on the file
-x → Returns true if current user has execute permission on the file
-o → Returns true if current user is owner of the file.
file1 -ot file2 → Returns true if file1 is older than file2 (last modified time)
file1 -nt file2 → Returns true if file1 is newer than file2 (last modified time)

Eg 1: Script to Test whether the given File Exists OR not?

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

```
read -p "Enter File Name to test:" fname
```

```
if [ -e $fname ]
then
    echo "$fname exists"
else
    echo "$fname not exists"
fi
```

Eg 2: Script to Test whether the given File is Regular File OR Directory?

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

```
read -p "Enter File Name to test:" fname
```

```
if [ -e $fname ]; then
    if [ -f $fname ]; then
        echo "It is regular file"
    elif [ -d $fname ]; then
        echo "It is Directory file"
    else
        echo "It is special file"
    fi
else
    echo "$fname does not exist"
fi
```

Output:

```
durgasoft@durgasoft:~/scripts$ test.sh
Enter File Name to test:a.txt
It is regular file
durgasoft@durgasoft:~/scripts$ test.sh
Enter File Name to test:dir1
It is Directory file
```



```
durgasoft@durgasoft:~/scripts$ test.sh
Enter File Name to test:dddddd
dddddd does not exist
durgasoft@durgasoft:~/scripts$ test.sh
Enter File Name to test:/bin
It is Directory file
durgasoft@durgasoft:~/scripts$ test.sh
Enter File Name to test:/dev/vcsa3
It is special file
```

Write a Script that Reads a File Name and Display its Content to the Terminal?

```
#!/bin/bash

read -p "Enter File Name to test:" fname

if [ -e $fname ]; then
    if [ -f $fname ]; then
        if [ -r $fname ]; then
            cat $fname
        else
            echo "User not having Read permission"
        fi
    else
        echo "It is not a regular file"
    fi
else
    echo "$fname does not exist"
fi
```

Q9) Write a Script that Reads File Name and Check whether it is Empty File OR not?

```
#!/bin/bash

read -p "Enter File Name to test:" fname

if [ -e $fname ]; then
    if [ -f $fname ]; then
        if [ -s $fname ]; then
            echo "$fname is not empty file"
        else
            echo "$fname is empty file"
        fi
    fi
fi
```



```
    else
        echo "It is not a regular file"
    fi
else
    echo "$fname does not exist"
fi
```

Q10) Write a Script that Accepts 2 File Names and Check whether both Files having same Content OR not?

```
#!/bin/bash

read -p "Enter First File Name: " fname1
read -p "Enter Second File Name: " fname2

result=$(cmp $fname1 $fname2)
if [ -z "$result" ]; then
    echo "The given 2 files having same content"
else
    echo "The given 2 files having different content"
fi
```

Note:

-z is string comparison option.
returns True if the string is empty.

Q11) Write a Script that Accepts a File Name and Display User Permissions?

```
#!/bin/bash

read -p "Enter First File Name: " fname

READ=NO
WRITE=NO
EXECUTE=NO

if [ -r $fname ]; then
    READ=YES
fi

if [ -w $fname ]; then
    WRITE=YES
fi

if [ -x $fname ]; then
    EXECUTE=YES
fi
```



```
if [ -x $fname ]; then
    EXECUTE=YES
fi
```

```
echo "User Permissions Summary"
echo "-----"
echo "Read Permission: $READ"
echo "Write Permission: $WRITE"
echo "Execute Permission: $EXECUTE"
```

Output:

```
durgasoft@durgasoft:~/scripts$ test.sh
Enter First File Name: a.txt
```

User Permissions Summary

```
Read Permission: YES
Write Permission: YES
Execute Permission: NO
```

Q12) Write a Script that Reads File Name and Remove the specified File?

```
#!/bin/bash

read -p "Enter file/directory name to delete:" fname
if [ -e $fname ]; then
    rm -r $fname
    echo "$fname removed successfully"
else
    echo "$fname does not exist"
fi
```

Mini Application:

Copy all files and directories present in the first directory to the second directory. We should create compressed tar file and have to move that tar file. After moving tar file to the second directory, extract all files and directories and remove that tar file.

```
copy /home/durgasoft/x /home/durgasoft/y
```



Tests to Perform:

- 1) The number of command line arguments should be 2
- 2) The source and destination directories should be available already
- 3) The source and destination arguments should be directories
- 4) The user should have read and execute permissions on source directory
- 5) The user should have write and execute permissions on destination directory
- 6) All error messages should be sent to error file and the file name should contain timestamp.
- 7) All intermediate steps should be displayed to the terminal.

String Test Options:

- 1) `str1 = str2` → Returns true if both strings are same
- 2) `str1 != str2` → Returns true if both strings are different
- 3) `-z str` → Returns true if the string is empty
- 4) `-n str` → Returns true if the string is not empty
- 5) `str1 > str2` → Returns true if the str1 is alphabetically greater than str2
- 5) `str1 < str2` → Returns true if the str1 is alphabetically less than str2

Demo Program For String Comparison:

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

```
read -p "Enter First String:" str1
read -p "Enter Second String:" str2
```

```
if [ $str1 = $str2 ]; then
    echo "Both strings are equal"

elif [ $str1 \< $str2 ]; then
    echo "$str1 is less than $str2"
else
    echo "$str1 is greater than $str2"
fi
```

Note:

```
elif [ $str1 \< $str2 ]; then
```

If we are not using \ symbol then < acts as input redirection operator. To consider < as symbol only we should use \.

```
durgasoft@durgasoft:~/scripts$ test.sh
Enter First String:Durga
Enter Second String:Durga
Both strings are equal
durgasoft@durgasoft:~/scripts$ test.sh
```



```
Enter First String:Apple
Enter Second String:Banana
Apple is less than Banana
durgasoft@durgasoft:~/scripts$ test.sh
Enter First String:Banana
Enter Second String:Apple
Banana is greater than Apple
```

Q13) Write a Script that Checks whether Login User is Root OR not. If Login User is Root then Display 1st 5 Current Running Processes Information

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

```
user=$(whoami)
```

```
if [ $user != "root" ]; then
    echo "Current user not root user and hence exiting"
    exit 1
fi
```

```
echo "The Five Current Running Processes information"
echo "===== "
ps -ef | head -5
```

```
durgasoft@durgasoft:~/scripts$ test.sh
Current user not root user and hence exiting
durgasoft@durgasoft:~/scripts$ sudo -i
[sudo] password for durgasoft:
root@durgasoft:~# whoami
root
root@durgasoft:~# test.sh
test.sh: command not found
root@durgasoft:~# pwd
/root
root@durgasoft:~# /home/durgasoft/scripts/test.sh
```

The 5 Current Running Processes Information

UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	1	0	0	19:32	?	00:00:03	/sbin/init splash
root	2	0	0	19:32	?	00:00:00	[kthreadd]
root	3	2	0	19:32	?	00:00:00	[rcu_gp]
root	4	2	0	19:32	?	00:00:00	[rcu_par_gp]



Q14) Write a Script to Test whether the given String is Empty OR not? If it is not Empty then Print its Length

```
#!/bin/bash

read -p "Enter Any String to test:" str

if [ -z $str ]; then
    echo "Provided input string is empty string"
else
    echo "Provided input string is not empty string"
    echo "Its length is : $( echo -n $str | wc -c )"
fi
```

2) case Statement:

- If multiple options are available then it is not recommended to use nested if-else statements. It increases length of the code and reduces readability.
- To handle such type of requirements we should go for case statement.

Syntax:

case \$variable in

```
option1 )
    action-1
    ;;
option2 )
    action-2
    ;;
option3 )
    action-3
    ;;
* )
    default action
    ;;
esac
```

Note:

- 1) space is optional while defining options.
- 2) ;; can be used to come out of case statement.
- 3) ;; is mandatory for all options except for default option.
- 4) If we take default option at the beginning, then for any input the same default option will be executed.



Eg 1: Write a script to read a number from 0 to 9 and print equivalent English word?

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

```
read -p "Enter Any digit from 0 to 9:" n
```

```
case $n in
```

```
0) echo "ZERO"
```

```
;;
```

```
1) echo "ONE"
```

```
;;
```

```
2) echo "TWO"
```

```
;;
```

```
3) echo "THREE"
```

```
;;
```

```
4) echo "FOUR"
```

```
;;
```

```
5) echo "FIVE"
```

```
;;
```

```
6) echo "SIX"
```

```
;;
```

```
7) echo "SEVEN"
```

```
;;
```

```
8) echo "EIGHT"
```

```
;;
```

```
9) echo "NINE"
```

```
;;
```

```
*) echo "Please enter a digit from 0 to 9 only"
```

```
esac
```

Q15) Write a Script that Reads Favourite Brand and Prints corresponding Meaningful Message?

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

```
read -p "Enter Your Favourite Brand: " brand
```

```
case $brand in
```

```
"KF")
```

```
    echo "It is childrens brand"
```

```
;;
```

```
"KO")
```

```
    echo "It is not that much kick"
```

```
;;
```



```
"RC")
    echo "It is too light"
;;
"FO")
    echo "Buy one get one FREE"
;;
*)
    echo "Other brands are not recommended"
esac
```

Q16) Write a Script that Accepts a Single Character and Check whether the given Character is Alphabet OR Digit OR Special Character?

```
#!/bin/bash

read -p "Enter Any Character to check: " ch

case $ch in
    [a-zA-Z])
        echo "It is an Alphabet symbol"
        ;;
    [0-9])
        echo "It is a Digit"
        ;;
    [^a-zA-Z0-9])
        echo "It is a Special Symbol"
        ;;
    *)
        echo "Enter only one character"
esac
```

```
durgasoft@durgasoft:~/scripts$ test.sh
Enter Any Character to check: a
It is an Alphabet symbol
durgasoft@durgasoft:~/scripts$ test.sh
Enter Any Character to check: 8
It is a Digit
durgasoft@durgasoft:~/scripts$ test.sh
Enter Any Character to check: $
It is a Special Symbol
durgasoft@durgasoft:~/scripts$ test.sh
Enter Any Character to check: durga
Enter only one character
```



Q17) Write a Script that Accepts a Single Character and Checks whether it is Digit OR Special Character OR Vowel OR Consonent?

```
#!/bin/bash

read -p "Enter Any Character to check: " ch

case $ch in
    [^a-zA-Z0-9])
        echo "It is a Special Character"
        ;;
    [0-9])
        echo "It is a Digit"
        ;;
    [aeiouAEIOU])
        echo "It is a Vowel"
        ;;
    [^aeiouAEIOU])
        echo "It is a Consonent"
        ;;
    *)
        echo "Enter only one character"
esac
```

```
durgasoft@durgasoft:~/scripts$ test.sh
Enter Any Character to check: a
It is a Vowel
durgasoft@durgasoft:~/scripts$ test.sh
Enter Any Character to check: s
It is a Consonent
durgasoft@durgasoft:~/scripts$ test.sh
Enter Any Character to check: 7
It is a Digit
durgasoft@durgasoft:~/scripts$ test.sh
Enter Any Character to check: $
It is a Special Character
```

Q18) Write a Script that performs File Operations based on provided Option?

- A → Display Content
- B → Append Content
- C → Overwrite Content
- D → Delete Content



The file name is abc.txt

```
#!/bin/bash
echo "A → Display Content"
echo "B → Append Content"
echo "C → Overwrite Content"
echo "D → Delete Content"
read -p "Choose Your Option A|B|C|D: " option

case $option in
A)
    if [ ! -s "abc.txt" ]; then
        echo "It is an empty file"
    else
        echo "The content of the file is:"
        echo "-----"
        cat abc.txt
    fi
;;
B)
    read -p "Provide your data to append:" data
    echo $data >> abc.txt
    echo "Data Appended"
;;
C)
    read -p "Provide your data to overwrite:" data
    echo $data > abc.txt
    echo "Old data Overwritten with new data "
;;
D)
    cat /dev/null > abc.txt
    echo "Deleted the content of the file"
;;
*)
    echo "Choose only A|B|C|D. Execute Again"
esac
```

Q19) Write a Script that Reads 2 Integer Numbers and perform required Mathematical Operation based on provided Option?

- 1 → Addition Operation
- 2 → Subtraction Operation
- 3 → Multiplication Operation
- 4 → Division Operation



`#!/bin/bash`

```
read -p "Enter First Number: " n1
read -p "Enter Second Number: " n2
echo ""
echo "1 --> Addition Operation"
echo "2 --> Subtraction Operation"
echo "3 --> Multiplication Operation"
echo "4 --> Division Operation"
read -p "Choose Your Option 1|2|3|4: " option
```

```
case $option in
  1)
    echo "$n1 + $n2 = $((n1+n2))"
    ;;
  2)
    echo "$n1 - $n2 = $((n1-n2))"
    ;;
  3)
    echo "$n1 * $n2 = $((n1*n2))"
    ;;
  4)
    echo "$n1 / $n2 = $((n1/n2))"
    ;;
  *)
    echo "Choose only 1|2|3|4. Execute Again"
esac
```

Iterative Statements:

If we want to execute a group of commands multiple times then we should go for iterative statements.

There are 3 types of iterative statements

- 1) while loop
- 2) until loop
- 3) for loop



1) while Loop:

If we don't know the number of iterations in advance, then we should go for while loop.

Syntax:

```
while [ condition ]  
do  
    body  
done
```

As long as condition is true, then body will be executed. Once condition fails then only loop will be terminated.

Q1) Write a Script to Print Numbers from 1 to 10

```
#!/bin/bash  
  
i=1  
  
while [ $i -le 10 ]  
do  
    echo $i  
    let i++  
done
```

Q2) Write a Script to generate Numbers until Limit which is provided by End User?

```
#!/bin/bash  
  
read -p "Enter Limit:" n  
i=1  
  
while [ $i -le $n ]  
do  
    echo $i  
    sleep 2  
    let i++  
done
```

Note: If we don't want to perform any operation for a particular amount of time (i.e just pausing is required) then we should go for sleep command. The argument to the sleep command is seconds.

Eg:

sleep 2
sleep 3
sleep 0.5

Q3) Write a Script to find the Sum of First n Integers, where n is provided by End User?

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

```
read -p "Enter n value:" n
```

```
i=1
sum=0
while [ $i -le $n ]
do
    let sum=sum+i
    let i++
done
```

```
echo "The Sum of first $n numbers: $sum"
```

Output:

```
durgasoft@durgasoft:~/scripts$ test.sh
Enter n value:3
The Sum of first 3 numbers: 6
durgasoft@durgasoft:~/scripts$ test.sh
Enter n value:10
The Sum of first 10 numbers: 55
durgasoft@durgasoft:~/scripts$ test.sh
Enter n value:12345
The Sum of first 12345 numbers: 76205685
```

Q4) Write a Script to Display Timer (Digital Timer)?

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

```
while [ true ]  
do  
    clear  
    printf "\n\n\n\n\n\n\t\t\t\t$(date +%H:%M:%S)"  
    sleep 1  
done
```



Note: To use escape characters like \n and \t, we should not use echo and we should use printf command.

Note: true and false are keywords which represents boolean values.

break Statement:

Based on some condition, if we want to break loop execution (i.e to come out of loop) then we should go for break statement.

Eg 1:

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

```
i=1
```

```
while [ $i -le 10 ]
do
    if [ $i -eq 5 ]; then
        break
    fi
    echo $i
    let i++
done
```

Output:

```
1
2
3
4
```

Eg 2:

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

```
while [ true ]
do
    clear
    printf "\n\n\n\n\n\t\t\t\t\t$(date +%H:%M:%S)"
    sleep 1
    h=$(date +%H)
    m=$(date +%M)
    if [ $h -eq 20 -a $m -eq 35 ]; then
        break
    fi
done
```



continue Statement:

We can use continue statement to skip current iteration and continue for the next iteration.

Eg:

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

```
i=0
```

```
while [ $i -lt 10 ]
do
    let i++
    if [ ${i%2} -eq 0 ]; then
        continue
    fi
    echo $i
done
```

Output:

```
1
3
5
7
9
```

Write a Script to Read File Name and Display its Content?

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

```
while [ true ]
do
    read -p "Enter File Name to Display content: " fname

    # Checking whether the file exists or not and whether regular file or not
    if [ -f $fname ]; then
        echo "The content of $fname : "
        echo "-----"
        cat $fname
    else
        echo "$fname does not exist"
    fi

    read -p "Do you want to display content of another file [Yes|No]: " option
    case $option in
        [yY]|[Yy][eE][sS])
```



```
        continue
        ;;
    [nN]|[nN][oO])
        break
        ;;
    esac
done
echo "Thanks for using application"
```

Output:

```
durgasoft@durgasoft:~/scripts$ test.sh
Enter File Name to Display content: abc.txt
```

The content of abc.txt:

```
Java
Python
Do you want to display content of another file [Yes|No]:y
Enter File Name to Display content: a.txt
```

The content of a.txt:

```
Tue Dec 10 17:00:48 IST 2019
Do you want to display content of another file [Yes|No]:YES
Enter File Name to Display content: b.txt
```

The content of b.txt:

```
December 2019
Su Mo Tu We Th Fr Sa
 1 2 3 4 5 6 7
 8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31
```

```
Do you want to display content of another file [Yes|No]:Y
Enter File Name to Display content: durga.txt
durga.txt does not exist
Do you want to display content of another file [Yes|No]:N
Thanks for using application
```



Write a Script that Reads a String as Input and find its Reverse?

Write a Script that performs Reverse of given String?

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

```
read -p "Enter any string to find reverse: " str
```

```
len=$( echo -n $str | wc -c )  
output=""
```

```
while [ $len -gt 0 ]  
do  
    ch=$( echo -n $str | cut -c $len )  
    output=$output$ch  
    let len--  
done  
echo "The Original String: $str"  
echo "The Reversed String: $output"
```

Output:

```
durgasoft@durgasoft:~/scripts$ test.sh  
Enter any string to find reverse: durga  
The Original String: durga  
The Reversed String: agrud  
durgasoft@durgasoft:~/scripts$ test.sh  
Enter any string to find reverse: abc  
The Original String: abc  
The Reversed String: cba  
durgasoft@durgasoft:~/scripts$ test.sh  
Enter any string to find reverse: durgasoft  
The Original String: durgasoft  
The Reversed String: tfosagrud  
durgasoft@durgasoft:~/scripts$
```

Eg:

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

```
while [ true ]  
do  
    clear  
    tput cup 7 25  
    echo "WELCOME TO DURGASOFT"  
    sleep 2  
    clear  
    tput cup 7 25
```



```
echo "UNIX/LINUX CLASSES"  
sleep 2  
done
```

Note: tput cup 7 25

It moves the cursor position to 7th row and 25th column.

Q5) Write a Script to Read Employee Data and Insert into emp.txt File?

```
#!/bin/bash  
  
while [ true ]  
do  
    read -p "Employee Number: " eno  
    read -p "Employee Name: " ename  
    read -p "Employee Salary: " esal  
    read -p "Employee Address: " eaddr  
    echo "$eno:$ename:$esal:$eaddr" >> emp.txt  
    echo "Employee Record Inserted Successfully"  
    read -p "Do you want to insert one more record [Yes|No]: " option  
    case $option in  
        [yY]|[Yy][eE][sS])  
            continue  
            ;;  
        [nN]|[nN][oO])  
            break  
            ;;  
        *)  
            ;;  
    esac  
done  
echo "Open emp.txt to see all employees information"
```

Output:

```
durgasoft@durgasoft:~/scripts$ test.sh  
Employee Number: 100  
Employee Name: Sunny  
Employee Salary: 1000  
Employee Address: Hyderabad  
Employee Record Inserted Successfully  
Do you want to insert one more record [Yes|No]: y  
Employee Number: 200  
Employee Name: Bunny  
Employee Salary: 2000  
Employee Address: Mumbai  
Employee Record Inserted Successfully
```



Do you want to insert one more record [Yes|No]: Yes
Employee Number: 300
Employee Name: Chinny
Employee Salary: 3000
Employee Address: Hyderabad
Employee Record Inserted Successfully
Do you want to insert one more record [Yes|No]: YES
Employee Number: 400
Employee Name: Vinny
Employee Salary: 4000
Employee Address: Chennai
Employee Record Inserted Successfully
Do you want to insert one more record [Yes|No]: N
Open emp.txt to see all employees information

Q6) Write a Script to implement cat Command with -n Option?

```
durgasoft@durgasoft:~/scripts$ cat -n emp.txt
```

```
1 100:Sunny:1000:Hyderabad
2 200:Bunny:2000:Mumbai
3 300:Chinny:3000:Hyderabad
4 400:Vinny:4000:Chennai
```

Syntax -1: To read data from the file by using while loop

```
cat emp.txt |
while read line
do
    do something with that line
done
```

Syntax -2: To read data from the file by using while loop

```
while read line
do
    do something with that line
done < emp.txt
```

test.sh

```
#!/bin/bash

fname=$1
if [ ! -f $fname ]; then
    echo "Please provide already existing regular file only"
    exit 1
fi
```



```
count=1
while read line
do
    echo " $count $line"
    let count++
done < $fname
```

Output

```
durgasoft@durgasoft:~/scripts$ test.sh emp.txt
1 100:Sunny:1000:Hyderabad
2 200:Bunny:2000:Mumbai
3 300:Chinny:3000:Hyderabad
4 400:Vinny:4000:Chennai
durgasoft@durgasoft:~/scripts$ test.sh abcd.txt
Please provide already existing regular file only
```

2) until Loop:

It is opposite to while loop.

Syntax:

```
until [ condition ]
do
    body
done
```

The body will be executed as long as condition returns false. Once condition returns true, then loop will be terminated.

Q7) Write a Script to Print 1 to 5 by using until Loop?

```
#!/bin/bash
i=1
until [ $i -gt 5 ]
do
    echo $i
    let i++
done
```

```
durgasoft@durgasoft:~/scripts$ test.sh
1
2
3
4
5
```



3) for Loop:

If we want to perform some action for every item in the given list, then we should go for for loop. It is similar to Java's for-each loop.

Syntax:

```
for variable in item1 item2 item3... itemN
do
    action
done
```

Eg 1: To print numbers from 1 to 5.

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

```
for i in 1 2 3 4 5
do
    echo "Current Number: $i"
done
```

Output:

```
durgasoft@durgasoft:~/scripts$ test.sh
Current Number: 1
Current Number: 2
Current Number: 3
Current Number: 4
Current Number: 5
```

Eg 2:

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

```
for course in java unix python testing datascience
do
    echo "Course Name: $course"
done
```

Output:

```
durgasoft@durgasoft:~/scripts$ test.sh
Course Name: java
Course Name: unix
Course Name: python
Course Name: testing
Course Name: datascience
```



Q8) Write a Script that Display Numbers from 1 to 100, which are divisible by 10.

```
#!/bin/bash

count=0
for num in {1..100}
do
    if [ $( num%10 ) -eq 0 ]; then
        echo "$num"
        let count++
    fi
done
echo "The number of values: $count"
```

Output:

```
durgasoft@durgasoft:~/scripts$ test.sh
10
20
30
40
50
60
70
80
90
100
The number of values: 10
```

Q9) Write a Script to Display all File Names Present in Current Working Directory?

```
#!/bin/bash

for name in *
do
    if [ -f $name ]; then
        echo $name
    fi
done
```



Q10) Write a Script to append Current Date and Time, Current Month Calender in every .txt File Present in Current Working Directory?

```
#!/bin/bash

for fname in *.txt
do
    date >> $fname
    cal >> $fname
done
echo "Task Completed"
```

Q11) Write a Script that Print all Command Line Arguments?

```
#!/bin/bash

if [ $# -ne 0 ]; then

    count=1
    for arg in $@
    do
        echo "Command Line Argument - $count: $arg"
        let count++
    done
else
    echo "Command line arguments are not passed"
fi
```

Output:

```
durgasoft@durgasoft:~/scripts$ test.sh
Command line arguments are not passed
durgasoft@durgasoft:~/scripts$ test.sh 10 20 30 40
Command Line Argument - 1: 10
Command Line Argument - 2: 20
Command Line Argument - 3: 30
Command Line Argument - 4: 40
durgasoft@durgasoft:~/scripts$ test.sh learning linux is very easy
Command Line Argument - 1: learning
Command Line Argument - 2: linux
Command Line Argument - 3: is
Command Line Argument - 4: very
Command Line Argument - 5: easy
```



Case Study:

emp.txt:

```
100:sunny:1000:Hyderabad
200:bunny:2000:Chennai
300:chinny:3000:Hyderabad
400:vinny:4000:Delhi
500:pinny:5000:Hyderabad
```

1) Write a Script to Display all Employees Information where Salary is Greater than 2500

```
#!/bin/bash
for record in $(cat emp.txt)
do
    sal=$(echo $record | cut -d ":" -f 3)
    if [ $sal -gt 2500 ]; then
        echo $record
    fi
done
```

2) Write a Script to Save all Employees Information where Salary is Greater than 2500 and City is Hyderabad to hyd.txt

```
#!/bin/bash

for record in $(cat emp.txt)
do
    sal=$(echo $record | cut -d ":" -f 3)
    city=$(echo $record | cut -d ":" -f 4)
    if [ $sal -gt 2500 -a $city = "Hyderabad" ]; then
        echo $record >> hyd.txt
    fi
done
echo "Task Completed"
```

3) Write a Script to Display Minimum and Maximum Salaries?

```
#!/bin/bash
max=$(cat emp.txt | head -1 | cut -d ":" -f 3)
min=$(cat emp.txt | head -1 | cut -d ":" -f 3)
max_record=$(cat emp.txt | head -1)
min_record=$(cat emp.txt | head -1)
for record in $(cat emp.txt)
do
    sal=$(echo $record | cut -d ":" -f 3)
```



```
if [ $sal -gt $max ]; then
    max=$sal
    max_record=$record
fi
if [ $sal -lt $min ]; then
    min=$sal
    min_record=$record
fi
done
echo "The Maximum Salary:$max"
echo "The Maximum Salaried Employeeed Information:"
echo "Employee No:$(echo $max_record | cut -d ":" -f 1)"
echo "Employee Name:$(echo $max_record | cut -d ":" -f 2)"
echo "Employee Salary:$(echo $max_record | cut -d ":" -f 3)"
echo "Employee Address:$(echo $max_record | cut -d ":" -f 4)"
echo
echo "The Minimum Salary:$min"
echo "The Minimum Salaried Employeeed Information:"
echo "Employee No:$(echo $min_record | cut -d ":" -f 1)"
echo "Employee Name:$(echo $min_record | cut -d ":" -f 2)"
echo "Employee Salary:$(echo $min_record | cut -d ":" -f 3)"
echo "Employee Address:$(echo $min_record | cut -d ":" -f 4)"
```

Q12) Write a Script to Display Multiple Files Content to the Terminal and all File Names passed as Command Line Arguments?

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

```
if [ $# -ne 0 ]; then
    for fname in $@
    do
        if [ -f $fname ]; then
            echo "$fname content:"
            echo "======"
            cat $fname
        else
            echo "$fname does not exist or it is not a regular file"
        fi
    done
else
    echo "Please pass atleast one file name"
fi
```



Q13) Write a Script to append Multiple Files Content to a Single File result.txt. File Names are passed as Command Line Arguments?

```
#!/bin/bash

if [ $# -ne 0 ]; then
    for fname in $@
    do
        if [ -f $fname ]; then
            cat $fname >> result.txt
        else
            echo "$fname does not exist or it is not a regular file"
        fi
    done
else
    echo "Please pass atleast one file name"
fi
```

Alternative Syntax of for Loop (Advanced for Loop):

Old Style of for Loop:

```
for variable in item1 item2 ... itemN
do
    body
done
```

Advanced for Loop:

```
for ((i=1; i<10; i++))
do
    body
done
```

Q14) Write a Script to Print Numbers from 0 to 4 by using advanced for Loop?

```
#!/bin/bash
for ((i=0; i<5; i++))
do
    echo $i
done
```



Output:

0
1
2
3
4

Q15) Write a Script to Print Numbers for Count Down from provided Number to 1 by using advanced for Loop?

```
#!/bin/bash
read -p "Enter n value:" n
for((i=n,i>=1;i--))
do
    echo $i
    sleep 1.5
done
```

Q16) Write a Script to Display nth Table?

```
#!/bin/bash
read -p "Enter n value:" n
for ((i=1; i<=10; i++))
do
    echo "$n * $i = ${n*i}"
done
```

durga@durga-VirtualBox:~/scripts\$ test.sh

Enter n value:9

9 * 1 = 9
9 * 2 = 18
9 * 3 = 27
9 * 4 = 36
9 * 5 = 45
9 * 6 = 54
9 * 7 = 63
9 * 8 = 72
9 * 9 = 81
9 * 10 = 90



Q17) Write a Script to generate Hotel Bill based on Customer selected Items. The Items and Price Information is as follows

A → Vadapov (Each Plate Rs 30 /-)

B → Dosa (Each Plate Rs 50 /-)

C → Poori (Each Plate Rs 40 /-)

D → Idli (Each Plate Rs 25 /-)

```
#!/bin/bash
echo "Welcome to DURGA HOTEL"
echo
amount=0
while [ true ]
do
    echo "Menu Items:"
    echo "....."
    echo "A --->Vadapov (Each Plate Rs 30 /-)"
    echo "B --->Dosa (Each Plate Rs 50 /-)"
    echo "C --->Poori (Each Plate Rs 40 /-)"
    echo "D --->Idli (Each Plate Rs 25 /-)"
    read -p "Choose Your Required Item A|B|C|D:" item
    case $item in
        A)
            read -p "Enter the number of plates of Vadapov, you required:" quantity
            amount=$((amount+quantity*30))
            ;;
        B)
            read -p "Enter the number of plates of Dosa, you required:" quantity
            amount=$((amount+quantity*50))
            ;;
        C)
            read -p "Enter the number of plates of Poori, you required:" quantity
            amount=$((amount+quantity*40))
            ;;
        D)
            read -p "Enter the number of plates of Idli, you required:" quantity
            amount=$((amount+quantity*25))
            ;;
        *)
            echo "You entered invalid option. Choose Again"
            continue
    esac
    read -p "Do you want to order any other item[Yes|No]:" option
    case $option in
```



```
[Yy]|[Yy][Ee][Ss])
    continue
    ;;
[Nn]|[Nn][Oo])
    break
    ;;
esac
done
echo
echo "Your Total Bill Amount: Rs $amount/-"
echo "Thanks for visiting DURGA HOTEL"
```

durga@durga-VirtualBox:~/scripts\$ test.sh
Welcome to DURGA HOTEL

Menu Items:
.....
A → Vadapov (Each Plate Rs 30 /-)
B → Dosa (Each Plate Rs 50 /-)
C → Poori (Each Plate Rs 40 /-)
D → Idli (Each Plate Rs 25 /-)
Choose Your Required Item A|B|C|D:D
Enter the number of plates of Idli, you required:4
Do you want to order any other item[Yes|No]:yes

Menu Items:
.....
A → Vadapov (Each Plate Rs 30 /-)
B → Dosa (Each Plate Rs 50 /-)
C → Poori (Each Plate Rs 40 /-)
D → Idli (Each Plate Rs 25 /-)
Choose Your Required Item A|B|C|D:C
Enter the number of plates of Poori, you required:3
Do you want to order any other item[Yes|No]:Yes

Menu Items:
.....
A → Vadapov (Each Plate Rs 30 /-)
B → Dosa (Each Plate Rs 50 /-)
C → Poori (Each Plate Rs 40 /-)
D → Idli (Each Plate Rs 25 /-)
Choose Your Required Item A|B|C|D:B
Enter the number of plates of Dosa, you required:2
Do you want to order any other item[Yes|No]:Yes



Menu Items:

.....

A → Vadapov (Each Plate Rs 30 /-)

B → Dosa (Each Plate Rs 50 /-)

C → Poori (Each Plate Rs 40 /-)

D → Idli (Each Plate Rs 25 /-)

Choose Your Required Item A|B|C|D:A

Enter the number of plates of Vadapov, you required:1

Do you want to order any other item[Yes|No]:No

Your Total Bill Amount: Rs 350/-

Thanks for visiting DURGA HOTEL

Q18) Write a Script to Test whether the given Number is Prime Number OR not?

If any number has two factors (1 and itself), such type of number is said to be prime number.

2 → 1, 2

3 → 1, 3

5 → 1, 5

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

```
read -p "Enter Any Number to test whether it is prime or not:" n
```

```
if [ $n -le 1 ]; then
```

```
    echo "$n is not a PRIME number"
```

```
    exit 1
```

```
fi
```

```
for ((i=2;i<=n/2;i++))
```

```
do
```

```
    if [ $(n%i) -eq 0 ]; then
```

```
        echo "$n is not PRIME number"
```

```
        exit 1
```

```
    fi
```

```
done
```

```
echo "$n is a PRIME number"
```

```
durga@durga-VirtualBox:~/scripts$ test.sh
```

```
Enter Any Number to test whether it is prime or not:23
```

```
23 is PRIME number
```

```
durga@durga-VirtualBox:~/scripts$ test.sh
```

```
Enter Any Number to test whether it is prime or not:29
```

```
29 is PRIME number
```

```
durga@durga-VirtualBox:~/scripts$ test.sh
```

```
Enter Any Number to test whether it is prime or not:35
```

```
35 is not PRIME number
```



Topic-32: Arrays

If we want to represent a group of values with a single name then we should go for arrays concept.

How to Create Arrays:

1. If we know elements at the beginning:

```
courses=(Java Python Linux Django)
```

2. If we don't know elements at the beginning:

```
courses[0]=Java  
courses[1]=Python  
courses[2]=Linux  
courses[3]=Django
```

The index values need not be consecutive. We can take randomly.

```
courses[10]=DataScience  
courses[20]=Devops
```

How to Access Elements:

We can access array elements by using index which is zero based. i.e the index of first element is zero.

```
${courses[0]} → First element  
${courses[1]} → Second element  
${courses[@]} → All elements present inside array.  
${courses[*]} → All elements present inside array into a single string separated by first  
character in IFS (Internal Field Separator)  
${!courses[@]} → It returns all indexes where elements are available.  
${#courses[@]} → It returns the number of elements present inside array.  
${#courses[0]} → It returns the length of first element.
```

Demo Script:

```
#!/bin/bash  
courses[0]=Java  
courses[1]=Python  
courses[2]=Linux  
courses[3]=Django  
courses[10]=DataScience
```



```
courses[20]=Devops
echo "First Element : ${courses[0]}"
echo "Second Element : ${courses[1]}"
echo "All Elements with @ : ${courses[@]}"
echo "All elements with * : ${courses[*]}"
echo "The indices where elements are available : ${!courses[@]}"
echo "The total number of elements : ${#courses[@]}"
echo "The length of first element : ${#courses[0]}"
```

Output:

```
durgasoft@durgasoft:~/scripts$ test.sh
First Element : Java
Second Element : Python
All Elements with @ : Java Python Linux Django DataScience Devops
All elements with * : Java Python Linux Django DataScience Devops
The indices where elements are available : 0 1 2 3 10 20
The total number of elements : 6
The length of first element : 4
```

Q1) Write a Script to Create an Array with some Elements and Print all Elements by using while Loop, for Loop and advanced for Loop?

```
#!/bin/bash

declare -a fruits
fruits=("Apple" "Orange" "Banana" "Mango")
size=${#fruits[@]}
i=0

echo "All elements by using while loop:"
while [ $i -lt $size ]
do
    echo ${fruits[$i]}
    let i++
done

echo "All elements by using for loop:"
for fruit in ${fruits[@]}
do
    echo $fruit
done

echo "All elements by using advanced for loop:"
for (( i=0; i < ${#fruits[@]}; i++ ))
```



```
do
    echo ${fruits[$i]}
done
```

Output:

durga@durga-VirtualBox:~/scripts\$ test.sh

All elements by using while loop:

```
Apple
Orange
Banana
Mango
```

All elements by using for loop:

```
Apple
Orange
Banana
Mango
```

All elements by using advanced for loop:

```
Apple
Orange
Banana
Mango
```

Note:

- 1) If we create an array with elements directly
fruits=("Apple" "Orange" "Banana" "Mango")
then the indices will be 0,1,2,3 etc
- 2) After creating array we can add extra elements also
fruits=("Apple" "Orange" "Banana" "Mango")
fruits[4]="Sapota"

Q2) Write a Script for accessing Array Elements by using for Loop if Indices are Random?

```
#!/bin/bash
declare -a fruits
fruits[10]="Apple"
fruits[20]="Banana"
fruits[30]="Orange"
fruits[40]="Mango"
```

```
echo "Accessing based on Values:"
for fruit in ${fruits[@]}
do
```



```
echo $fruit
done
echo
echo "Accessing based on Index:"
for index in ${!fruits[@]}
do
    echo ${fruits[$index]}
done
```

Output:

durga@durga-VirtualBox:~/scripts\$ test.sh

Accessing based on Values:

Apple
Banana
Orange
Mango

Accessing based on Index:

Apple
Banana
Orange
Mango

Q3) Is it Possible to Remove Array Elements?

Yes possible by using unset command.

Eg:

```
#!/bin/bash
declare -a fruits
fruits[10]="Apple"
fruits[20]="Banana"
fruits[30]="Orange"
fruits[40]="Mango"
```

```
echo "All Array Elements Before Removal: ${fruits[@]}"
unset fruits[20]
unset fruits[40]
echo "All Array Elements After Removal: ${fruits[@]}"
```

Output:

durga@durga-VirtualBox:~/scripts\$ test.sh

All Array Elements Before Removal: Apple Banana Orange Mango

All Array Elements After Removal: Apple Orange



Q4) Write a Script to Store given n Numbers in to an Array?

```
#!/bin/bash
read -p "Enter The Number Of values:" n

for ((i=0,j=1;i<n;i++))
do
    read -p "Enter The Number-${j++}:" NUM[${i}]
done
```

Output:

```
durga@durga-VirtualBox:~/scripts$ test.sh
Enter The Number Of values:3
Enter The Number-1:10
Enter The Number-2:20
Enter The Number-3:30
```

Q5) Write a Script to Read n Numbers and Store inside Array. Print the Sum of Even Numbers and Odd Numbers separately?

```
#!/bin/bash
read -p "Enter Number of Values:" n

for((i=0,j=1;i<n;i++))
do
    read -p "Enter The Number-${j++}:" NUM[${i}]
done

esum=0
osum=0
for val in ${NUM[@]}
do
    if [ ${val%2} -eq 0 ]; then
        let esum=esum+val
    else
        let osum=osum+val
    fi
done
echo "The Sum of Even Numbers: $esum"
echo "The Sum of Odd Numbers: $osum"
```



Output:

durga@durga-VirtualBox:~/scripts\$ test.sh

Enter Number of Values:5

Enter The Number-1:10

Enter The Number-2:12

Enter The Number-3:13

Enter The Number-4:15

Enter The Number-5:20

The Sum of Even Numbers: 42

The Sum of Odd Numbers: 28

Q6) Write a Script to Store all .txt File Names Present in Current Working Directory in to an Array and Print Permissions of every File

```
#!/bin/bash
files=( $(ls *.txt) )
for fname in ${files[@]}
do
    echo -ne "$fname:\t"
    if [ -r $fname ]; then
        echo -ne "READ(Y)\t"
    else
        echo -ne "READ(N)\t"
    fi
    if [ -w $fname ]; then
        echo -ne "WRITE(Y)\t"
    else
        echo -ne "WRITE(N)\t"
    fi
    if [ -x $fname ]; then
        echo "EXECUTE(Y)"
    else
        echo "EXECUTE(N)"
    fi
done
```



Output:

`durga@durga-VirtualBox:~/scripts$ test.sh`

<code>abcd.txt:</code>	<code>READ(Y)</code>	<code>WRITE(Y)</code>	<code>EXECUTE(N)</code>
<code>abc.txt:</code>	<code>READ(Y)</code>	<code>WRITE(Y)</code>	<code>EXECUTE(N)</code>
<code>a.txt:</code>	<code>READ(Y)</code>	<code>WRITE(Y)</code>	<code>EXECUTE(N)</code>
<code>b.txt:</code>	<code>READ(Y)</code>	<code>WRITE(Y)</code>	<code>EXECUTE(N)</code>
<code>c.txt:</code>	<code>READ(Y)</code>	<code>WRITE(Y)</code>	<code>EXECUTE(N)</code>
<code>emp.txt:</code>	<code>READ(Y)</code>	<code>WRITE(Y)</code>	<code>EXECUTE(N)</code>
<code>hyd.txt:</code>	<code>READ(Y)</code>	<code>WRITE(Y)</code>	<code>EXECUTE(N)</code>
<code>output.txt:</code>	<code>READ(Y)</code>	<code>WRITE(Y)</code>	<code>EXECUTE(N)</code>
<code>result.txt:</code>	<code>READ(Y)</code>	<code>WRITE(Y)</code>	<code>EXECUTE(N)</code>
<code>z.txt:</code>	<code>READ(N)</code>	<code>WRITE(N)</code>	<code>EXECUTE(N)</code>



Topic-33: Shell Script Functions

If any group of commands are repeatedly required, then it is not recommended to write these commands separately everytime. It increases length of the code and reduces readability.

Such type of repeated code we have to define inside a block and we can call that block where ever required. This block of commands is nothing but function.

Hence function is nothing but a block of repeatable commands.

Advantages of Functions:

- 1) It reduces length of the code.
- 2) It improves readability.
- 3) It improves maintainability.
- 4) It promotes DRY principle.
 DRY → Don't Repeat Yourself.

How to define a Function?

1st Way:

```
function function_name()
{
    commands
}
```

2nd Way:

```
function_name()
{
    commands
}
```



How to call a Function:

function_name param1 param2 param3 ...

Q1) Write a Function to Display wish Message?

```
#!/bin/bash

#defining a function
wish()
{
    echo "Hello Friends... Good Evening"
}

wish # calling a function
wish
wish
```

Output:

```
durgasoft@durgasoft:~/scripts$ test.sh
Hello Friends... Good Evening
Hello Friends... Good Evening
Hello Friends... Good Evening
```

Note: Before calling a function, it should be defined.

Eg 2:

```
#!/bin/bash

f1()
{
    echo "I am in f1 function"
}
f2()
{
    echo "I am in f2 function"
    f1
}
f1
f2
```



Output:

```
durgasoft@durgasoft:~/scripts$ test.sh
I am in f1 function
I am in f2 function
I am in f1 function
```

Function with Parameters:

Function can accept parameters also. With in the function we can access parameters as follows:

- \$1 → First Parameter
- \$2 → Second Parameter
- \$@ → All Paramters
- \$* → All parameters
- \$# → Total number of parameters
- \$0 → It is script name but not function name

Q2) Write a Function to demonstrate how to access Function Parameters?

```
#!/bin/bash
function demo()
{
    echo "First Parameter: $1"
    echo "Second Parameter: $2"
    echo "Third Parameter: $3"
    echo "Total Number of Paramerters: $#"
```

```
demo 10 20 30 40 50
```

Output:

```
First Parameter: 10
Second Parameter: 20
Third Parameter: 30
Total Number of Paramerters: 5
All Parameters with @: 10 20 30 40 50
All Parameters with *: 10 20 30 40 50
Script Name:/home/durga/scripts/test.sh
```



Eg:

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

```
wish()
{
    if [ $# -eq 0 ]; then
        echo "Hello Guest Good Evening"
    else
        echo "Hello $1 Good Evening"
    fi
}
wish
wish Durga
wish Sunny
```

Output:

```
durgasoft@durgasoft:~/scripts$ test.sh
Hello Guest Good Evening
Hello Durga Good Evening
Hello Sunny Good Evening
```

Q3) Write a Function that takes 2 Integer Numbers as Parameters and perform Arithmetic Operations

```
#!/bin/bash
calc()
{
    if [ $# -ne 2 ]; then
        echo "You should pass exactly 2 arguments"
    else
        a=$1
        b=$2
        echo "$a+$b = $((a+b))"
        echo "$a-$b = $((a-b))"
        echo "$a*$b = $((a*b))"
        echo "$a/$b = $((a/b))"
        echo
    fi
}
calc 10
calc 20 10
calc 200 100
calc 2000 1000
```



Output:

You should pass exactly 2 arguments

20+10 = 30

20-10 = 10

20*10 = 200

20/10 = 2

200+100 = 300

200-100 = 100

200*100 = 20000

200/100 = 2

2000+1000 = 3000

2000-1000 = 1000

2000*1000 = 2000000

2000/1000 = 2

Q4) Write a Function to Print all Parameters?

```
#!/bin/bash
parameter_printing()
{
    if [ $# -eq 0 ]; then
        echo "No parameters passed to this function"
    else
        echo "All Passed Parameters are:"
        echo "....."
        for p in $@
        do
            echo $p
        done
    fi
}
parameter_printing
parameter_printing A B C D E
```

Output: No parameters passed to this function

All Passed Parameters are:

A
B
C
D
E



Q5) Write a Function to find Maximum of 2 given Integer Numbers?

```
#!/bin/bash
max()
{
    if [ $1 -gt $2 ]; then
        echo "The Maximum of $1 and $2 :$1"
    else
        echo "The Maximum of $1 and $2: $2"
    fi
}
max 10 20
max 200 100
```

Output:

The Maximum of 10 and 20: 20
The Maximum of 200 and 100 :200

Q6) Write a Function to find Maximum of 3 given Integer Numbers?

```
#!/bin/bash
max()
{
    a=$1
    b=$2
    c=$3
    if [ $a -gt $b -a $a -gt $c ]; then
        echo "Biggest Number:$a"
    elif [ $b -gt $c ]; then
        echo "Biggest Number:$b"
    else
        echo "Biggest Number:$c"
    fi
}
read -p "Enter First Number:" n1
read -p "Enter Second Number:" n2
read -p "Enter Third Number:" n3
max $n1 $n2 $n3
```



Q7) Write a Function to compare the given 2 Integers?

```
#!/bin/bash
compare()
{
    if [ $1 -eq $2 ]; then
        echo "Both Numbers are equal"
    elif [ $1 -gt $2 ]; then
        echo "$1 is greater than $2"
    else
        echo "$1 is less than $2"
    fi
}
compare 10 10
compare 10 20
compare 200 100
```

Output:

```
Both Numbers are equal
10 is less than 20
200 is greater than 100
```

Q8) Write a Function to find Factorial of a given Integer Number?

```
#!/bin/bash
factorial()
{
    original=$n
    fact=1
    while [ $n -gt 1 ]
    do
        let fact=fact*n
        let n--
    done
    echo "The Factorial of $original is: $fact"
}
read -p "Enter a number to find factorial:" n
factorial $n
}
```

```
#!/bin/bash
factorial()
```



```
{
  original=$1
  x=$1
  fact=1
  while [ $x -gt 1 ]
  do
    let fact=fact*x
    let x--
  done
  echo "The Factorial of $original is: $fact"
}
read -p "Enter a number to find factorial:" n
factorial $n
}
```

Q9) Write a Function to find Factorial of 1st 10 Natural Numbers?

```
#!/bin/bash
factorial()
{
  original=$1
  n=$1
  fact=1
  while [ $n -gt 1 ]
  do
    let fact=fact*n
    let n--
  done
  echo "The Factorial of $original: $fact"
}
for i in {1..10}
do
  factorial $i
done
```

```
durga@durga-VirtualBox:~/scripts$ test.sh
The Factorial of 1: 1
The Factorial of 2: 2
The Factorial of 3: 6
The Factorial of 4: 24
The Factorial of 5: 120
The Factorial of 6: 720
The Factorial of 7: 5040
```



The Factorial of 8: 40320
The Factorial of 9: 362880
The Factorial of 10: 3628800

Q10) Write a Function to Check whether the given Number is Even Number OR not?

```
#!/bin/bash
even_odd()
{
    n=$1
    if [ ${n%2} -eq 0 ]; then
        echo "$n is an Even Number"
    else
        echo "$n is an Odd Number"
    fi
}

read -p "Enter Any Number to test whether it is Even or Odd:" n
even_odd $n
```

```
durga@durga-VirtualBox:~/scripts$ test.sh
Enter Any Number to test whether it is Even or Odd:10
10 is an Even Number
durga@durga-VirtualBox:~/scripts$ test.sh
Enter Any Number to test whether it is Even or Odd:13
13 is an Odd Number
```

Q11) Write a Function to Test whether the given Number is Positive OR Negative Number?

```
#!/bin/bash
positive_negative()
{
    n=$1
    if [ $n -gt 0 ]; then
        echo "$n is a Positive Number"
    elif [ $n -lt 0 ]; then
        echo "$n is a Negative Number"
    else
        echo "It is just Zero"
    fi
}

read -p "Enter Any number to test whether it is positive or negative:" n
```



positive_negative \$n

```
durga@durga-VirtualBox:~/scripts$ test.sh
Enter Any number to test whether it is positive or negative:10
10 is a Positive Number
durga@durga-VirtualBox:~/scripts$ test.sh
Enter Any number to test whether it is positive or negative:-20
-20 is a Negative Number
durga@durga-VirtualBox:~/scripts$ test.sh
Enter Any number to test whether it is positive or negative:0
It is just Zero
```

Prime Numbers:

Any positive number greater than 1, which has no other factors except 1 and the number itself, is called prime number.

Eg: 2, 3, 5, 7, 11, 13 etc

Any Positive Number which has more than 2 factors, such type of numbers are called Composite Numbers.

Eg: 4, 6, 8, 12 etc

1 is neither prime nor composite. It is a unique number.

Q12) Write a Function to Check whether the given Number is Prime Number OR not?

```
#!/bin/bash
prime_check()
{
    n=$1
    if [ $n -le 1 ]; then
        echo "$n is not a PRIME number"
    else
        is_prime="yes"
        for ((i=2;i<n;i++))
        do
            if [ ${n%i} -eq 0 ]; then
                echo "$n is not a PRIME Number"
                is_prime="no"
                break
            fi
        done
    fi
}
```



```
done
if [ $is_prime = "yes" ]; then
    echo "$n is a PRIME Number"
fi
fi
}
read -p "Enter Any Number to test whether it is PRIME or not:" n
prime_check $n
```

```
durga@durga-VirtualBox:~/scripts$ test.sh
Enter Any Number to test whether it is PRIME or not:129
129 is not a PRIME Number
durga@durga-VirtualBox:~/scripts$ test.sh
Enter Any Number to test whether it is PRIME or not:127
127 is a PRIME Number
durga@durga-VirtualBox:~/scripts$ test.sh
Enter Any Number to test whether it is PRIME or not:19
19 is a PRIME Number
durga@durga-VirtualBox:~/script
```

Q13) Write a Function to generate Prime Numbers which are Less than OR Equal to given Number?

```
#!/bin/bash
prime_numbers_generator()
{
    n=$1
    for((n1=2;n1<=n;n1++)) //Repeat from 2 to n every number to test
    do
        is_prime="yes"
        for((i=2;i<n1;i++)) //To test current number n1 is prime or not
        do
            if [ ${n1%i} -eq 0 ]; then
                is_prime="no"
                break
            fi
        done
        if [ $is_prime = yes ]; then
            echo $n1
        fi
    done
}
read -p "Enter N value:" n
prime_numbers_generator $n
```



```
durga@durga-VirtualBox:~/scripts$ test.sh
Enter N value:20
2
3
5
7
11
13
17
19
```

Q14) Write a Function to generate 1st n Prime Numbers?

```
#!/bin/bash
prime_numbers_generator()
{
    n=$1
    count=0
    for((n1=2;n1>=2;n1++)) # n1=2,3,4,5,6,7,8,...
    do
        is_prime=yes
        for((i=2;i<n1;i++))
        do
            if [ ${n1%i} -eq 0 ]; then
                is_prime=no
                break
            fi
        done
        if [ $is_prime = yes ]; then
            echo $n1
            let count++
        fi
        if [ $count -eq $n ]; then
            break
        fi
    done
}
read -p "Enter n value:" n
prime_numbers_generator $n
```

```
durga@durga-VirtualBox:~/scripts$ test.sh
Enter n value:1
2
durga@durga-VirtualBox:~/scripts$ test.sh
```



```
Enter n value:2
2
3
durga@durga-VirtualBox:~/scripts$ test.sh
Enter n value:5
2
3
5
7
11
```

Variable Scope:

By default every variable in shell script is global. i.e we can access anywhere in our script. But before using a variable, it should be declared already.

Eg 1:

```
#!/bin/bash
f1()
{
    echo "x value : $x"
}
x=10
f1
```

Output: x value : 10

Eg 2:

```
#!/bin/bash
f1()
{
    x=20
    echo "x value : $x"
}
x=10
f1
echo "After f1 execution x value: $x"
```

Output:

x value : 20
After f1 execution x value: 20



Eg 3:

```
#!/bin/bash
f1()
{
    echo "x value : $x"
}
f1
x=10
f1
```

Output:

```
x value :
x value : 10
```

The variables which are declared inside a function, can be accessed outside of that function, because every variable has global scope by default.

Eg 4:

```
#!/bin/bash
f1()
{
    x=10
}
f1
echo "x value : $x"
```

Output: x value : 10

If we want a variable only within the function and should not be available outside of that function, then we have to use local keyword for the variable.
local variables can be accessed only inside function and cannot be accessed outside of that function.

Eg:

```
#!/bin/bash
f1()
{
    local x=10
    echo "Inside function x value: $x"
}
f1
echo "outside function x value : $x"
```



Ouput:

Inside function x value: 10

outside function x value :

Return Statement in Functions:

Every function in shell scripting returns some value. The default returned value is the exit code of the last command inside function.

But based on our requirement we can return values explicitly by using return statement.

`return <exitcode>`

The allowed values for exitcode are 0 to 255.

0 means successful

non-zero means unsuccessful.

We can get return value of function by using `$?`.

Eg:

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

```
sum()
```

```
{
```

```
  if [ $# != 2 ]; then
```

```
    echo "You should pass exactly two numbers"
```

```
    return 1
```

```
  else
```

```
    echo "The SUM:${($1+$2)}"
```

```
  fi
```

```
}
```

```
sum 10 20
```

```
echo "The Return value of this function:$?"
```

```
echo
```

```
sum 10
```

```
echo "The Return value of this function:$?"
```

```
durgasoft@durgasoft:~/scripts$ test.sh
```

```
The SUM:30
```

```
The Return value of this function:0
```

```
You should pass exactly two numbers
```

```
The Return value of this function:1
```



Use Case:

```
backup()
{
    commands to take backup
}
backup
if [ $? != 0 ]; then
    something goes wrong backup failed
else
    backup successful
fi
```

break vs exit vs return:

1) break:

We can use break only inside loops to break loop execution. We will come out of the loop and remaining statements after loop will be executed.

2) exit:

We can use anywhere exit statement to terminate script execution. The script will be terminated. No chance of executing any other statement.

3) return:

We can use return statement only inside function to stop function execution. After return statement, the remaining statements inside function won't be executed. But after function call the remaining statements will be executed.

How to Call Functions Present in another Script:

util.sh

```
#!/bin/bash
x=888
y=999
add()
{
    echo "$1 + $2 = $[$1+$2]"
}
multiply()
{
    echo "$1 * $2 = $[$1*$2]"
}
subtract()
```



```
{  
    echo "$1 - $2 = ${1-$2}"  
}  
divide()  
{  
    echo "$1 / $2 = ${1/$2}"  
}
```

test.sh

```
#!/bin/bash  
. ./util.sh #This is just inclusion and we are not executing util.sh  
add 10 20  
subtract 20 10  
multiply 10 20  
divide 20 10  
echo "The value of x:$x"  
echo "The value of y:$y"
```

```
durga@durga-VirtualBox:~/scripts$ test.sh  
10 + 20 = 30  
20 - 10 = 10  
10 * 20 = 200  
20 / 10 = 2  
The value of x:888  
The value of y:999
```