DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AMITY SCHOOL OF ENGINEERING AND TECHNOLOGY AMITY UNIVERSITY

UTTAR PRADESH

7th SEMESTER

LINUX FOR DEVICES

Lab Assignment

SUBMITTED BY:

NAME: **Krishan Kumar Makkar**

ENROLLMENT NO: **A2305218297**

BRANCH: **B. TECH. 7CSE5X**

BATCH: **2018-2022**

SUBMITTED TO:

FACULTY NAME: **Dr. Debjani Ghosh**

15th November, 2021

Date of submission**:**

Index

10	Write a shell script program to develop a scientific calculator.		07/09/21	14/09/21	
11	Write a shell Script program to check whether the given number is even or odd.		14/09/21	21/09/21	
12	Shell script Program to search whether element is present is in the list or not.		21/09/21	28/09/21	
13	Shell script program to check whether given file is a directory or not.		28/09/21	05/10/21	
14	Shell script program to count number of files in a Directory.		05/10/21	12/10/21	
15	Shell script program to copy contents of one file to another.		12/10/21	19/10/21	
16	Create directory, write contents on that and Copy to a suitable location in your home directory.		19/10/21	26/10/21	
17	Use a pipeline and command substitution to set the length of a line in file to a variable.		26/10/21	02/11/21	
18	Write a program using sed command to print duplicated lines of Input.		02/11/21	09/11/21	
19	Study the process of writing a device driver, or a kernel module.		09/11/21	09/11/21	

Experiment-1

Aim: Study of logging/logout details.

Theory:

To ensure security and organisation on a system with many users, Linux/Unix machines employ a system of user accounts. The user accounting features of Unix provide a basis for analysis and control of system resources, preventing any user from taking up more than his or her share, and preventing unauthorised people from accessing the system. Every user of a Unix system must get permission b some access control mechanism.

Commands related to logging in Linux

1 **login** - begin session on the system

Synopsis:

login [-p] [-h host] [username] [ENV=VAR...]
login [-p] [-h host] -f username
login [-p] -r host

Description:

The login program is used to establish a new session with the system. It is normally invoked automatically by responding to the login: prompt on the user's terminal. login may be special to the shell and may not be invoked as a sub-process. When called from a shell, login should be executed as exec login which will cause the user to exit from the current shell (and thus will prevent the new logged in user to return to the session of the caller). Attempting to execute login from any shell but the login shell will produce an error message.

The user is then prompted for a password, where appropriate.Echoing is disabled to prevent revealing the password. Only a small number of password failures are permitted before login exits and the communications link is severed.

Options:

-f

Do not perform authentication, user is pre-authenticated.
Note: In that case, username is mandatory.

-h

Name of the remote host for this login.

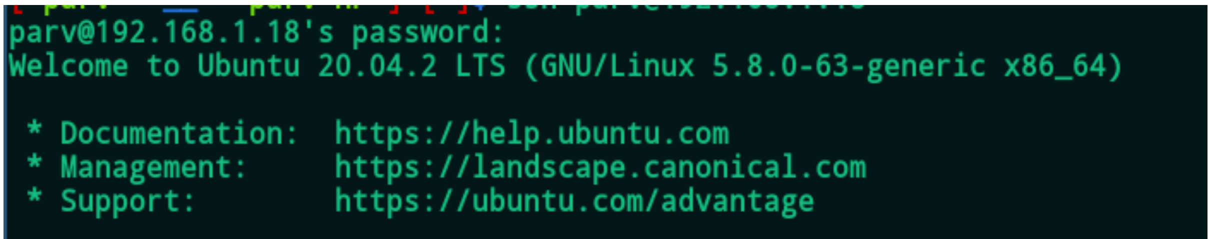
-p

Preserve environment.

-r

Perform auto-login protocol for login.

The -r, -h and -f options are only used when login is invoked by root.



```
parv@192.168.1.18's password:
Welcome to Ubuntu 20.04.2 LTS (GNU/Linux 5.8.0-63-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage
```

2 **passwd -**change user password

Synopsis:

```
passwd [options] [LOGIN]
```

Description:

The passwd command changes passwords for user accounts. A normal user may only change the password for their own account, while the superuser may change the password for any account. passwd also changes the account or associated password validity period.

The user is first prompted for their old password, if one is present. This password is then encrypted and compared against the stored password. The user has only one chance to enter the correct password. The superuser is permitted to bypass this step so that forgotten passwords may be changed.

After the password has been entered, password aging information is checked to see if the user is permitted to change the password at this time. If not, passwd refuses to change the password and exits.

The user is then prompted twice for a replacement password. The second entry is compared against the first and both are required to match in order for the password to be changed.

Then, the password is tested for complexity. As a general guideline, passwords should consist of 6 to 8 characters including one or more characters from each of the following sets:

- lower case alphabetics
- digits 0 through 9
- punctuation marks

Care must be taken not to include the system default erase or kill characters. passwd will reject any password which is not suitably complex.

Options:

The options which apply to the passwd command are:

-a, --all

This option can be used only with -S and causes show status for all users.

-d, --delete

Delete a user's password (make it empty). This is a quick way to disable a password for an account. It will set the named account passwordless.

-e, --expire

Immediately expire an account's password. This in effect can force a user to change their password at the user's next login.

-h, --help

Display help message and exit.

-i, --inactive INACTIVE

This option is used to disable an account after the password has been expired for a number of days. After a user account has had an expired password for INACTIVE days, the user may no longer sign on to the account.

-k, —keep-tokens

Indicate password change should be performed only for expired authentication tokens (passwords). The user wishes to keep their non-expired tokens as before.

-l, —lock

Lock the password of the named account. This option disables a password by changing it to a value which matches no possible encrypted value (it adds a ′!′ at the beginning of the password).
Note that this does not disable the account. The user may still be able to login using another authentication token (e.g. an SSH key). To disable the account, administrators should use usermod --expiredate 1 (this set the account's expire date to Jan 2, 1970).

-n, --mindays MIN_DAYS

Set the minimum number of days between password changes to MIN_DAYS. A value of zero for this field indicates that the user may change their password at any time.

-q, —quiet

Quiet mode.

-r, --repository REPOSITORY

change password in REPOSITORY repository

-R, --root CHROOT_DIR

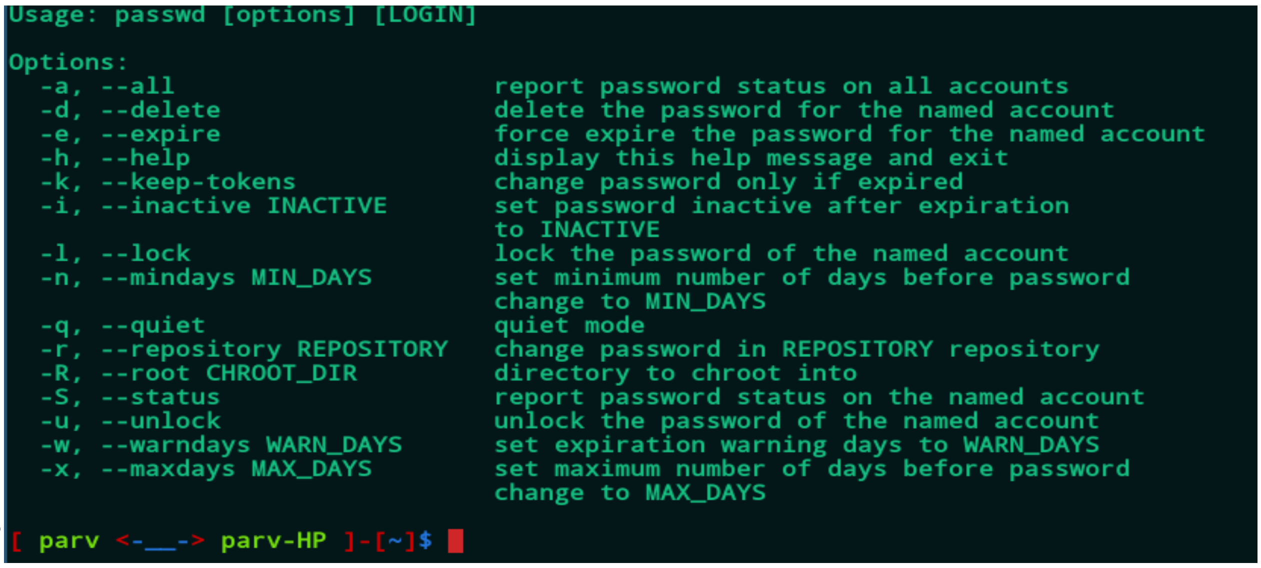
Apply changes in the CHROOT_DIR directory and use the configuration files from the CHROOT_DIR directory.

-S, —status

Display account status information. The status information consists of 7 fields. The first field is the user's login name. The second field indicates if the user account has a locked password (L), has no password (NP), or has a usable password (P). The third field gives the date of the last password change. The next four fields are the minimum age, maximum age, warning period, and inactivity period for the password. These ages are expressed in days.

-u, —unlock

Unlock the password of the named account. This option re-enables a password by changing the password back to itsprevious value (to the value before using the -l option)



```
Usage: passwd [options] [LOGIN]

Options:
  -a, --all                     report password status on all accounts
  -d, --delete                  delete the password for the named account
  -e, --expire                  force expire the password for the named account
  -h, --help                    display this help message and exit
  -k, --keep-tokens             change password only if expired
  -i, --inactive INACTIVE       set password inactive after expiration
                                to INACTIVE
  -l, --lock                    lock the password of the named account
  -n, --mindays MIN_DAYS        set minimum number of days before password
                                change to MIN_DAYS
  -q, --quiet                   quiet mode
  -r, --repository REPOSITORY   change password in REPOSITORY repository
  -R, --root CHROOT_DIR         directory to chroot into
  -S, --status                  report password status on the named account
  -u, --unlock                  unlock the password of the named account
  -w, --warndays WARN_DAYS      set expiration warning days to WARN_DAYS
  -x, --maxdays MAX_DAYS        set maximum number of days before password
                                change to MAX_DAYS

[ parv <-__-> parv-HP ]-[~]$ 
```

3 **who -** show who is logged on

Synopsis:

who [OPTION]... [FILE | ARG1 ARG2]

Description:

Print information about users who are currently logged in.

Options:

-a, --all
same as -b -d --login -p -r -t -T -u

-b, --boot
time of last system boot

-d, --dead
print dead processes

-H, --heading
print line of column headings

--ips print ips instead of hostnames. with --lookup, canonicalizes based on stored IP, if available, rather than stored hostname

-l, --login
print system login processes--lookup
attempt to canonicalize hostnames via DNS

-m

only hostname and user associated with stdin

-p, --process

print active processes spawned by init

-q, —count

all login names and number of users logged on

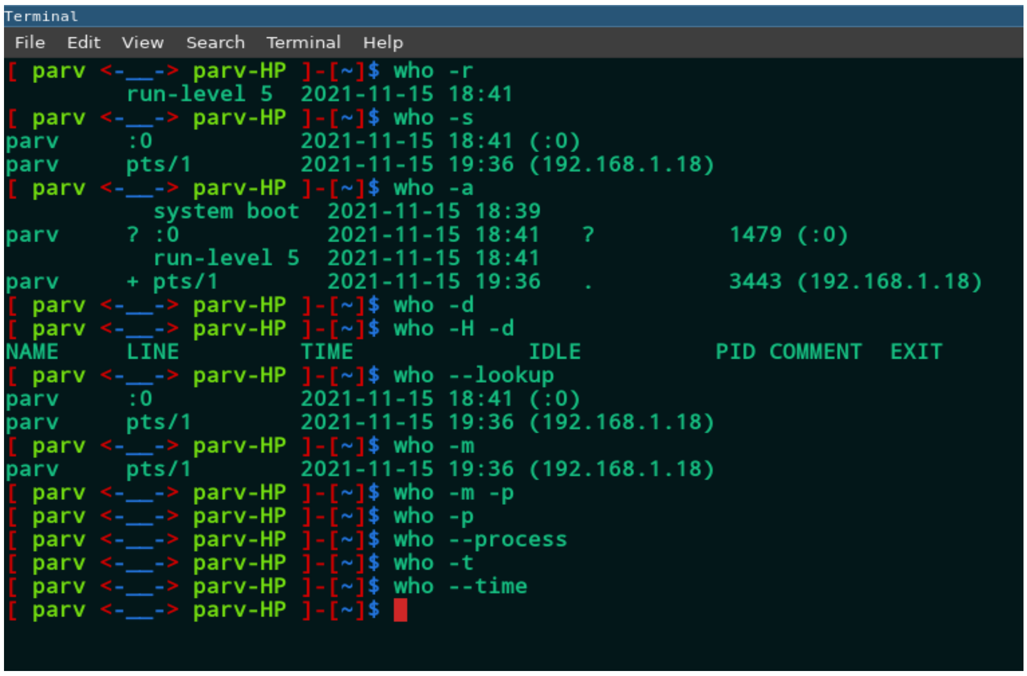
-r, --runlevel

print current runlevel

-s, --short

print only name, line, and time (default)

-t, --time print last system clock change



```
[ parv <-__-> parv-HP ]-[~]$ who -r
         run-level 5  2021-11-15 18:41
[ parv <-__-> parv-HP ]-[~]$ who -s
parv     :0           2021-11-15 18:41 (:0)
parv     pts/1        2021-11-15 19:36 (192.168.1.18)
[ parv <-__-> parv-HP ]-[~]$ who -a
           system boot  2021-11-15 18:39
parv     ? :0           2021-11-15 18:41   ?          1479 (:0)
           run-level 5  2021-11-15 18:41
parv     + pts/1        2021-11-15 19:36   .          3443 (192.168.1.18)
[ parv <-__-> parv-HP ]-[~]$ who -d
[ parv <-__-> parv-HP ]-[~]$ who -H -d
NAME     LINE         TIME             IDLE          PID COMMENT   EXIT
[ parv <-__-> parv-HP ]-[~]$ who --lookup
parv     :0           2021-11-15 18:41 (:0)
parv     pts/1        2021-11-15 19:36 (192.168.1.18)
[ parv <-__-> parv-HP ]-[~]$ who -m
parv     pts/1        2021-11-15 19:36 (192.168.1.18)
[ parv <-__-> parv-HP ]-[~]$ who -m -p
[ parv <-__-> parv-HP ]-[~]$ who -p
[ parv <-__-> parv-HP ]-[~]$ who --process
[ parv <-__-> parv-HP ]-[~]$ who -t
[ parv <-__-> parv-HP ]-[~]$ who --time
[ parv <-__-> parv-HP ]-[~]$
```

Commands related to logging out in Linux

4 **exit:**

This command is used to exit the present session. All the processes running in the current session are terminated and the temporary environment variables are removed.

According to the manual the exit command is or exit() terminates the current process immediately. The signal for the same is sent to the parent processes as well.

5 **clear**:

This command is used to clear the terminal screen. All the written command are removes from the standard output.

clear clears your screen if this is possible, including its scrollback buffer (if the extended "E3" capability is defined). clear looks in the environment for the terminal type given by the environment variable TERM, and then in the terminfo database to determine how to clear the screen.

clear writes to the standard output. You can redirect the standard output to a file (which prevents clear from actually clearing the screen), and later cat the file to the screen, clearing it at that point.

Experiment-3

Aim: Study of Unix/Linux general purpose utility command list obtained from (man, who, cat, cd, cp, ps, ls, mv, rm, mkdir, rmdir, echo, more, date, time, kill, history, chmod, chown, finger, pwd, cal, logout, shutdown) commands.

Procedure:

1 **who-** to display the current user logged into the system and the processes they are executing.
2 **whoami**- to display the current user logged into the terminal.

3 **pwd Command**

The pwd command is used to display the location of the current working directory.

4 **mkdir Command**

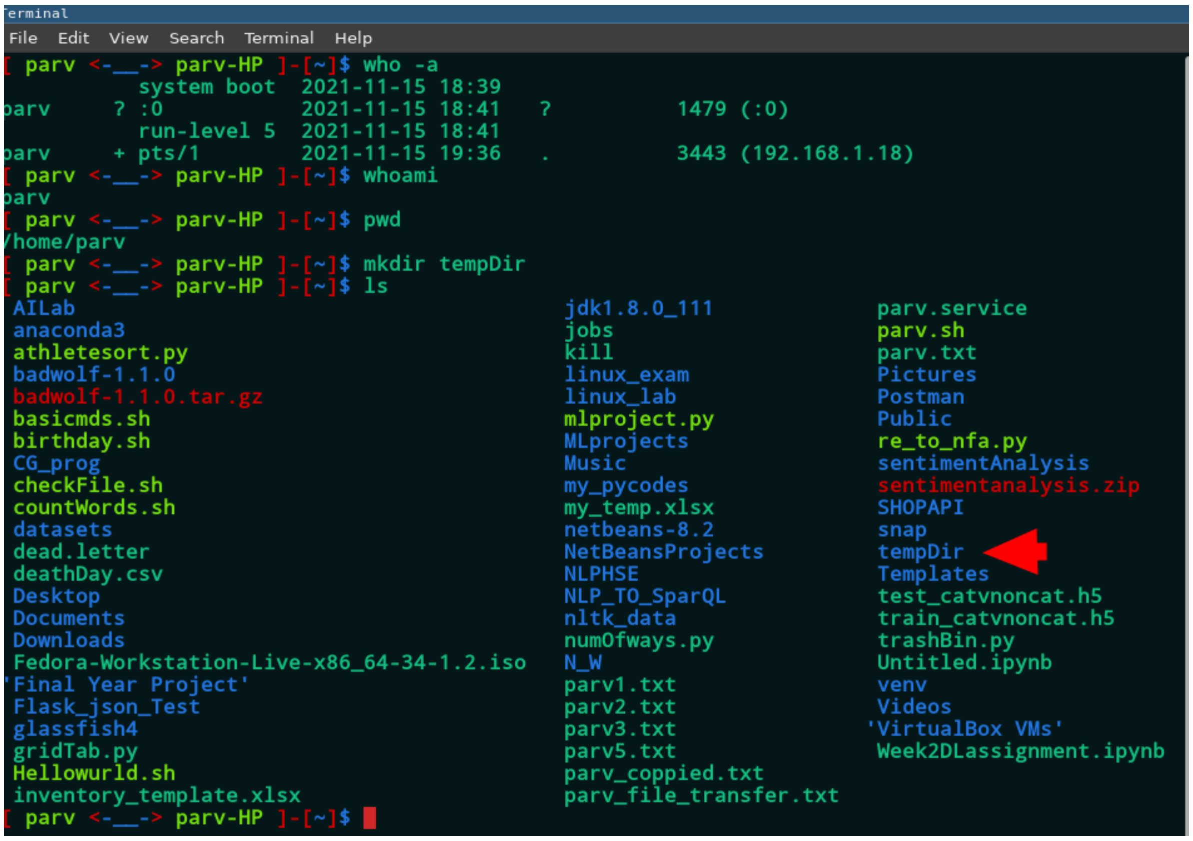
The mkdir command is used to create a new directory under any directory.

5 **rmdir Command**

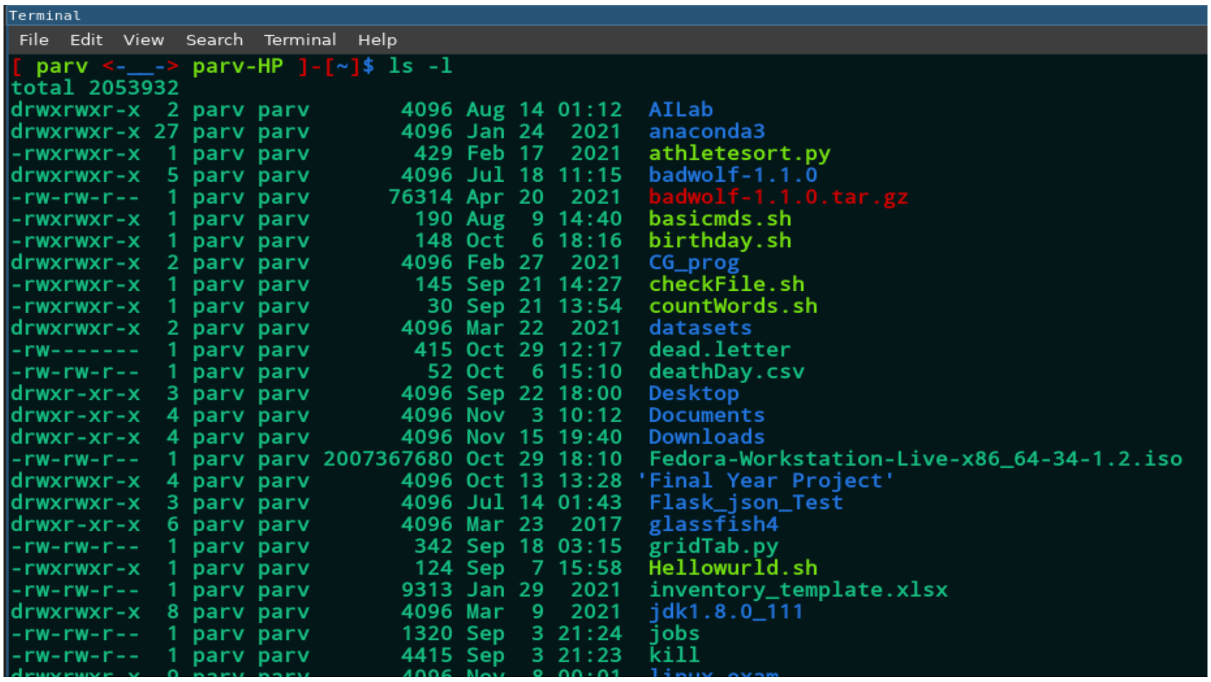
The command is used to remove a directory. The directory to remove should be empty.

6 **ls Command**

The ls command is used to display a list of content of a directory.



```
[ parv <-__-> parv-HP ]-[~]$ who -a
           system boot  2021-11-15 18:39
parv     ? :0           2021-11-15 18:41   ?          1479 (:0)
           run-level 5  2021-11-15 18:41
parv     + pts/1        2021-11-15 19:36   .          3443 (192.168.1.18)
[ parv <-__-> parv-HP ]-[~]$ whoami
parv
[ parv <-__-> parv-HP ]-[~]$ pwd
/home/parv
[ parv <-__-> parv-HP ]-[~]$ mkdir tempDir
[ parv <-__-> parv-HP ]-[~]$ ls
 AILab                                       jdk1.8.0_111             parv.service
 anaconda3                                   jobs                     parv.sh
 athletesort.py                              kill                     parv.txt
 badwolf-1.1.0                               linux_exam               Pictures
 badwolf-1.1.0.tar.gz                        linux_lab                Postman
 basicmds.sh                                 mlproject.py             Public
 birthday.sh                                 MLprojects               re_to_nfa.py
 CG_prog                                     Music                    sentimentAnalysis
 checkFile.sh                                my_pycodes               sentimentanalysis.zip
 countWords.sh                               my_temp.xlsx             SHOPAPI
 datasets                                    netbeans-8.2             snap
 dead.letter                                 NetBeansProjects         tempDir
 deathDay.csv                                NLPHSE                   Templates
 Desktop                                     NLP_TO_SparQL            test_catvnoncat.h5
 Documents                                   nltk_data                train_catvnoncat.h5
 Downloads                                   numOfways.py             trashBin.py
 Fedora-Workstation-Live-x86_64-34-1.2.iso   N_W                      Untitled.ipynb
'Final Year Project'                         parv1.txt                venv
 Flask_json_Test                             parv2.txt                Videos
 glassfish4                                  parv3.txt               'VirtualBox VMs'
 gridTab.py                                  parv5.txt                Week2DLassignment.ipynb
 Hellowurld.sh                               parv_coppied.txt
 inventory_template.xlsx                     parv_file_transfer.txt
[ parv <-__-> parv-HP ]-[~]$ 
```



```
[ parv <-__-> parv-HP ]-[~]$ ls -l
total 2053932
drwxrwxr-x  2 parv parv       4096 Aug 14 01:12  AILab
drwxrwxr-x 27 parv parv       4096 Jan 24  2021  anaconda3
-rwxrwxr-x  1 parv parv        429 Feb 17  2021  athletesort.py
drwxrwxr-x  5 parv parv       4096 Jul 18 11:15  badwolf-1.1.0
-rw-rw-r--  1 parv parv      76314 Apr 20  2021  badwolf-1.1.0.tar.gz
-rwxrwxr-x  1 parv parv        190 Aug  9 14:40  basicmds.sh
-rwxrwxr-x  1 parv parv        148 Oct  6 18:16  birthday.sh
drwxrwxr-x  2 parv parv       4096 Feb 27  2021  CG_prog
-rwxrwxr-x  1 parv parv        145 Sep 21 14:27  checkFile.sh
-rwxrwxr-x  1 parv parv         30 Sep 21 13:54  countWords.sh
drwxrwxr-x  2 parv parv       4096 Mar 22  2021  datasets
-rw-------  1 parv parv        415 Oct 29 12:17  dead.letter
-rw-rw-r--  1 parv parv         52 Oct  6 15:10  deathDay.csv
drwxr-xr-x  3 parv parv       4096 Sep 22 18:00  Desktop
drwxr-xr-x  4 parv parv       4096 Nov  3 10:12  Documents
drwxr-xr-x  4 parv parv       4096 Nov 15 19:40  Downloads
-rw-rw-r--  1 parv parv 2007367680 Oct 29 18:10  Fedora-Workstation-Live-x86_64-34-1.2.iso
drwxrwxr-x  4 parv parv       4096 Oct 13 13:28 'Final Year Project'
drwxrwxr-x  3 parv parv       4096 Jul 14 01:43  Flask_json_Test
drwxr-xr-x  6 parv parv       4096 Mar 23  2017  glassfish4
-rw-rw-r--  1 parv parv        342 Sep 18 03:15  gridTab.py
-rwxrwxr-x  1 parv parv        124 Sep  7 15:58  Hellowurld.sh
-rw-rw-r--  1 parv parv       9313 Jan 29  2021  inventory_template.xlsx
drwxrwxr-x  8 parv parv       4096 Mar  9  2021  jdk1.8.0_111
-rw-rw-r--  1 parv parv       1320 Sep  3 21:24  jobs
-rw-rw-r--  1 parv parv       4415 Sep  3 21:23  kill
drwxrwxr-x  9 parv parv       4096 Nov  8 00:01  linux_exam
```

7 **cd Command**

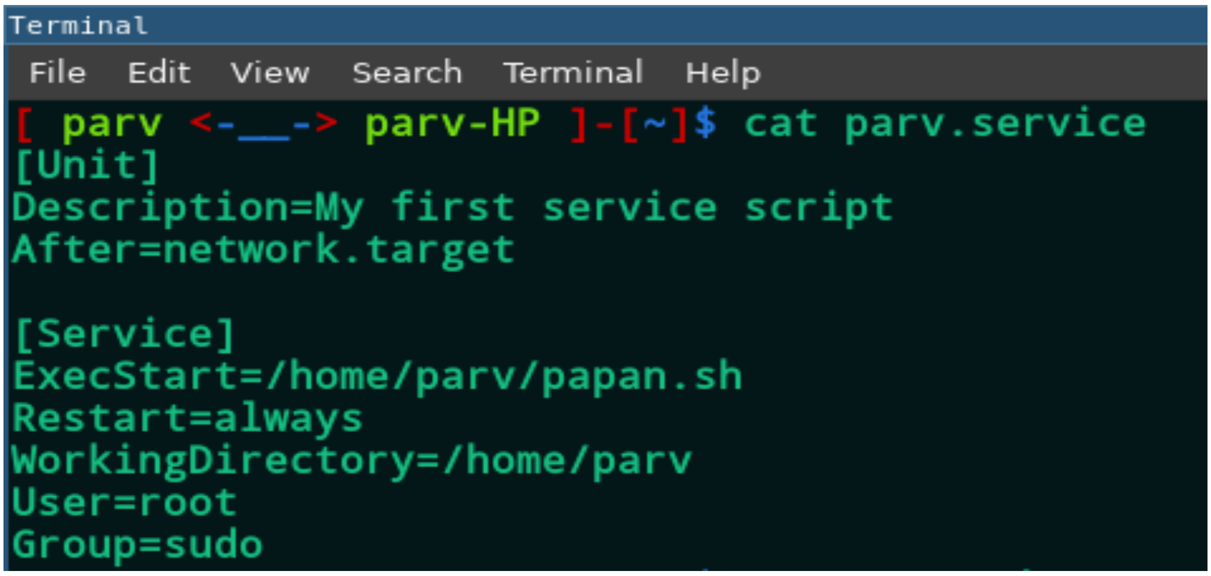
The cd command is used to change the current directory.

8 **touch Command**

The touch command is used to create empty files. We can create multiple empty files by executing it once.

9 **cat Command**

The cat command is a multi-purpose utility in the Linux system. It can be used to create a file,



```
[ parv <-__-> parv-HP ]-[~]$ cat parv.service
[Unit]
Description=My first service script
After=network.target

[Service]
ExecStart=/home/parv/papan.sh
Restart=always
WorkingDirectory=/home/parv
User=root
Group=sudo
```

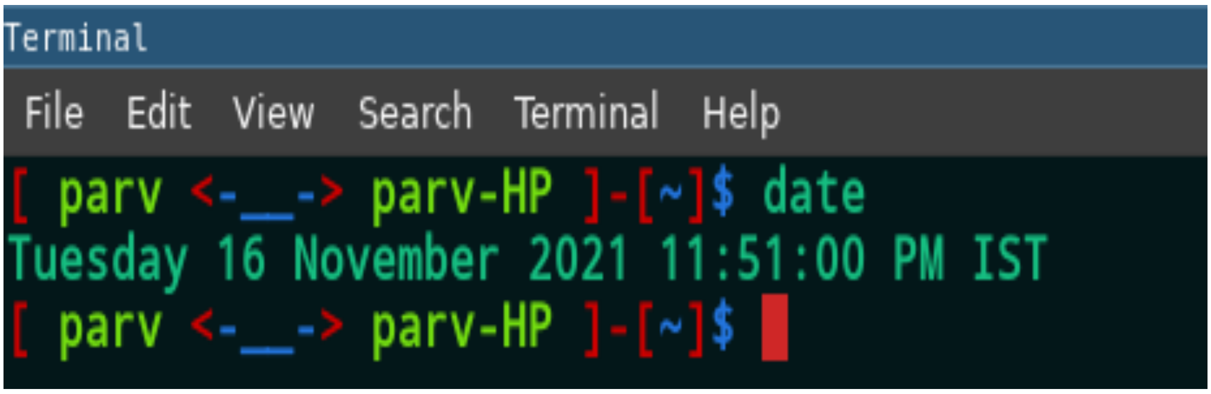
display content of the file, copy the content of one file to another file, and more.

10 Man command

man command in Linux is used to display the user manual of any command that we can run on the terminal. It provides a detailed view of the command which includes NAME, SYNOPSIS, DESCRIPTION, OPTIONS, EXIT STATUS, RETURN VALUES, ERRORS, FILES, VERSIONS, EXAMPLES, AUTHORS and SEE ALSO.

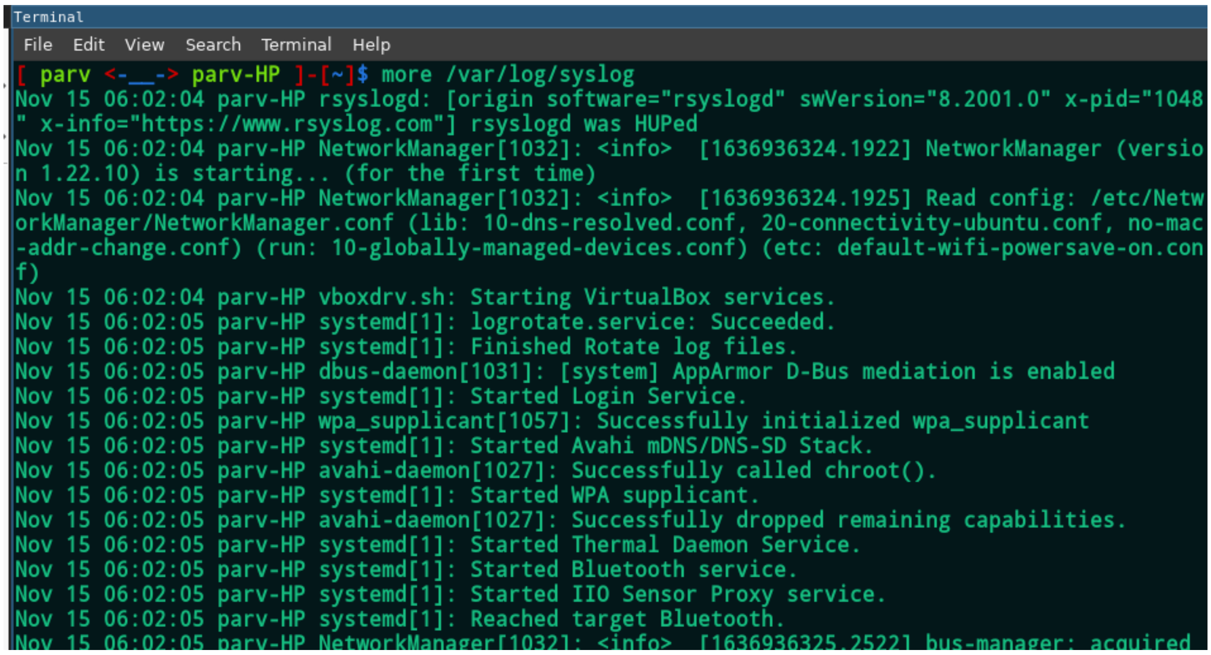
9 **cp** - this command is used to copy the contents of one file to other.

10 **date:** Displays the current date and time.



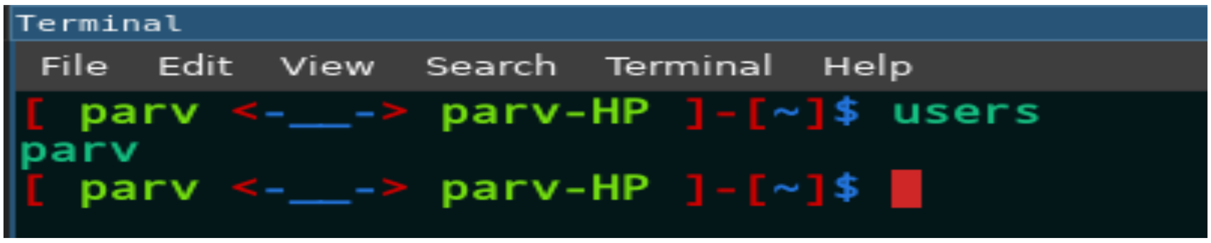
```
[ parv <-__-> parv-HP ]-[~]$ date
Tuesday 16 November 2021 11:51:00 PM IST
[ parv <-__-> parv-HP ]-[~]$ 
```

11 **more:** Displays the text files or log files in the CLI.



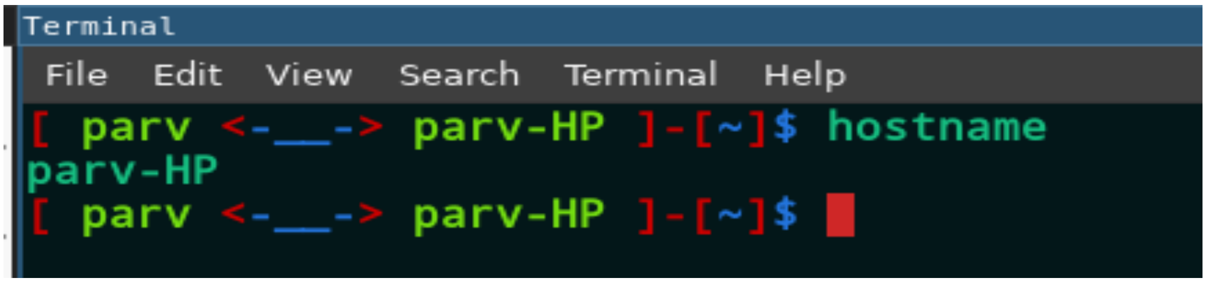
```
[ parv <-__-> parv-HP ]-[~]$ more /var/log/syslog
Nov 15 06:02:04 parv-HP rsyslogd: [origin software="rsyslogd" swVersion="8.2001.0" x-pid="1048
" x-info="https://www.rsyslog.com"] rsyslogd was HUPed
Nov 15 06:02:04 parv-HP NetworkManager[1032]: <info>  [1636936324.1922] NetworkManager (versio
n 1.22.10) is starting... (for the first time)
Nov 15 06:02:04 parv-HP NetworkManager[1032]: <info>  [1636936324.1925] Read config: /etc/Netw
orkManager/NetworkManager.conf (lib: 10-dns-resolved.conf, 20-connectivity-ubuntu.conf, no-mac
-addr-change.conf) (run: 10-globally-managed-devices.conf) (etc: default-wifi-powersave-on.con
f)
Nov 15 06:02:04 parv-HP vboxdrv.sh: Starting VirtualBox services.
Nov 15 06:02:05 parv-HP systemd[1]: logrotate.service: Succeeded.
Nov 15 06:02:05 parv-HP systemd[1]: Finished Rotate log files.
Nov 15 06:02:05 parv-HP dbus-daemon[1031]: [system] AppArmor D-Bus mediation is enabled
Nov 15 06:02:05 parv-HP systemd[1]: Started Login Service.
Nov 15 06:02:05 parv-HP wpa_supplicant[1057]: Successfully initialized wpa_supplicant
Nov 15 06:02:05 parv-HP systemd[1]: Started Avahi mDNS/DNS-SD Stack.
Nov 15 06:02:05 parv-HP avahi-daemon[1027]: Successfully called chroot().
Nov 15 06:02:05 parv-HP systemd[1]: Started WPA supplicant.
Nov 15 06:02:05 parv-HP avahi-daemon[1027]: Successfully dropped remaining capabilities.
Nov 15 06:02:05 parv-HP systemd[1]: Started Thermal Daemon Service.
Nov 15 06:02:05 parv-HP systemd[1]: Started Bluetooth service.
Nov 15 06:02:05 parv-HP systemd[1]: Started IIO Sensor Proxy service.
Nov 15 06:02:05 parv-HP systemd[1]: Reached target Bluetooth.
Nov 15 06:02:05 parv-HP NetworkManager[1032]: <info>  [1636936325.2522] bus-manager: acquired
```

12 **users:** Used to show usernames of users currently logged in to the current host.



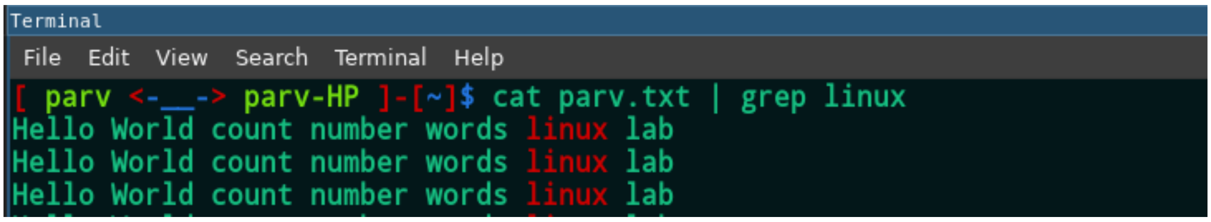
```
[ parv <-__-> parv-HP ]-[~]$ users
parv
[ parv <-__-> parv-HP ]-[~]$ 
```

13 **hostname:** Get or set the hostname or DNS domain name. Can also be used to display the IP of remote machine.



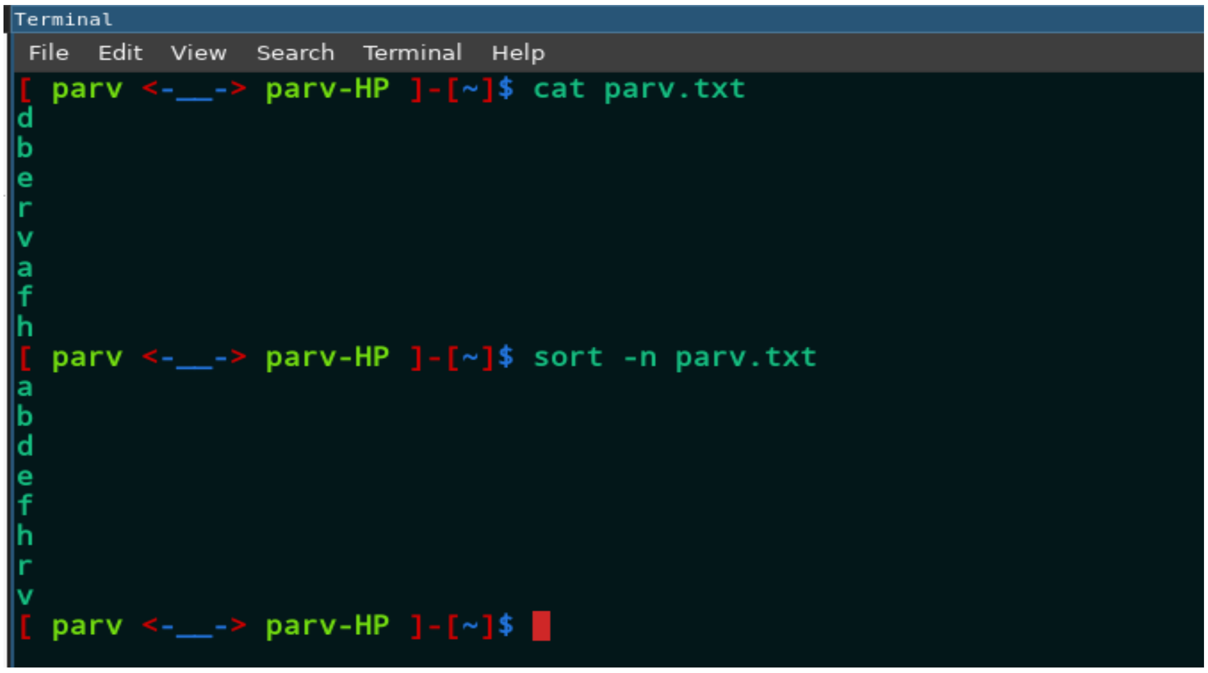
```
[ parv <-__-> parv-HP ]-[~]$ hostname
parv-HP
[ parv <-__-> parv-HP ]-[~]$ 
```

14 **grep:** Searches a file for a particular pattern of characters, and displays all lines that contain that pattern.



```
[ parv <-__-> parv-HP ]-[~]$ cat parv.txt | grep linux
Hello World count number words linux lab
Hello World count number words linux lab
Hello World count number words linux lab
```

15 **sort:** It is used to print the output of file in a reordered/sorted way to read the data



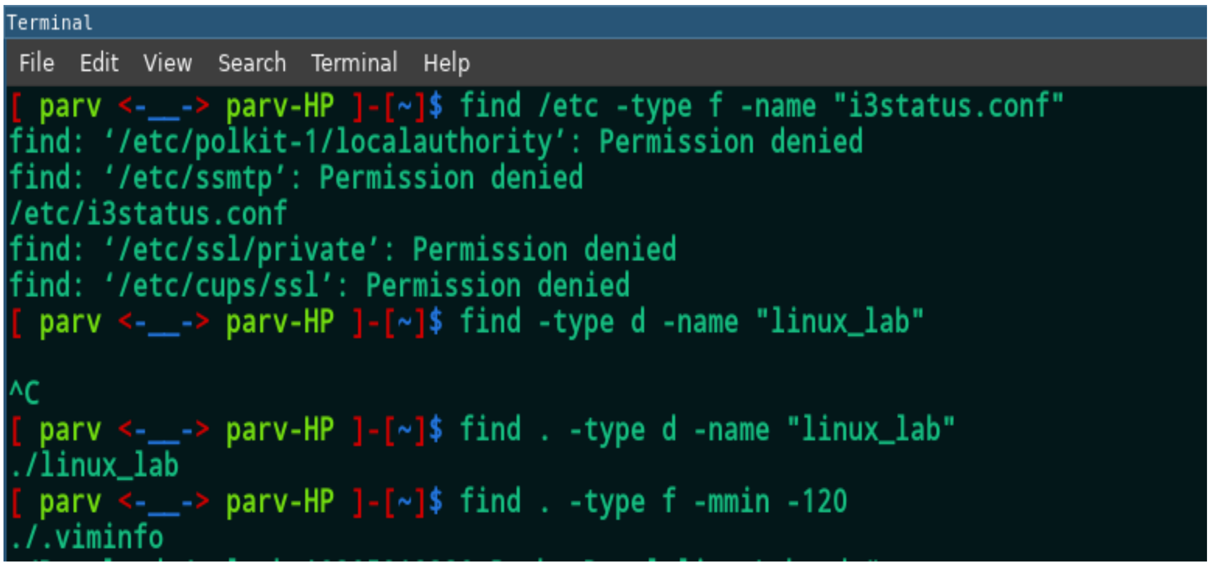
```
[ parv <-__-> parv-HP ]-[~]$ cat parv.txt
d
b
e
r
v
a
f
h
[ parv <-__-> parv-HP ]-[~]$ sort -n parv.txt
a
b
d
e
f
h
r
v
[ parv <-__-> parv-HP ]-[~]$
```

more efficiently

16 **echo:** Used to display the line or text passed as an argument.

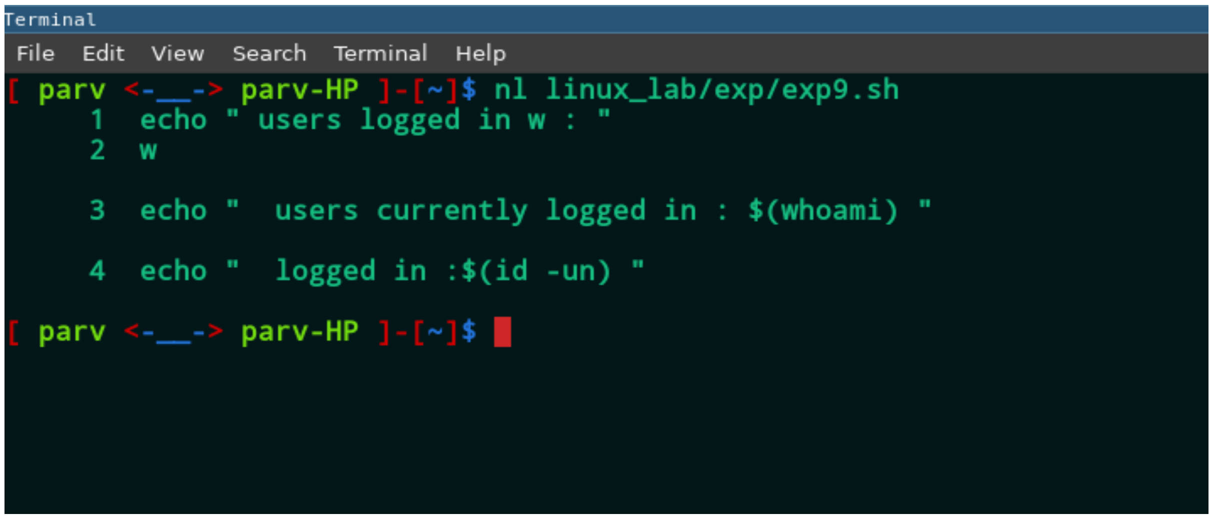
17 **chmod:** To change directory or file permission in Linux.

18 **nano:** Opens the text editor, nano. Another alternative is vim.



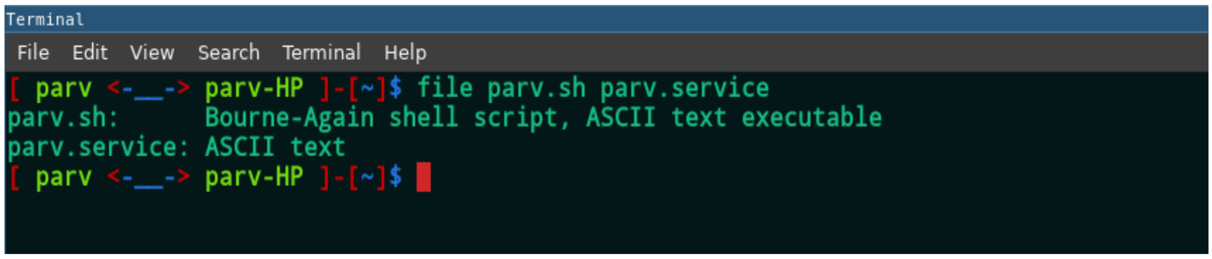
```
[ parv <-__-> parv-HP ]-[~]$ find /etc -type f -name "i3status.conf"
find: '/etc/polkit-1/localauthority': Permission denied
find: '/etc/ssmtp': Permission denied
/etc/i3status.conf
find: '/etc/ssl/private': Permission denied
find: '/etc/cups/ssl': Permission denied
[ parv <-__-> parv-HP ]-[~]$ find -type d -name "linux_lab"

^C
[ parv <-__-> parv-HP ]-[~]$ find . -type d -name "linux_lab"
./linux_lab
[ parv <-__-> parv-HP ]-[~]$ find . -type f -mmin -120
./.viminfo
```

19 **find:** Used to find a file/directory in Linux.



```
[ parv <-__-> parv-HP ]-[~]$ nl linux_lab/exp/exp9.sh
     1	echo " users logged in w : "
     2	w

     3	echo "  users currently logged in : $(whoami) "

     4	echo "  logged in :$(id -un) "

[ parv <-__-> parv-HP ]-[~]$
```

20 **nl:** Outputs file in a numbered format.



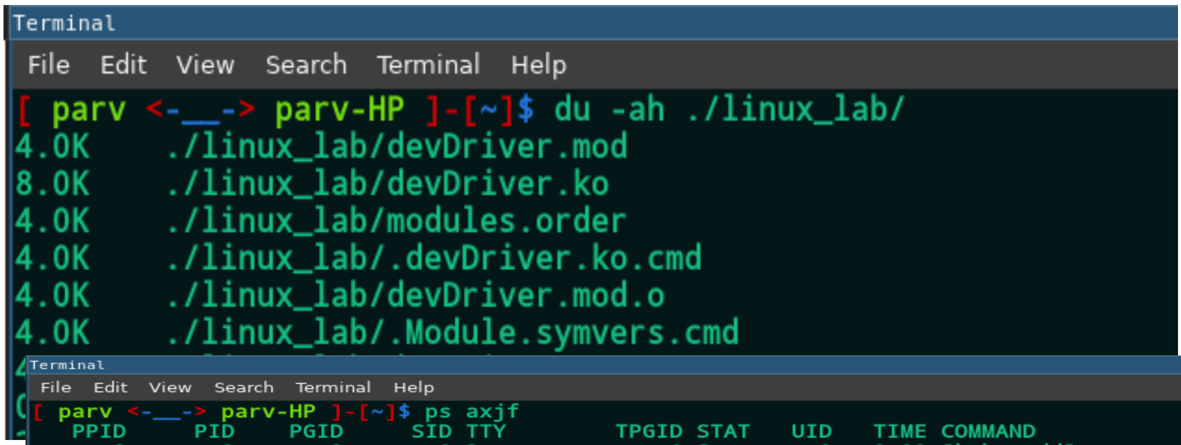
```
[ parv <-__-> parv-HP ]-[~]$ file parv.sh parv.service
parv.sh:      Bourne-Again shell script, ASCII text executable
parv.service: ASCII text
[ parv <-__-> parv-HP ]-[~]$
```

21 **file** Gives information about the type of file.

22 **mv:** Change file name or directory location.

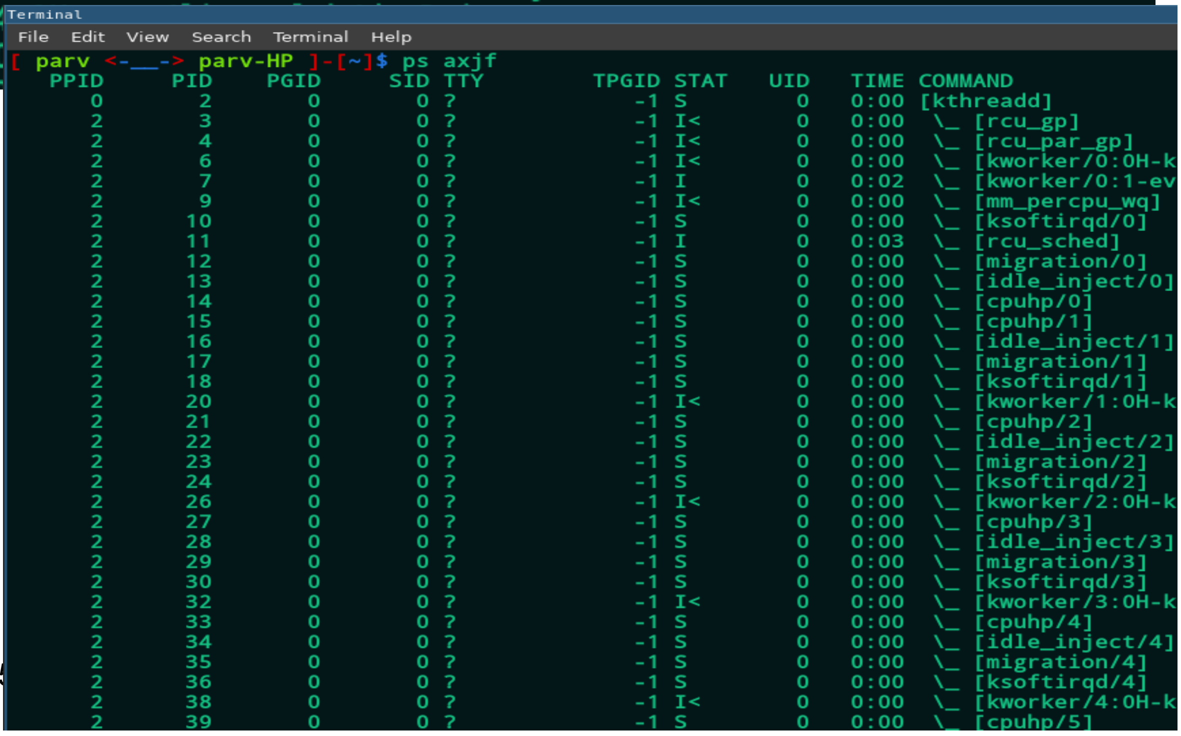
23 **rm:** Remove files.

24 **df:** Summarises free space on disk device.


Terminal

File Edit View Search Terminal Help

```
[ parv <-__-> parv-HP ]-[~]$ du -ah ./linux_lab/
4.0K    ./linux_lab/devDriver.mod
8.0K    ./linux_lab/devDriver.ko
4.0K    ./linux_lab/modules.order
4.0K    ./linux_lab/.devDriver.ko.cmd
4.0K    ./linux_lab/devDriver.mod.o
4.0K    ./linux_lab/.Module.symvers.cmd
```


Terminal

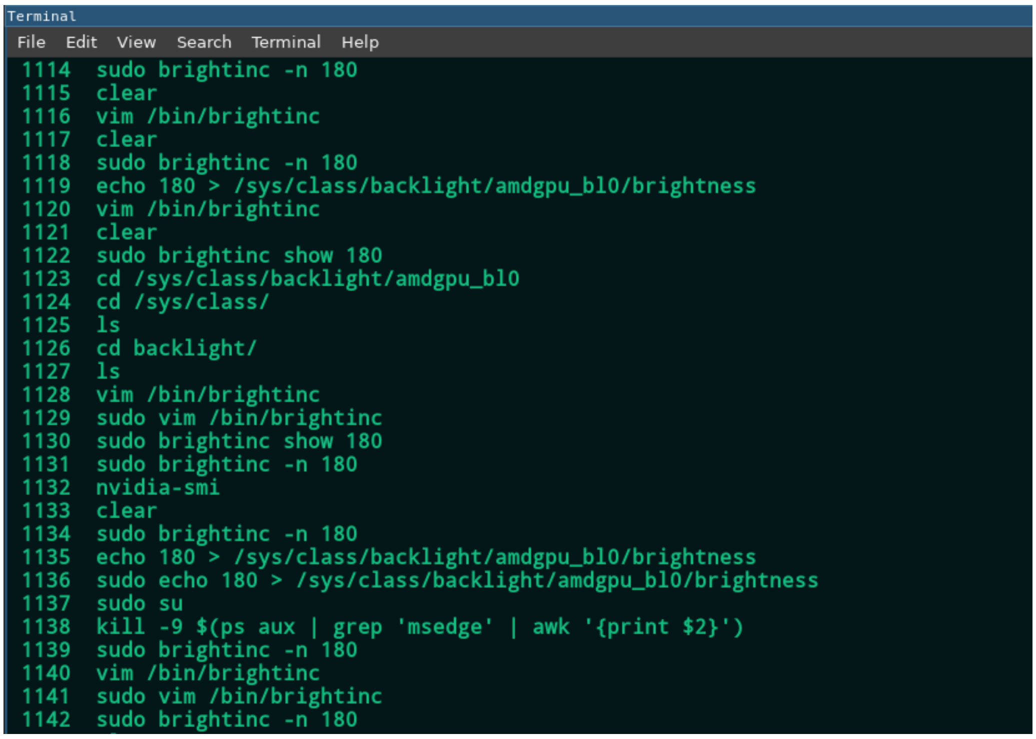
File Edit View Search Terminal Help

```
[ parv <-__-> parv-HP ]-[~]$ ps axjf
   PPID     PID    PGID     SID TTY        TPGID STAT   UID   TIME COMMAND
      0       2       0       0 ?             -1 S        0   0:00 [kthreadd]
      2       3       0       0 ?             -1 I<       0   0:00  \_ [rcu_gp]
      2       4       0       0 ?             -1 I<       0   0:00  \_ [rcu_par_gp]
      2       6       0       0 ?             -1 I<       0   0:00  \_ [kworker/0:0H-k
      2       7       0       0 ?             -1 I        0   0:02  \_ [kworker/0:1-ev
      2       9       0       0 ?             -1 I<       0   0:00  \_ [mm_percpu_wq]
      2      10       0       0 ?             -1 S        0   0:00  \_ [ksoftirqd/0]
      2      11       0       0 ?             -1 I        0   0:03  \_ [rcu_sched]
      2      12       0       0 ?             -1 S        0   0:00  \_ [migration/0]
      2      13       0       0 ?             -1 S        0   0:00  \_ [idle_inject/0]
      2      14       0       0 ?             -1 S        0   0:00  \_ [cpuhp/0]
      2      15       0       0 ?             -1 S        0   0:00  \_ [cpuhp/1]
      2      16       0       0 ?             -1 S        0   0:00  \_ [idle_inject/1]
      2      17       0       0 ?             -1 S        0   0:00  \_ [migration/1]
      2      18       0       0 ?             -1 S        0   0:00  \_ [ksoftirqd/1]
      2      20       0       0 ?             -1 I<       0   0:00  \_ [kworker/1:0H-k
      2      21       0       0 ?             -1 S        0   0:00  \_ [cpuhp/2]
      2      22       0       0 ?             -1 S        0   0:00  \_ [idle_inject/2]
      2      23       0       0 ?             -1 S        0   0:00  \_ [migration/2]
      2      24       0       0 ?             -1 S        0   0:00  \_ [ksoftirqd/2]
      2      26       0       0 ?             -1 I<       0   0:00  \_ [kworker/2:0H-k
      2      27       0       0 ?             -1 S        0   0:00  \_ [cpuhp/3]
      2      28       0       0 ?             -1 S        0   0:00  \_ [idle_inject/3]
      2      29       0       0 ?             -1 S        0   0:00  \_ [migration/3]
      2      30       0       0 ?             -1 S        0   0:00  \_ [ksoftirqd/3]
      2      32       0       0 ?             -1 I<       0   0:00  \_ [kworker/3:0H-k
      2      33       0       0 ?             -1 S        0   0:00  \_ [cpuhp/4]
      2      34       0       0 ?             -1 S        0   0:00  \_ [idle_inject/4]
      2      35       0       0 ?             -1 S        0   0:00  \_ [migration/4]
      2      36       0       0 ?             -1 S        0   0:00  \_ [ksoftirqd/4]
      2      38       0       0 ?             -1 I<       0   0:00  \_ [kworker/4:0H-k
      2      39       0       0 ?             -1 S        0   0:00  \_ [cpuhp/5]
```

25

26 **ps**: Command used to list the current processes running in our system.

27 **history:** Shows the list of previously executed commands.



```
 1114  sudo brightinc -n 180
 1115  clear
 1116  vim /bin/brightinc
 1117  clear
 1118  sudo brightinc -n 180
 1119  echo 180 > /sys/class/backlight/amdgpu_bl0/brightness
 1120  vim /bin/brightinc
 1121  clear
 1122  sudo brightinc show 180
 1123  cd /sys/class/backlight/amdgpu_bl0
 1124  cd /sys/class/
 1125  ls
 1126  cd backlight/
 1127  ls
 1128  vim /bin/brightinc
 1129  sudo vim /bin/brightinc
 1130  sudo brightinc show 180
 1131  sudo brightinc -n 180
 1132  nvidia-smi
 1133  clear
 1134  sudo brightinc -n 180
 1135  echo 180 > /sys/class/backlight/amdgpu_bl0/brightness
 1136  sudo echo 180 > /sys/class/backlight/amdgpu_bl0/brightness
 1137  sudo su
 1138  kill -9 $(ps aux | grep 'msedge' | awk '{print $2}')
 1139  sudo brightinc -n 180
 1140  vim /bin/brightinc
 1141  sudo vim /bin/brightinc
 1142  sudo brightinc -n 180
```

Experiment-4

Aim: Study of vi editor.

Theory:

The VI editor is the most popular and classic text editor in the Linux family. Below, are some reasons which make it a widely used editor –

1 It is available in almost all Linux Distributions

2 It works the same across different platforms and Distributions

3 It is user-friendly. Hence, millions of Linux users love it and use it for their editing needs

VI Command mode:

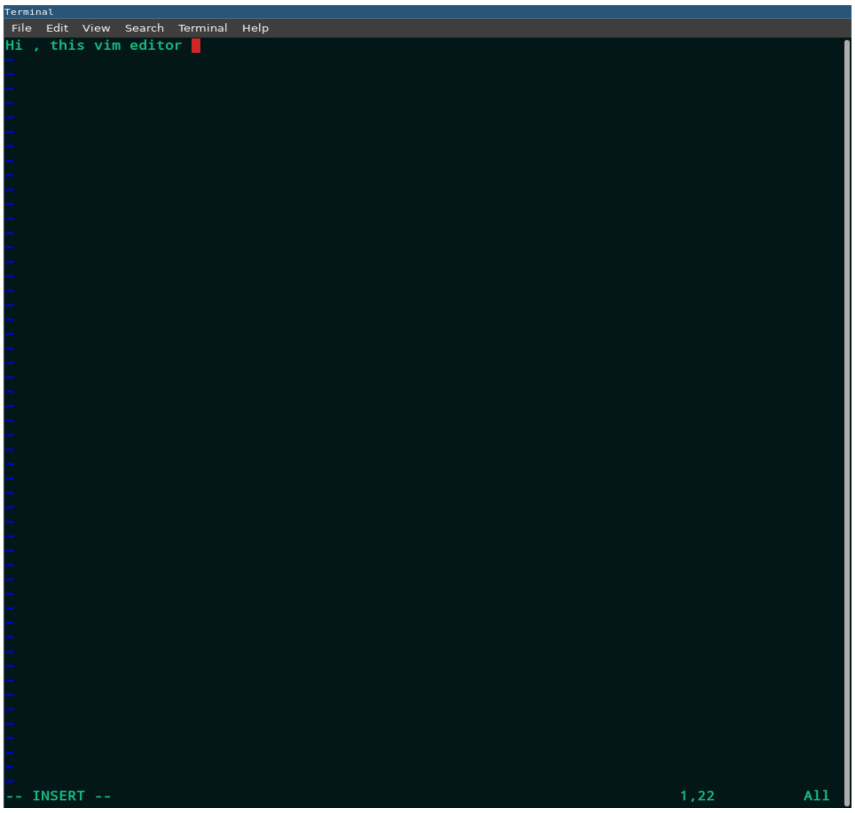
- The vi editor opens in this mode, and it only understands commands
- In this mode, you can, move the cursor and cut, copy, paste the text
- This mode also saves the changes you have made to the file
- Commands are case sensitive. You should use the right letter case.

VI Editor Insert mode:

- This mode is for inserting text in the file.
- You can switch to the Insert mode from the command mode by pressing 'i' on the keyboard
- Once you are in Insert mode, any key would be taken as an input for the file on which you are currently working.
- To return to the command mode and save the changes you have made you need to press the Esc key

Procedure:

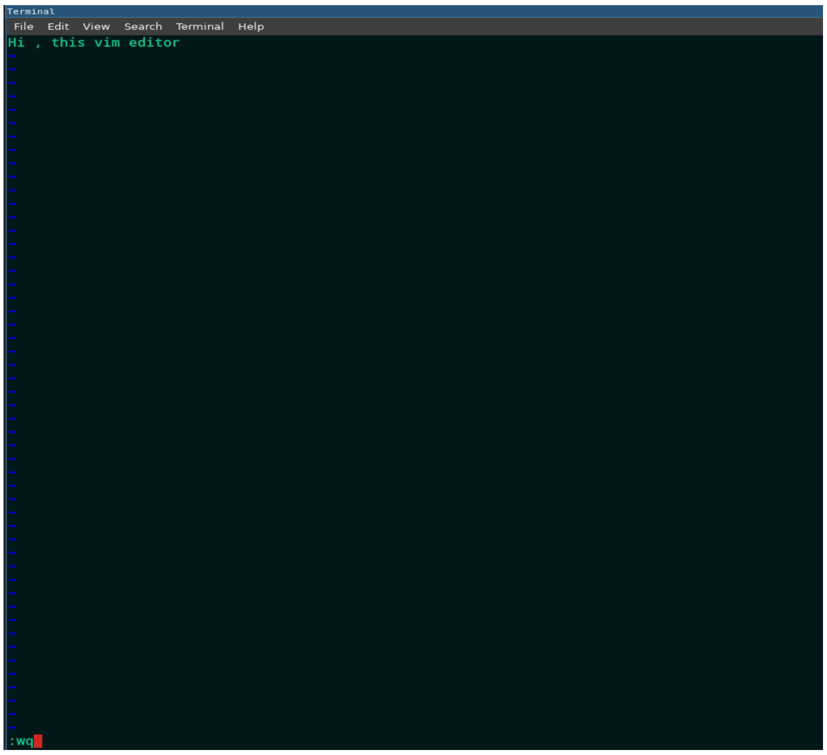
1 First, we create a file using ‘vim test’



```
Hi , this vim editor
-- INSERT --                                     1,22          All
```

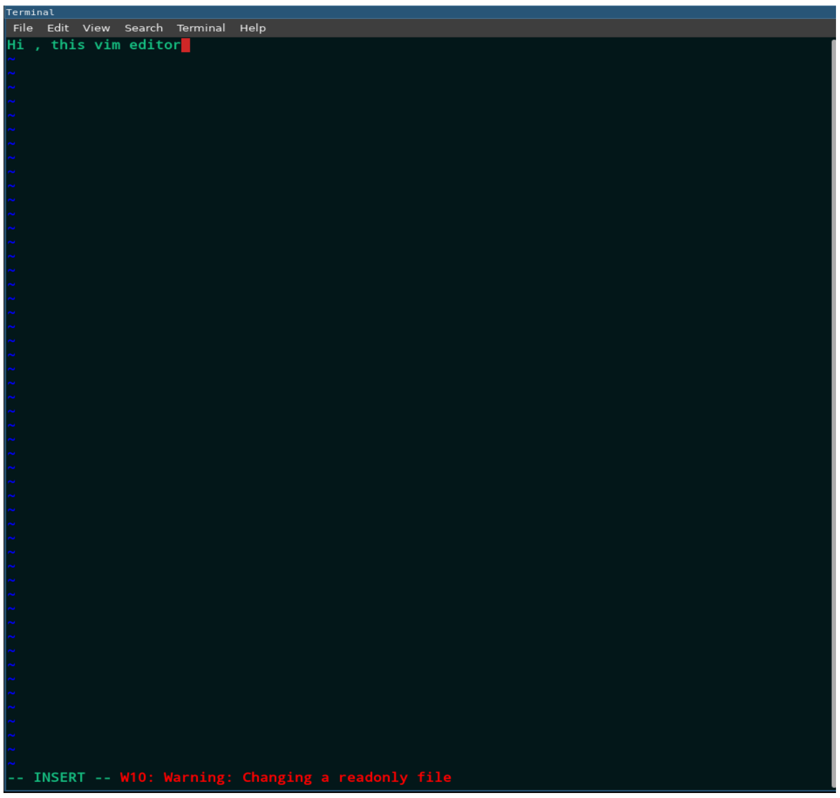
2 Press i to start the INSERT MODE of vi edtor

3 To go to the COMMAND MODE press Esc key, then, to get out of vi, type :wq (w is to



```
Hi , this vim editor
~
~
~
:wq
```

save the file, while q is to quit)



```
Hi , this vim editor
~
~
~
-- INSERT -- W10: Warning: Changing a readonly file
```

4 Now we'll type 'vi -R parv.test'

5 Now we can see the file in read only mode, to exit from here without saving any changes type :qa!

6 NOTE: for doing undo in INSERT MODE in vi editor, press CTRL+U

Experiment-5

Aim: Implementation of docker on Linux operating system.

Theory:

Docker is a set of platform as a service (PaaS) products that use OS-level virtualization to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, libraries and configuration files; they can communicate with each other through well-defined channels. Because all of the containers share the services of a single operating system kernel, they use fewer resources than virtual machines.
The service has both free and premium tiers. The software that hosts the containers is called Docker Engine.

Components

The Docker software as a service offering consists of three components:

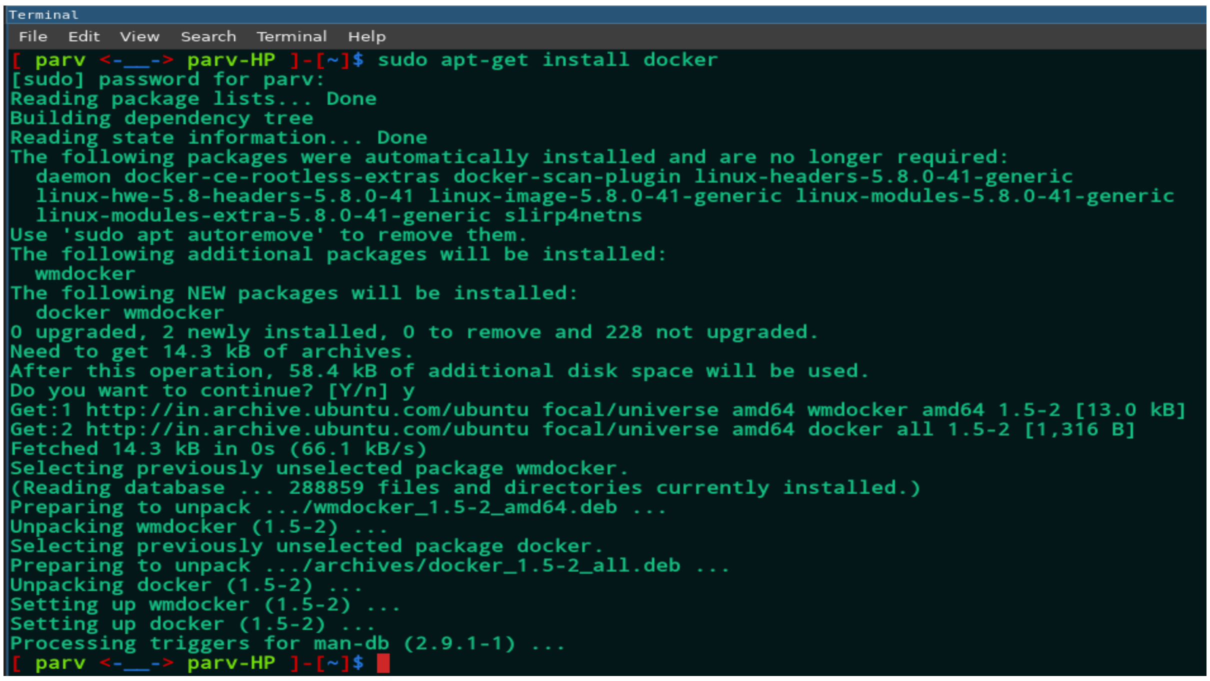
- Software: The Docker daemon, called dockerd, is a persistent process that manages Docker containers and handles container objects. The daemon listens for requests sent via the Docker Engine API. The Docker client program, called docker, provides a command-line interface (CLI), that allows users to interact with Docker daemons.
- Objects: Docker objects are various entities used to assemble an application in Docker. The main classes of Docker objects are images, containers, and services.
    - A Docker container is a standardized, encapsulated environment that runs applications. A container is managed using the Docker API or CLI.
    - A Docker image is a read-only template used to build containers. Images are used to store and ship applications.
    - A Docker service allows containers to be scaled across multiple Docker daemons. The result is known as a swarm, a set of cooperating daemons that communicate through the Docker API.
- Registries: A Docker registry is a repository for Docker images. Docker clients connect to registries to download ("pull") images for use or upload ("push") images that they have built. Registries can be public or private. Two main public registries are Docker Hub and Docker Cloud. Docker Hub is the default registry where Docker looks for images.Docker registries also allow the creation of notifications based on events.

Tools

- Docker Compose is a tool for defining and running multi-container Docker applications. It uses YAML files to configure the application's services and performs the creation and start-up process of all the containers with a single command. The docker-compose CLI utility allows users to run commands on multiple containers at once, for example, building images, scaling containers, running containers that were stopped, and more. Commands related to image manipulation, or user-interactive options, are not relevant in Docker Compose because they address one container.[31] The docker-compose.yml file is used to define an application's services and includes various configuration options. For example, the build option defines configuration options such as the Dockerfile path, the command option allows one to override default Docker commands, and more. The first public beta version of Docker Compose (version 0.0.1) was released on December 21, 2013.The first production-ready version (1.0) was made available on October 16, 2014.

- Docker Swarm provides native clustering functionality for Docker containers, which turns a group of Docker engines into a single virtual Docker engine. In Docker 1.12 and higher, Swarm mode is integrated with Docker Engine. The docker swarm CLI utility allows users to run Swarm containers, create discovery tokens, list nodes in the cluster, and more. The docker node CLI utility allows users to run various commands to manage nodes in a swarm, for example, listing the nodes in a swarm, updating nodes, and removing nodes from the swarm. Docker manages swarms using the Raft consensus algorithm. According to Raft, for an update to be performed, the majority of Swarm nodes need to agree on the update.
- Docker Volume If you copy or create a file in a container, when you stop that container that file (and any other files created or copied) will be deleted. Docker Volume is a solution for this issue.

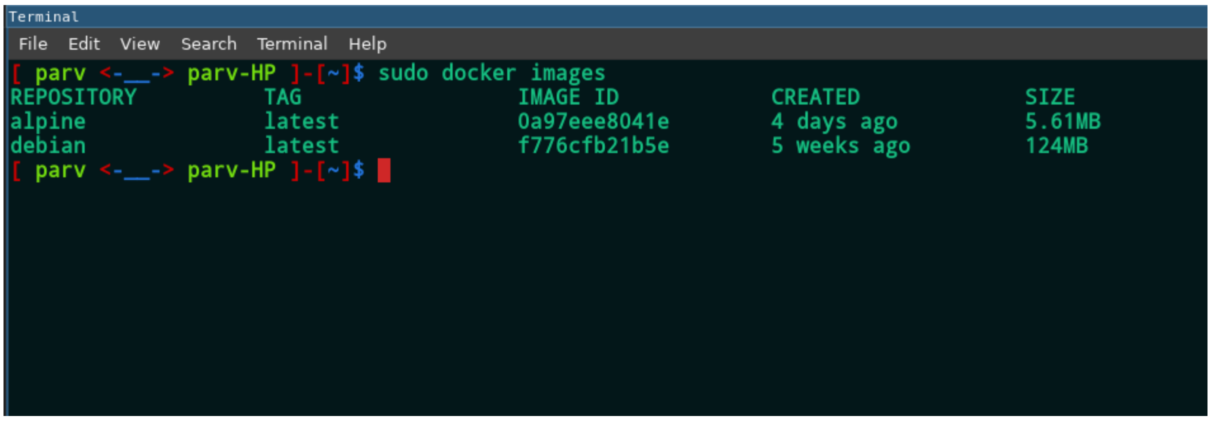
Procedure:



```
[ parv <-__-> parv-HP ]-[~]$ sudo apt-get install docker
[sudo] password for parv:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  daemon docker-ce-rootless-extras docker-scan-plugin linux-headers-5.8.0-41-generic
  linux-hwe-5.8-headers-5.8.0-41 linux-image-5.8.0-41-generic linux-modules-5.8.0-41-generic
  linux-modules-extra-5.8.0-41-generic slirp4netns
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  wmdocker
The following NEW packages will be installed:
  docker wmdocker
0 upgraded, 2 newly installed, 0 to remove and 228 not upgraded.
Need to get 14.3 kB of archives.
After this operation, 58.4 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://in.archive.ubuntu.com/ubuntu focal/universe amd64 wmdocker amd64 1.5-2 [13.0 kB]
Get:2 http://in.archive.ubuntu.com/ubuntu focal/universe amd64 docker all 1.5-2 [1,316 B]
Fetched 14.3 kB in 0s (66.1 kB/s)
Selecting previously unselected package wmdocker.
(Reading database ... 288859 files and directories currently installed.)
Preparing to unpack .../wmdocker_1.5-2_amd64.deb ...
Unpacking wmdocker (1.5-2) ...
Selecting previously unselected package docker.
Preparing to unpack .../archives/docker_1.5-2_all.deb ...
Unpacking docker (1.5-2) ...
Setting up wmdocker (1.5-2) ...
Setting up docker (1.5-2) ...
Processing triggers for man-db (2.9.1-1) ...
[ parv <-__-> parv-HP ]-[~]$
```

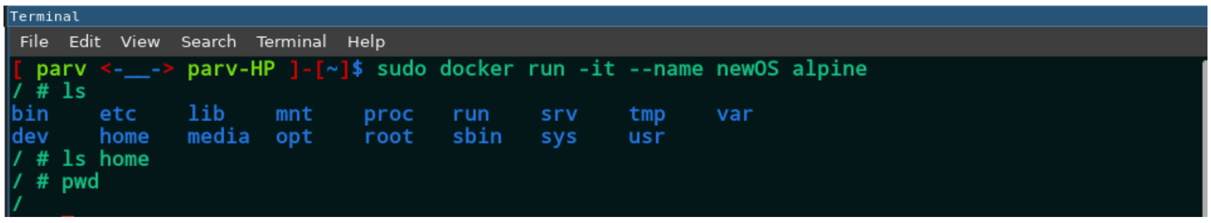
7 Download docker using the **apt-get install** command.

8 Pull the image from the docker hub. Using the s**udo docker pull <image name>.** One can list the images which are there on the system using **sudo docker images.**



```
[ parv <-__-> parv-HP ]-[~]$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
alpine              latest              0a97eee8041e        4 days ago          5.61MB
debian              latest              f776cfb21b5e        5 weeks ago         124MB
[ parv <-__-> parv-HP ]-[~]$ 
```

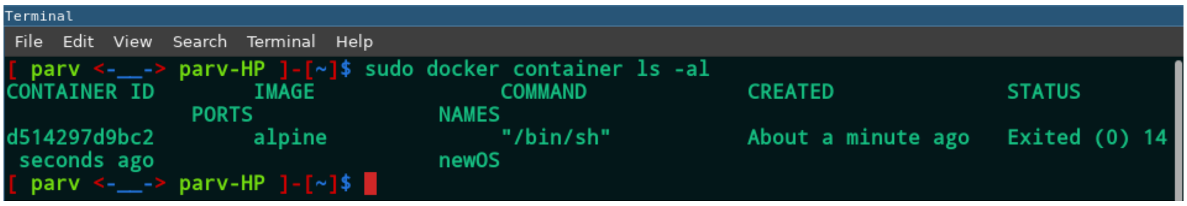
9 Now run the container for the image using the **sudo docker run -it - -name <any name> <image name><command>.** This commands also pulls the image it's not already present



```
[ parv <-__-> parv-HP ]-[~]$ sudo docker run -it --name newOS alpine
/ # ls
bin    etc    lib    mnt    proc   run    srv    tmp    var
dev    home   media  opt    root   sbin   sys    usr
/ # ls home
/ # pwd
/
```

in our system.

10 We can remove the container by first stropping the container using **sudo docker container stop <container-name >.** After stopping the container , then we can remove it by using the following command: **sudo docker container rm <container-name or id>.**



```
[ parv <-__-> parv-HP ]-[~]$ sudo docker container ls -al
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
          PORTS               NAMES
d514297d9bc2        alpine              "/bin/sh"           About a minute ago  Exited (0) 14
 seconds ago                  newOS
[ parv <-__-> parv-HP ]-[~]$ 
```

Experiment-6

Aim: Study of Bash shell, Bourne shell and C shell in Unix/Linux operating system.

Theory:

Shell: SHELL is a program which provides the interface between the user and an operating system. When the user logs in OS starts a shell for user.

Types of Shells in Linux/UNIX:

Bourne shell:

The Bourne shell, called "sh," is one of the original shells, developed for Unix computers by Stephen Bourne at AT&T's Bell Labs in 1977. Its long history of use means many software developers are familiar with it. It offers features such as input and output redirection, shell scripting with string and integer variables, and condition testing and looping.

Bash shell:

The popularity of sh motivated programmers to develop a shell that was compatible with it, but with several enhancements. Linux systems still offer the sh shell, but "bash" -- the "Bourne-again Shell," based on sh -- has become the new default standard. One attractive feature of bash is its ability to run sh shell scripts unchanged. Shell scripts are complex sets of commands that automate programming and maintenance chores; being able to reuse these scripts saves programmers time. Conveniences not present with the original Bourne shell include command completion and a command history.

C shell:

Developers have written large parts of the Linux operating system in the C and C++ languages. Using C syntax as a model, Bill Joy at Berkeley University developed the "C-shell," csh, in 1978. Ken Greer, working at Carnegie-Mellon University, took csh concepts a step forward with a new shell, tcsh, which Linux systems now offer. Tcsh fixed problems in csh and added command completion, in which the shell makes educated "guesses" as you type, based on your system's directory structure and files. Tcsh does not run bash scripts, as the two have substantial differences.

Zsh (Z-Shell)

Zsh or Z-Shell is a modern-day shell designed to be innovative and interactive by offering unique features in addition to the features of other Unix or GNU Linux shells, such as ksh, tcsh, Bash, etc. This open-source shell offers scripting features and is customisable, easy-to-use, and offers command completion, spelling correction, and more. If you want an advanced Linux shell, go for the Zsh shell. The installation procedure for Zsh is also effortless. In Zsh, you can even use open-source frameworks, such as oh-my-zsh customisable plugins and options.
The Zsh shell offers various features for Linux, including:

- Fantastic auto-completion functionality for files and paths.
- Command history sharing mechanism.
- Concept index, functions index, key index, and variable index.
- Various interactive features, such as smart escaping, spelling correction, recursive globbing, and more.

Ksh (Korn Shell)

The full form of Ksh is Korn shell because it was designed by David G. Korn. Ksh is a powerful, interactive command language and high-level programming language that can compete with other Unix shells. The development of the Korn shell was inspired by the interactivity of the C shell interactivity and the productivity of the Bash shell.
The following is a list of some of the features available in the Korn shell:

- Unique options to improve performance and capability, as shellcode is stored in the memory.
- Ctrl+Z tweak that can quickly stop a running job, and you can continue to execute your commands if they were initiated with fg (foreground) or bg (background) commands.
- Contains various advanced features for fast-paced executions.
- Includes advanced command-line editing features to edit commands more easily.

Tcsh (Tenex C Shell)

The full form of Tcsh is Tenex C Shell. This shell is an improved version of the C shell and is used as a shell script command processor and interactive login shell. Tcsh offers multiple options, including a command-line editor, job control, spellcheck support, configurable command-line completion, a modernized history mechanism, and more. This open-source shell for Linux is best for programmers because its syntax is like the C language, so these users can use the scripting features in Tcsh without any knowledge of Bash.
The features offered by Tcsh include the following:

- Filename completion and programmable words.
- C-like syntax and a command-line editor.
- FreeBSD operating system to power up modern servers.
- Job control and spelling correction features

Experiment-7

Aim: Study of Unix/Linux file system (tree structure).

Theory:

A Linux file system is a structured collection of files on a disk drive or a partition. A partition is a segment of memory and contains some specific data. In our machine, there can be various partitions of the memory. Generally, every partition contains a file system.

The general-purpose computer system needs to store data systematically so that we can easily access the files in less time. It stores the data on hard disks (HDD) or some equivalent storage type. There may be below reasons for maintaining the file system:

- Primarily the computer saves data to the RAM storage; it may lose the data if it gets turned off. However, there is non-volatile RAM (Flash RAM and SSD) that is available to maintain the data after the power interruption.
- Data storage is preferred on hard drives as compared to standard RAM as RAM costs more than disk space. The hard disks costs are dropping gradually comparatively the RAM.

The Linux file system contains the following sections:

- The root directory (/)
- A specific data storage format (EXT3, EXT4, BTRFS, XFS and so on)
- A partition or logical volume having a particular file system.

From top to bottom, the directories you are seeing are as follows:

/bin

/bin is the directory that contains binaries, that is, some of the applications and programs you can run.

/boot

The /boot directory contains files required for starting your system.

/dev

/dev contains device files. Many of these are generated at boot time or even on the fly. For example, if you plug in a new webcam or a USB pendrive into your machine, a new device entry will automagically pop up here.

/etc

/etc gets its name from the earliest Unixes and it was literally "et cetera". Nowadays, it would be more appropriate to say that etc stands for "Everything to configure," as it contains most, if not all system-wide configuration files.

/home

/home is where you will find your users' personal directories.

/lib

/lib is where libraries live. Libraries are files containing code that your applications can use. They contain snippets of code that applications use to draw windows on your desktop, control peripherals, or send files to your hard disk.
it contains the all-important kernel modules. The kernel modules are drivers that make things like your video card, sound card, WiFi, printer, and so on, work.

/media

The /media directory is where external storage will be automatically mounted when you plug it in and try to access it.

/mnt

The /mnt directory, however, is a bit of remnant from days gone by. This is where you would manually mount storage devices or partitions. It is not used very often nowadays.

/opt

The /opt directory is often where software you compile (that is, you build yourself from source code and do not install from your distribution repositories) sometimes lands. Applications will end up in the /opt/bin directory and libraries in the /opt/lib directory.

/proc
/proc, like /dev is a virtual directory. It contains information about your computer, such as information about your CPU and the kernel your Linux system is running. As with /dev, the files and directories are generated when your computer starts, or on the fly, as your system is running and things change.

/root
/root is the home directory of the superuser (also known as the "Administrator") of the system.

/run

/run is another new directory. System processes use it to store temporary data for their own nefarious reasons.

/sbin

/sbin is similar to /bin, but it contains applications that only the superuser (hence the initial s) will need. You can use these applications with the sudo command that temporarily concedes you superuser powers on many distributions.

/usr

The /usr directory was where users' home directories were originally kept back in the early days of UNIX. However, now /home is where users kept their stuff as we saw above. These days, /usr contains a mishmash of directories which in turn contain applications, libraries, documentation, wallpapers, icons and a long list of other stuff that need to be shared by applications and services.

/srv

The /srv directory contains data for servers. If you are running a web server from your Linux box, your HTML files for your sites would go into /srv/http (or /srv/www). If you were running an FTP server, your files would go into /srv/ftp.

/sys

/sys is another virtual directory like /proc and /dev and also contains information from devices connected to your computer.
In some cases you can also manipulate those devices. But to do that you have to become superuser.

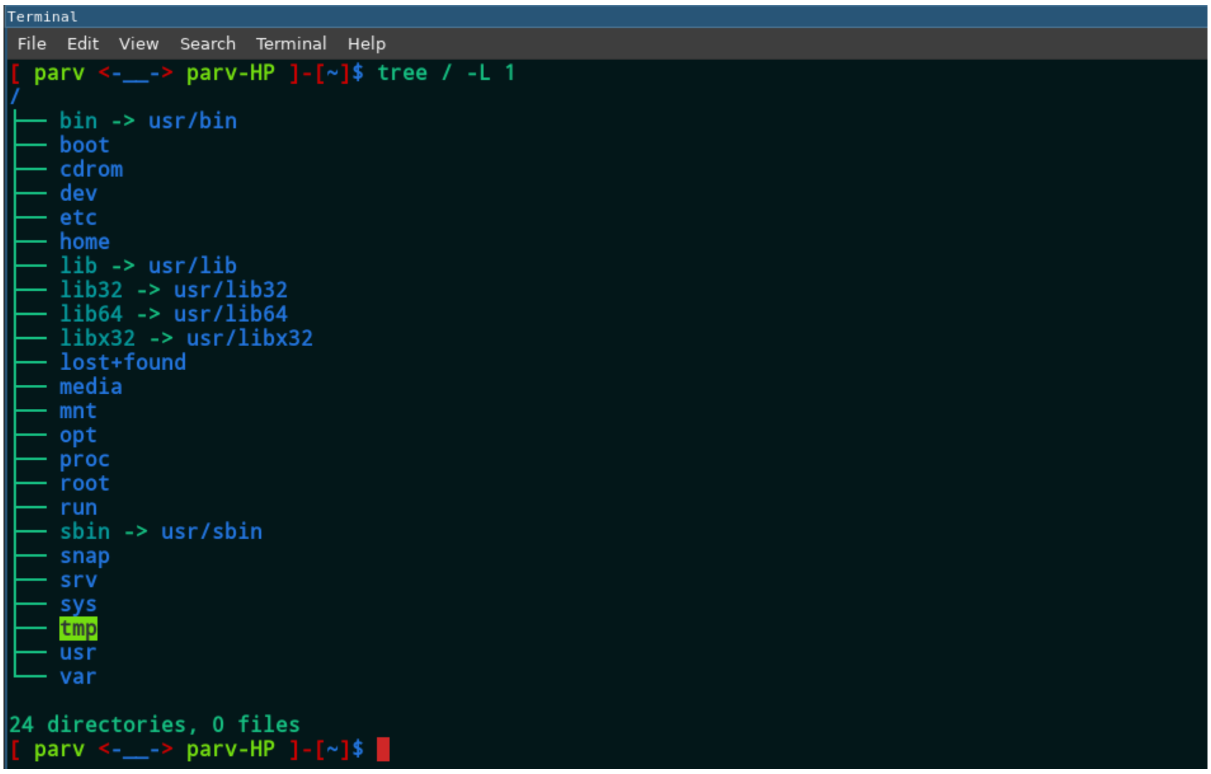
/tmp

/tmp contains temporary files, usually placed there by applications that you are running. The files and directories often (not always) contain data that an application doesn't need right now, but may need later on.
You can also use /tmp to store your own temporary files — /tmp is one of the few directories hanging off / that you can actually interact with without becoming superuser.

/var

/var was originally given its name because its contents was deemed variable, in that it changed frequently. Today it is a bit of a misnomer because there are many other directories that also contain data that changes frequently, especially the virtual directories we saw above.
Be that as it may, /var contains things like logs in the /var/log subdirectories. Logs are files that register events that happen on the system.

Tree structure of Linux File System from the Root (1 level deep)



```
[ parv <-__-> parv-HP ]-[~]$ tree / -L 1
/
├── bin -> usr/bin
├── boot
├── cdrom
├── dev
├── etc
├── home
├── lib -> usr/lib
├── lib32 -> usr/lib32
├── lib64 -> usr/lib64
├── libx32 -> usr/libx32
├── lost+found
├── media
├── mnt
├── opt
├── proc
├── root
├── run
├── sbin -> usr/sbin
├── snap
├── srv
├── sys
├── tmp
├── usr
└── var

24 directories, 0 files
[ parv <-__-> parv-HP ]-[~]$
```

Experiment-8

Aim: Study of .bashrc, /etc/bashrc and Environment variables.

Theory:

The /etc/bashrc is executed for both interactive and non-interactive shells. /etc/bashrc or /etc/bash.bashrc is the systemwide bash per-interactive-shell startup file. Is is used system wide functions and aliases. However, environment stuff goes in /etc/profile file.the /etc/profile is executed only for interactive shells .bashrc is a shell script that Bash runs whenever it is started interactively. It initialises an interactive shell session. .bashrc runs on every interactive shell launch.

Following is the partial list of important environment variables :-

1 **DISPLAY :** Contains the identifier for the display that X11 programs should use by default.

2 **HOME :** Indicates the home directory of the current user: the default argument for the cd built-in command.

3 **IFS :** Indicates the Internal Field Separator that is used by the parser for word splitting after expansion.

4 **LANG :** LANG expands to the default system locale; LC_ALL can be used to override this. For example, if its value is pt_BR, then the language is set to (Brazilian) Portuguese and the locale to Brazil.

5 **LD_LIBRARY_PATH :** On many Unix systems with a dynamic linker, contains a colonseparated list of directories that the dynamic linker should search for shared objects when building a process image after exec, before searching in any other directories.

6 **PATH :** Indicates search path for commands. It is a colon-separated list of directories in which the shell looks for commands.

7 **PWD :** Indicates the current working directory as set by the cd command.

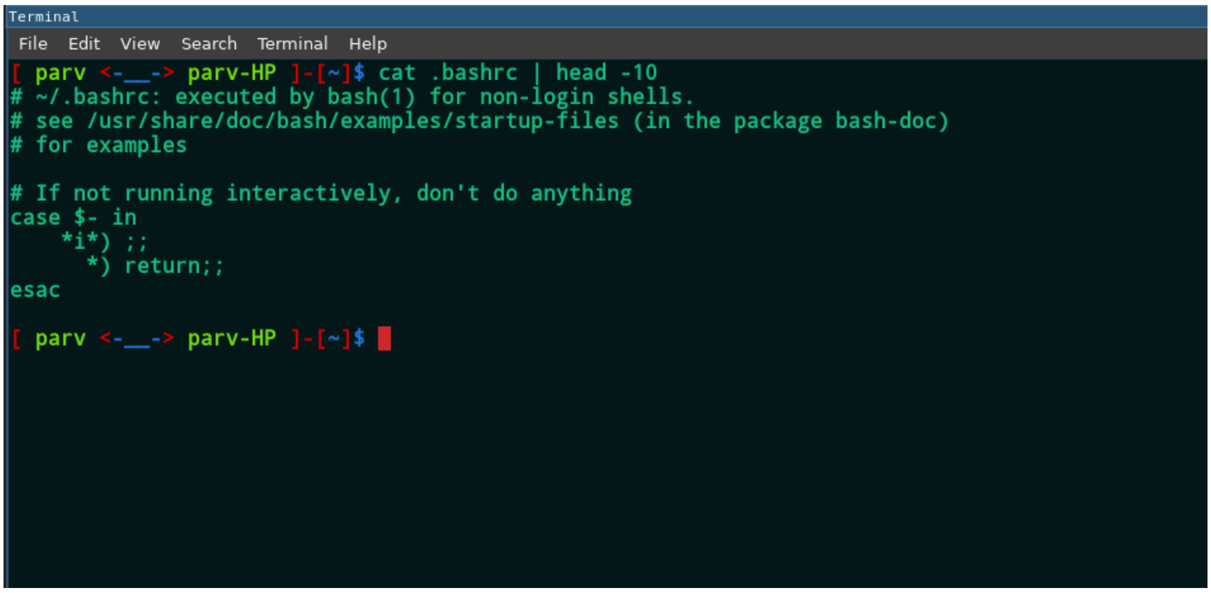
8 **RANDOM :** Generates a random integer between 0 and 32,767 each time it is referenced.

9 **SHLVL :** Increments by one each time an instance of bash is started. This variable is useful for determining whether the built-in exit command ends the current session.

10 **TERM :** Refers to the display type

11 **VZ :** Refers to Time zone. It can take values like GMT, AST, etc.

12 **UID :** Expands to the numeric user ID of the current user, initialized at shell startup.



```
[ parv <-__-> parv-HP ]-[~]$ cat .bashrc | head -10
# ~/.bashrc: executed by bash(1) for non-login shells.
# see /usr/share/doc/bash/examples/startup-files (in the package bash-doc)
# for examples

# If not running interactively, don't do anything
case $- in
    *i*) ;;
      *) return;;
esac

[ parv <-__-> parv-HP ]-[~]$ 
```

Result / Outputs:

The .bashrc file:

Experiment-9

AIM: Write a shell script program to display list of user currently logged in.

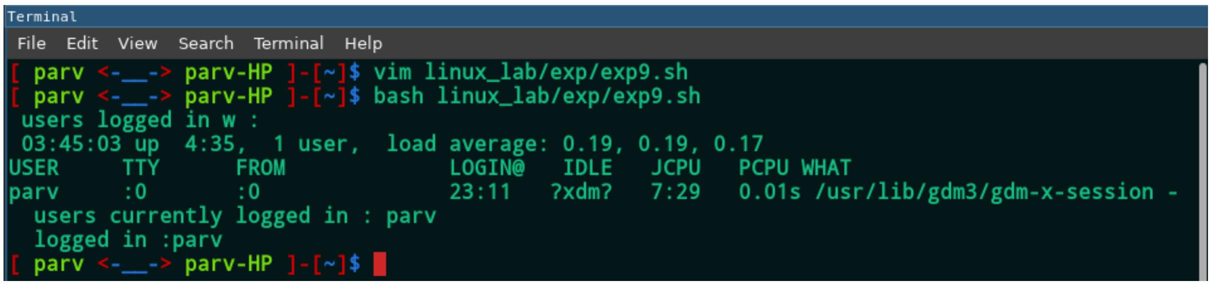
Theory:

The shell can be defined as a command interpreter within an operating system like Linux/GNU or Unix. It is a program that runs other programs.
The shell facilitates every user of the computer as an interface to the Unix/GNU Linux system. Hence, the user can execute different tools/utilities or commands with a few input data.

Script:

```
echo " users logged in w : "
w

echo "  users currently logged in : $(whoami) "

echo "  logged in :$(id -un) "
```

Output:



```
[ parv <-__-> parv-HP ]-[~]$ vim linux_lab/exp/exp9.sh
[ parv <-__-> parv-HP ]-[~]$ bash linux_lab/exp/exp9.sh
 users logged in w :
 03:45:03 up  4:35,  1 user,  load average: 0.19, 0.19, 0.17
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
parv     :0       :0               23:11   ?xdm?   7:29   0.01s /usr/lib/gdm3/gdm-x-session -
  users currently logged in : parv
  logged in :parv
[ parv <-__-> parv-HP ]-[~]$
```

Experiment-10

Aim: Write a shell script program to display “HELLO WORLD”.

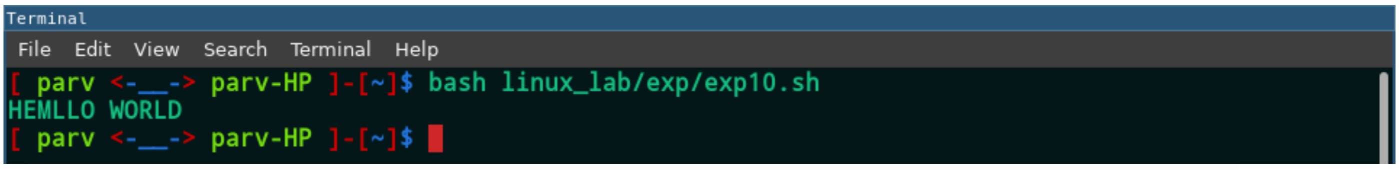
Theory:

The shell can be defined as a command interpreter within an operating system like Linux/GNU or Unix. It is a program that runs other programs.
The shell facilitates every user of the computer as an interface to the Unix/GNU Linux system. Hence, the user can execute different tools/utilities or commands with a few input data.

Script:

```
#! /bin/bash
var="Hello World"

# print it
echo "$var"
```

Result/Output:



```
[ parv <-__-> parv-HP ]-[~]$ bash linux_lab/exp/exp10.sh
HEMLLO WORLD
[ parv <-__-> parv-HP ]-[~]$ 
```

Experiment-11

Aim: Write a shell script program to develop a scientific calculator.

Theory:

The shell can be defined as a command interpreter within an operating system like Linux/GNU or Unix. It is a program that runs other programs.
The shell facilitates every user of the computer as an interface to the Unix/GNU Linux system. Hence, the user can execute different tools/utilities or commands with a few input data.

Script/Code:

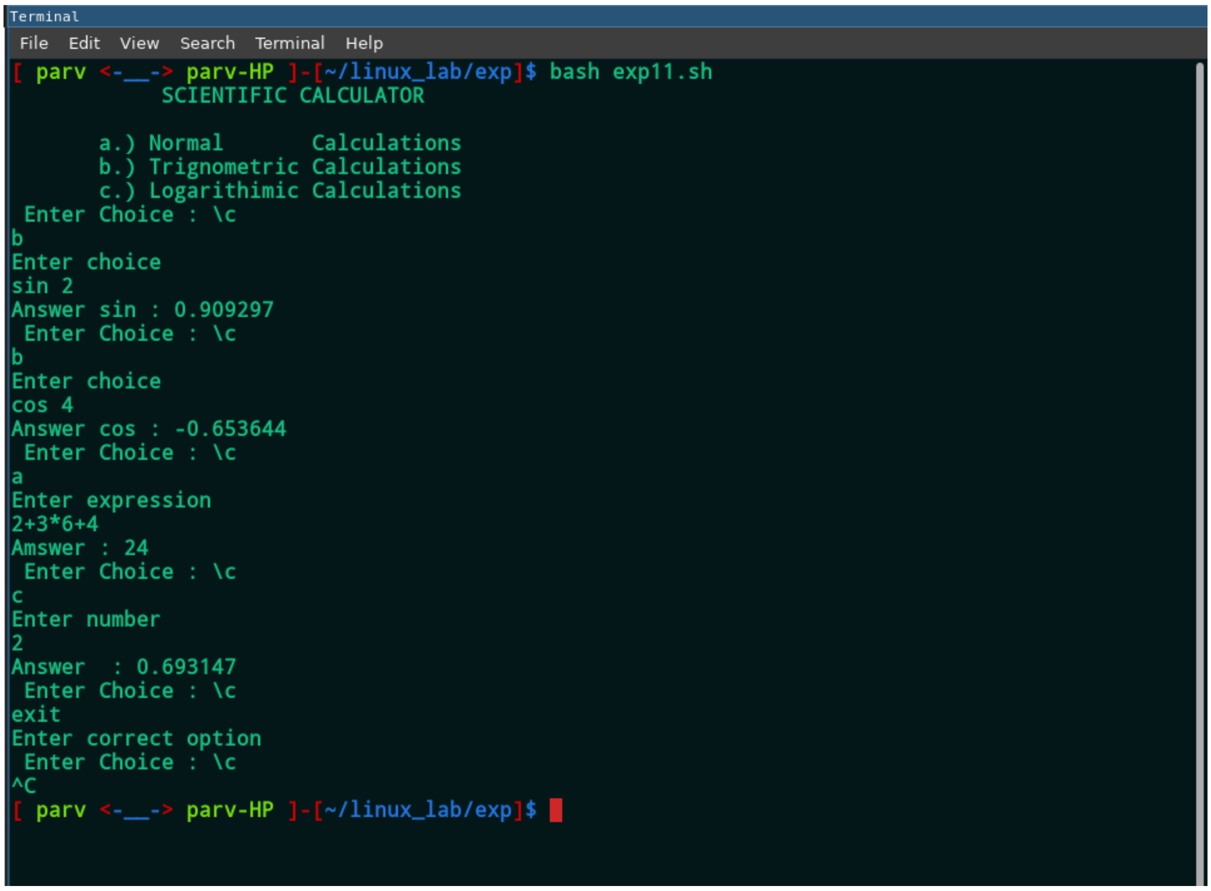
Trig.cpp(C++ Code):

```
#include<math.h>
#include<stdlib.h>
#include<string>
#include<iostream>
using namespace std;
int main(int argc , char **argv)
{

string temp(argv[1]);

if(!temp.compare("-s") )
{


cout<<sin(atof(argv[2]));
}

else if(!temp.compare("-c"))
{

cout<<cos(atof(argv[2]));
}

else if(!temp.compare("-t"))
{

cout<<tan(atof(argv[2]));
}
else if(!temp.compare("-ln"))
{
cout<<log(atof(argv[2]));
}

return 0;
}
```

Calculator.sh

```
cat << MENU
        SCIENTIFIC CALCULATOR

       a.) Normal       Calculations
       b.) Trignometric Calculations
       c.) Logarithimic Calculations
MENU

while true
do

echo ' Enter Choice : \c '

read choice

case $choice in
    a) echo "Enter expression"
       read exp
          echo "Amswer : $((exp))"
       ;;
    b) echo "Enter choice "
       read func number
       if [ func="sin" ];
       then
           echo "Answer : $(trig -s $number)"
       fi

       if [ func="cos" ];
       then
           echo "Answer : $(trig -c $number)"
       fi
       if [ func="tan" ];
       then
          echo "Answer : $(trig -t $number)"
       fi ;;
       c)  echo "Enter number"
               read num

         echo "Answer  : $(trig -ln $num)"


            ;;

       *) echo "enter correct option"
        ;;
esac
done
```

Result/Output:



```
[ parv <-__-> parv-HP ]-[~/linux_lab/exp]$ bash exp11.sh
            SCIENTIFIC CALCULATOR

       a.) Normal       Calculations
       b.) Trignometric Calculations
       c.) Logarithimic Calculations
 Enter Choice : \c
b
Enter choice
sin 2
Answer sin : 0.909297
 Enter Choice : \c
b
Enter choice
cos 4
Answer cos : -0.653644
 Enter Choice : \c
a
Enter expression
2+3*6+4
Amswer : 24
 Enter Choice : \c
c
Enter number
2
Answer  : 0.693147
 Enter Choice : \c
exit
Enter correct option
 Enter Choice : \c
^C
[ parv <-__-> parv-HP ]-[~/linux_lab/exp]$
```

Experiment-12

Aim:

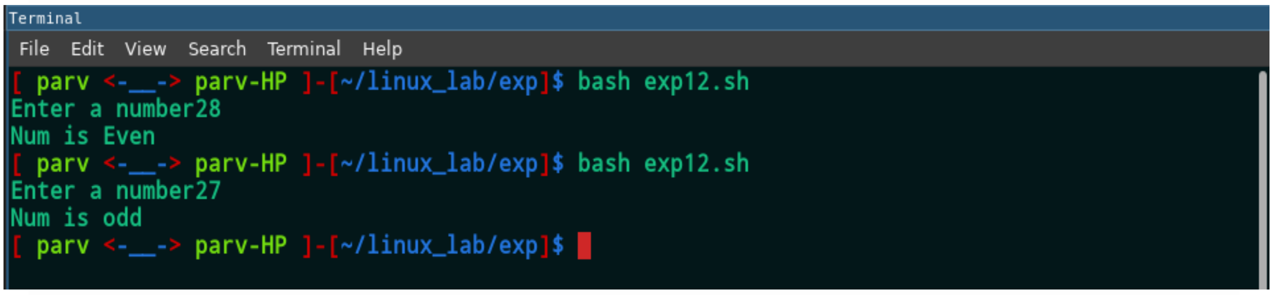
Theory:

The shell can be defined as a command interpreter within an operating system like Linux/GNU or Unix. It is a program that runs other programs.
The shell facilitates every user of the computer as an interface to the Unix/GNU Linux system. Hence, the user can execute different tools/utilities or commands with a few input data.

Script:

```
read -p "Enter a number" num

if [[ $((num%2)) == 0 ]]; then

        echo "Num is Even"

else

        echo "Num is odd"

fi
```

Result/Output:



```
[ parv <-__-> parv-HP ]-[~/linux_lab/exp]$ bash exp12.sh
Enter a number28
Num is Even
[ parv <-__-> parv-HP ]-[~/linux_lab/exp]$ bash exp12.sh
Enter a number27
Num is odd
[ parv <-__-> parv-HP ]-[~/linux_lab/exp]$
```

Experiment-13

Aim: Shell script Program to search whether element is present is in the list or not.

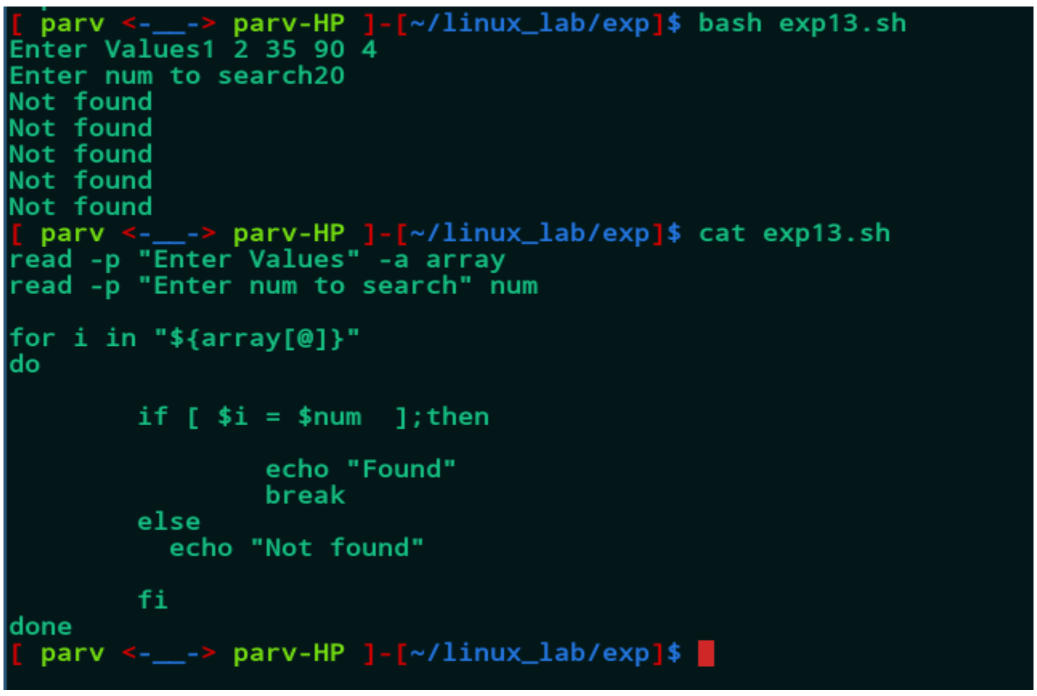
Theory:

The shell can be defined as a command interpreter within an operating system like Linux/GNU or Unix. It is a program that runs other programs.
The shell facilitates every user of the computer as an interface to the Unix/GNU Linux system. Hence, the user can execute different tools/utilities or commands with a few input data.

Script:

```
read -p "Enter Values" -a array
read -p "Enter num to search" num

for i in "${array[@]}"
do

   if [ $i = $num  ];then

         echo "Found"
         break
   else
     echo "Not found"

   fi
done
```

Result/Output:



```
[ parv <-___-> parv-HP ]-[~/linux_lab/exp]$ bash exp13.sh
Enter Values1 2 35 90 4
Enter num to search20
Not found
Not found
Not found
Not found
Not found
[ parv <-___-> parv-HP ]-[~/linux_lab/exp]$ cat exp13.sh
read -p "Enter Values" -a array
read -p "Enter num to search" num

for i in "${array[@]}"
do

        if [ $i = $num  ];then

                echo "Found"
                break
        else
          echo "Not found"

        fi
done
[ parv <-___-> parv-HP ]-[~/linux_lab/exp]$
```

Experiment-14

Aim: Shell script program to check whether given file is a directory or not.

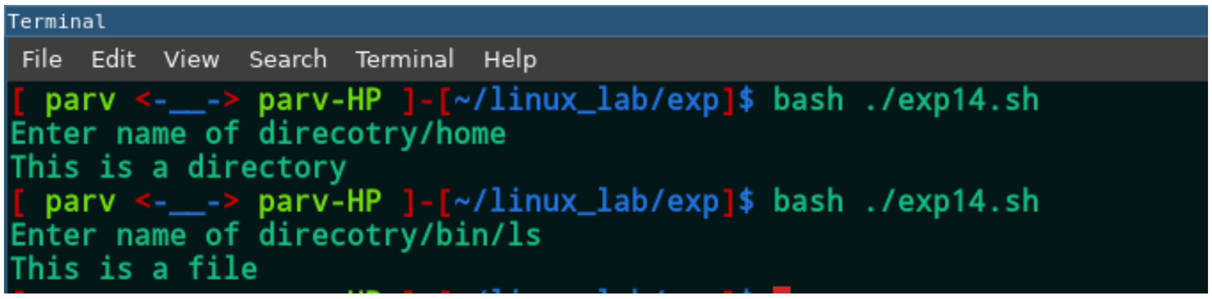
Theory:

The shell can be defined as a command interpreter within an operating system like Linux/GNU or Unix. It is a program that runs other programs.
The shell facilitates every user of the computer as an interface to the Unix/GNU Linux system. Hence, the user can execute different tools/utilities or commands with a few input data.

Script:

```
read -p "Enter name of direcotry" name

if [ -d $name ]; then
        echo "This is a directory"
elif [ -f $name ];then
        echo "This is a file"
fi
```

Result/Output:



```
[ parv <-__-> parv-HP ]-[~/linux_lab/exp]$ bash ./exp14.sh
Enter name of direcotry/home
This is a directory
[ parv <-__-> parv-HP ]-[~/linux_lab/exp]$ bash ./exp14.sh
Enter name of direcotry/bin/ls
This is a file
```

Experiment-16

Aim: Shell script program to copy contents of one file to another.

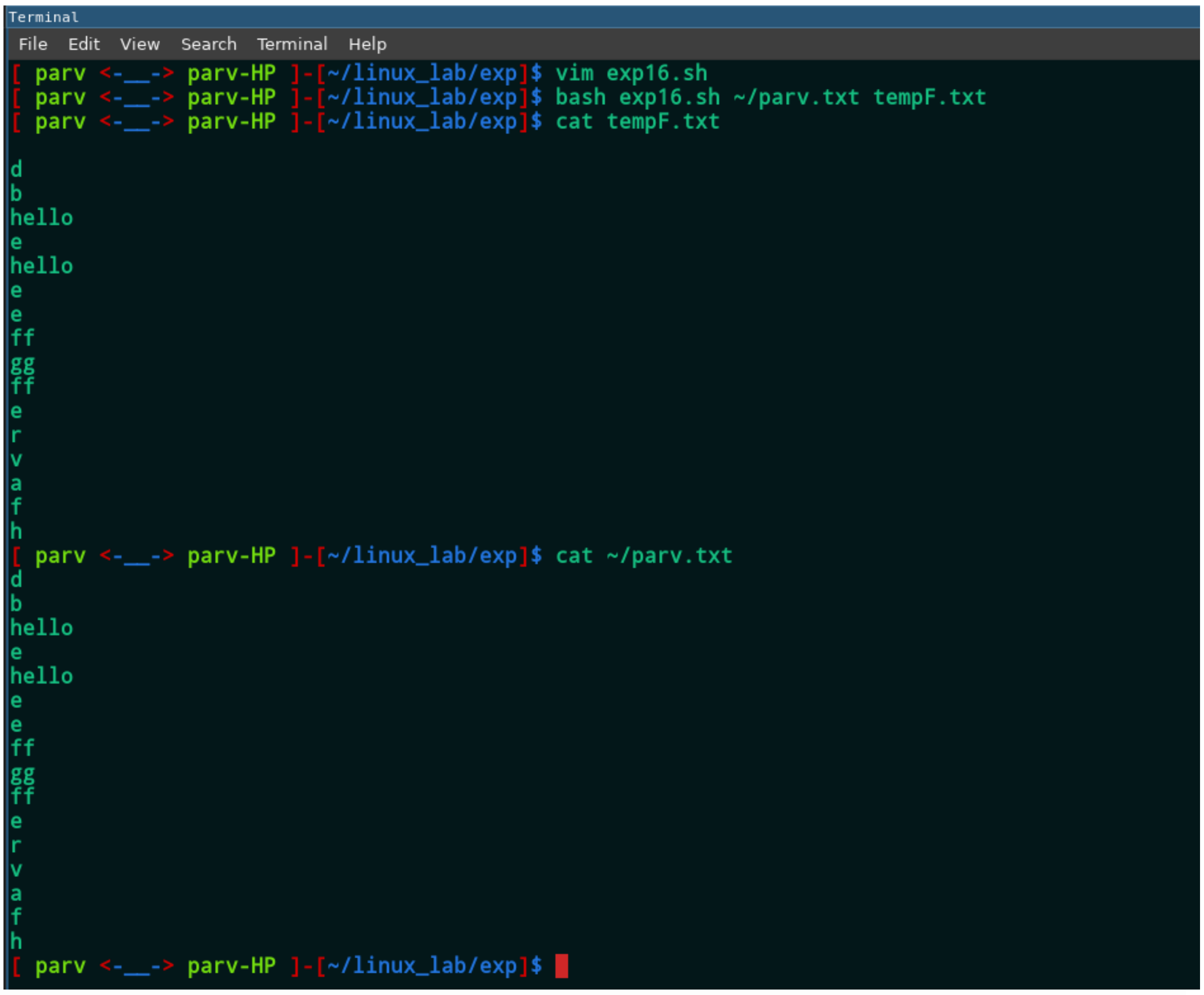
Theory:

The shell can be defined as a command interpreter within an operating system like Linux/GNU or Unix. It is a program that runs other programs.
The shell facilitates every user of the computer as an interface to the Unix/GNU Linux system. Hence, the user can execute different tools/utilities or commands with a few input data.

Script:

```
#! /bin/bash
cat $1 >> $2
```

Result/Output:



```
[ parv <-__-> parv-HP ]-[~/linux_lab/exp]$ vim exp16.sh
[ parv <-__-> parv-HP ]-[~/linux_lab/exp]$ bash exp16.sh ~/parv.txt tempF.txt
[ parv <-__-> parv-HP ]-[~/linux_lab/exp]$ cat tempF.txt

d
b
hello
e
hello
e
e
ff
gg
ff
e
r
v
a
f
h
[ parv <-__-> parv-HP ]-[~/linux_lab/exp]$ cat ~/parv.txt
d
b
hello
e
hello
e
e
ff
gg
ff
e
r
v
a
f
h
[ parv <-__-> parv-HP ]-[~/linux_lab/exp]$
```

Experiment-17

Aim: Create directory, write contents on that and Copy to a suitable location in your home directory.

Theory:

The shell can be defined as a command interpreter within an operating system like Linux/GNU or Unix. It is a program that runs other programs.
The shell facilitates every user of the computer as an interface to the Unix/GNU Linux system. Hence, the user can execute different tools/utilities or commands with a few input data.

Script:

```
#! /bin/bash

#Creating a directory
mkdir new_directory

#writing in it
echo "Hey hows it going!" > /new_directory/new_file.txt

#copying the whole directory to the home directory
cp -R ./new_directory ~/
```

Experiment-18

Aim: Use a pipeline and command substitution to set the length of a line in file to a variable.

Theory:

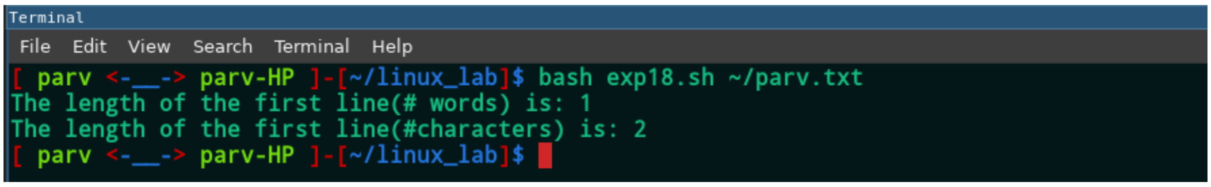
The shell can be defined as a command interpreter within an operating system like Linux/GNU or Unix. It is a program that runs other programs.
The shell facilitates every user of the computer as an interface to the Unix/GNU Linux system. Hence, the user can execute different tools/utilities or commands with a few input data.

A **pipe** is a form of redirection (transfer of standard output to some other destination) that is used in Linux and other Unix-like operating systems to send the output of one command/program/process to another command/program/process for further processing. The Unix/Linux systems allow stdout of a command to be connected to stdin of another command. You can make it do so by using the pipe character '**|**'.

Script:

```
#! /bin/bash

file=$1

var=$(head -n 1 $file | wc -w)
ch_var=$(head -n 1 $file | wc -c)

echo "The length of the first line(# words) is: $var"
echo "The length of the first line(#characters) is: $ch_var”
```

Result/Output:



```
[ parv <-__-> parv-HP ]-[~/linux_lab]$ bash exp18.sh ~/parv.txt
The length of the first line(# words) is: 1
The length of the first line(#characters) is: 2
[ parv <-__-> parv-HP ]-[~/linux_lab]$ 
```

Experiment-19

Aim: Write a program using sed command to print duplicated lines of Input.

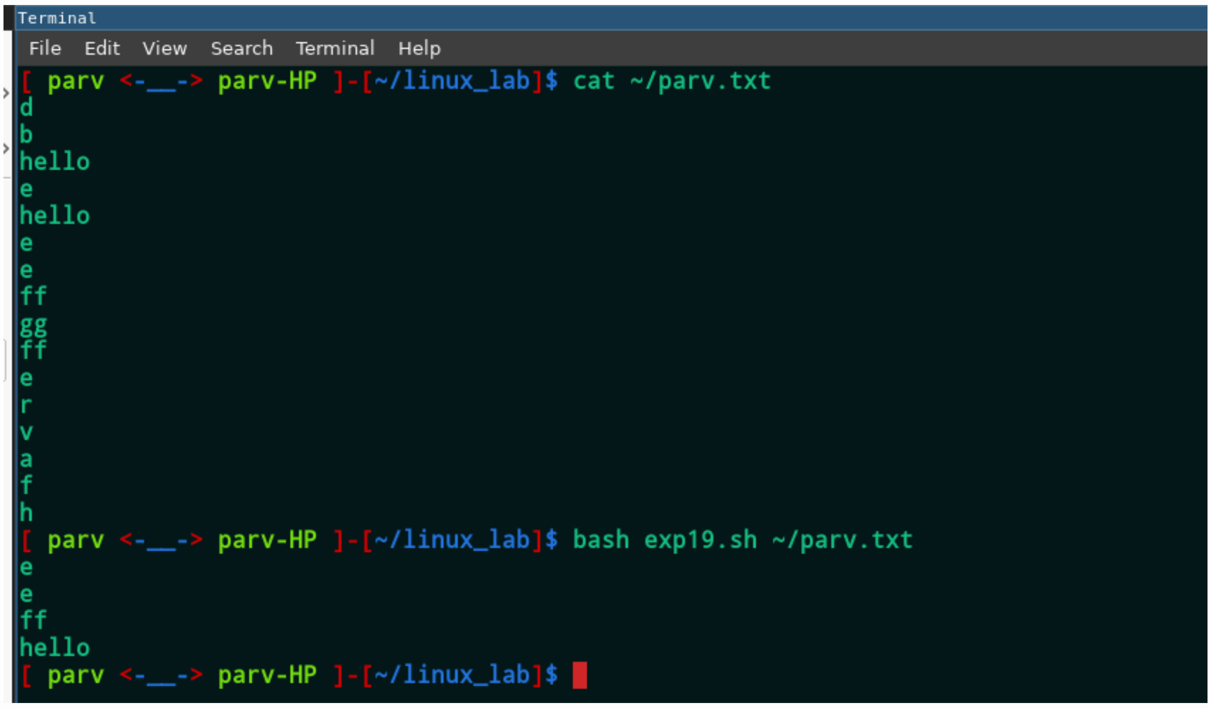
Theory:

The shell can be defined as a command interpreter within an operating system like Linux/GNU or Unix. It is a program that runs other programs.
The shell facilitates every user of the computer as an interface to the Unix/GNU Linux system. Hence, the user can execute different tools/utilities or commands with a few input data.

Script:

```
#! /bin/bash

file=$1

sort $file | sed '$!N; s/^\(.*\)\n\1$/\1/; t; D'
```

Result/Output:



```
[ parv <-__-> parv-HP ]-[~/linux_lab]$ cat ~/parv.txt
d
b
hello
e
hello
e
e
ff
gg
ff
e
r
v
a
f
h
[ parv <-__-> parv-HP ]-[~/linux_lab]$ bash exp19.sh ~/parv.txt
e
e
ff
hello
[ parv <-__-> parv-HP ]-[~/linux_lab]$
```

Experiment-20

Aim: Study the process of writing a device driver, or a kernel module.

Theory:

Kernel modules are pieces of code that can be loaded and unloaded into the kernel upon demand. They extend the functionality of the kernel without the need to reboot the system. A module can be configured as built-in or loadable. To dynamically load or remove a module, it has to be configured as a loadable module in the kernel configuration.
There are several advantages that come with using kernel modules:

- The kernel does not have to rebuild your kernel as often. This saves time and prevents the possibility of introducing an error in rebuilding and reinstalling the base kernel. Once you have a working base kernel, it is good to leave it untouched as long as possible.
- It is easier to diagnose system problems. A bug in a device driver which is bound into the kernel can stop the system from booting at all. It can also be really hard to tell which part of the base kernel caused the trouble. If the same device driver is a module, though, the base kernel is up and running before the device driver even gets loaded. If the system dies after the base kernel is up and running, it's an easy matter to track the problem down to the trouble-making device driver and just not load that device driver until the problem is fixed.
- Using modules can save memory, because they are loaded only when the system is actually using them. All parts of the base kernel stay loaded, in real storage, not just virtual storage.
- Modules are much faster to maintain and debug. What would require a full reboot to do with a filesystem driver built into the kernel can be done with a few quick commands using modules. It is possible to try out different parameters or even change the code repeatedly in rapid succession, without waiting for a boot.
- Modules are not slower, by the way, than base kernel modules. Calling either one is simply a branch to the memory location where it resides.

There are two types of modules

10.1 **Static Kernel Module:** Compiled along with the source code of the base kernel.

10.2 **Dynamic/Loadable Kernel Module**: doesn't require the whole source code of the base kernel to be compiled again. Only the kernel module code is compiled and the module is loaded onto the kernel and unloaded as and when required.

Procedure:

1 Download the necessary kernel-header required for building a kernel module.
In Ubuntu these header are already included inside **/usr/src/** . We require the headers that goes with the current version of our kernel. To check which kernel is installed, the command **name -r** is used.

2 Make sure that the gcc compiler and the make command is installed inside the system.

3 Create the C program file (in our case we are creating a simple Hello_world program).

```
#include<linux/init.h>
#include<linux/module.h>
#include<linux/kernel.h>
MODULE_LICENSE("GPL") ;

static int my_init(void)
{
    printk(KERN_INFO "HELLO WORLD \n");
    return 0;
}

static void my_exit(void)
{

    printk(KERN_INFO "Good Bye \n");

    return ;
}

module_init(my_init);
module_exit(my_exit);
```

4 Create the Makefile inside the same directory in which this .c file is created.

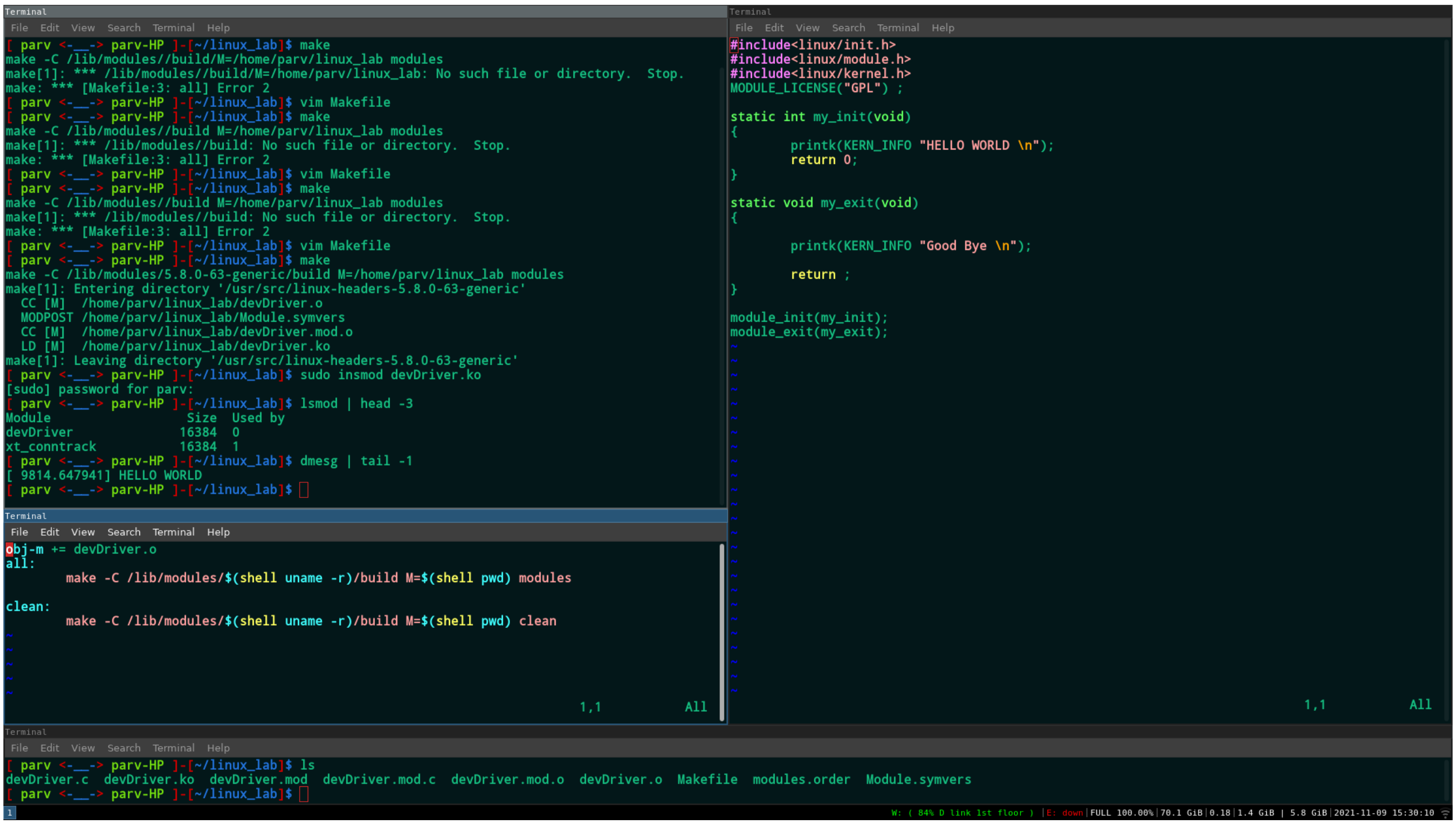
```
obj-m += devDriver.o
all:
        make -C /lib/modules/5.11.0-38-generic/build M=$(shell pwd) modules
clean:
        make -C /lib/modules/5.11.0-38-generic/build  M=$(shell pwd) clean
```

5 Now using the **make** command , compile the .c file. The following new files will be generated.

6 Now , insert the module into the kernel using the insmod command. The file used here is the .ko file generated in the previous step. We can see that our module has been inserted by using the lsmod command to list the current modules loaded into the kernel.

7 Now to check the output produced by our module on initialisation can be printed using the **dmesg** command.

8 Using the **rmmod** command we can unload our kernel.



```
[ parv <-__-> parv-HP ]-[~/linux_lab]$ make
make -C /lib/modules//build/M=/home/parv/linux_lab modules
make[1]: *** /lib/modules//build/M=/home/parv/linux_lab: No such file or directory.  Stop.
make: *** [Makefile:3: all] Error 2
[ parv <-__-> parv-HP ]-[~/linux_lab]$ vim Makefile
[ parv <-__-> parv-HP ]-[~/linux_lab]$ make
make -C /lib/modules//build M=/home/parv/linux_lab modules
make[1]: *** /lib/modules//build: No such file or directory.  Stop.
make: *** [Makefile:3: all] Error 2
[ parv <-__-> parv-HP ]-[~/linux_lab]$ vim Makefile
[ parv <-__-> parv-HP ]-[~/linux_lab]$ make
make -C /lib/modules//build M=/home/parv/linux_lab modules
make[1]: *** /lib/modules//build: No such file or directory.  Stop.
make: *** [Makefile:3: all] Error 2
[ parv <-__-> parv-HP ]-[~/linux_lab]$ vim Makefile
[ parv <-__-> parv-HP ]-[~/linux_lab]$ make
make -C /lib/modules/5.8.0-63-generic/build M=/home/parv/linux_lab modules
make[1]: Entering directory '/usr/src/linux-headers-5.8.0-63-generic'
  CC [M]  /home/parv/linux_lab/devDriver.o
  MODPOST /home/parv/linux_lab/Module.symvers
  CC [M]  /home/parv/linux_lab/devDriver.mod.o
  LD [M]  /home/parv/linux_lab/devDriver.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.8.0-63-generic'
[ parv <-__-> parv-HP ]-[~/linux_lab]$ sudo insmod devDriver.ko
[sudo] password for parv:
[ parv <-__-> parv-HP ]-[~/linux_lab]$ lsmod | head -3
Module                  Size  Used by
devDriver              16384  0
xt_conntrack           16384  1
[ parv <-__-> parv-HP ]-[~/linux_lab]$ dmesg | tail -1
[ 9814.647941] HELLO WORLD
[ parv <-__-> parv-HP ]-[~/linux_lab]$
```

```
obj-m += devDriver.o
all:
        make -C /lib/modules/$(shell uname -r)/build M=$(shell pwd) modules

clean:
        make -C /lib/modules/$(shell uname -r)/build M=$(shell pwd) clean
~
~
~
~
~
                                                              1,1           All
```

```
#include<linux/init.h>
#include<linux/module.h>
#include<linux/kernel.h>
MODULE_LICENSE("GPL") ;

static int my_init(void)
{
        printk(KERN_INFO "HELLO WORLD \n");
        return 0;
}

static void my_exit(void)
{

        printk(KERN_INFO "Good Bye \n");

        return ;
}

module_init(my_init);
module_exit(my_exit);
                                                              1,1           All
```

```
[ parv <-__-> parv-HP ]-[~/linux_lab]$ ls
devDriver.c  devDriver.ko  devDriver.mod  devDriver.mod.c  devDriver.mod.o  devDriver.o  Makefile  modules.order  Module.symvers
[ parv <-__-> parv-HP ]-[~/linux_lab]$
```

```
W: ( 84% D link 1st floor ) |E: down|FULL 100.00%|70.1 GiB|0.18|1.4 GiB | 5.8 GiB|2021-11-09 15:30:10
```