

Practical 1 : Study of Basic commands of Linux/UNIX

The important Linux commands are grouped according to their functions.

- (A) Directory oriented commands.
- (B) File oriented commands.
- (C) File access permission command
- (D) Process oriented command
- (E) General purpose command

(A) Directory Oriented Commands:

1. ls

Description: Lists the contents of the current directory.

Syntax: ls [OPTION] [DIRECTORY]

Example Output:

```
ubuntu@ubuntu:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos  snap
```

2. ls -r

Description: Lists the contents of the current directory in reverse order.

Syntax: ls -r

Example Output:

```
ubuntu@ubuntu:~$ ls -r
snap  Videos  Templates  Public  Pictures  Music  Downloads  Documents  Desktop
```

3. ls -l

Description: Lists directory contents in long format

(permissions, ownership, size, and modification date).

Syntax: `ls -l`

Example Output:

```
ubuntu@ubuntu:~$ ls -l
total 0
drwxr-xr-x 2 ubuntu ubuntu 60 Feb 18 14:38 Desktop
drwxr-xr-x 2 ubuntu ubuntu 40 Feb 18 14:39 Documents
drwxr-xr-x 2 ubuntu ubuntu 40 Feb 18 14:39 Downloads
drwxr-xr-x 2 ubuntu ubuntu 40 Feb 18 14:39 Music
drwxr-xr-x 2 ubuntu ubuntu 40 Feb 18 14:39 Pictures
drwxr-xr-x 2 ubuntu ubuntu 40 Feb 18 14:39 Public
drwxr-xr-x 2 ubuntu ubuntu 40 Feb 18 14:39 Templates
drwxr-xr-x 2 ubuntu ubuntu 40 Feb 18 14:39 Videos
drwx----- 4 ubuntu ubuntu 80 Feb 18 14:40 snap
```

4. `ls -t`

Description: Lists directory contents sorted by modification time.

Syntax: `ls -t`

Example Output:

```
ubuntu@ubuntu:~$ ls -t
snap Documents Downloads Music Pictures Public Templates Videos Desktop
```

5. `ls -a`

Description: Lists all contents, including hidden files (starting with a dot). Syntax: `ls -a`

Example Output:

```
ubuntu@ubuntu:~$ ls -a
.  ..  .bash_logout  .bashrc  .cache  .config  .gnupg  .gvfs  .local  .profile  .ssh
```

6. ls -s

Description: Displays the size of files in blocks along side their names.

Syntax: ls -s

Example Output:

```
ubuntu@ubuntu:~$ ls -s
total 0
0 Desktop 0 Documents 0 Downloads 0 Music 0 Pictures 0 Public 0 Templates 0 Videos
```

7. ls -p

Description: Appends a slash (`/`) after directory names.

Syntax: ls -p

Example Output:

```
ubuntu@ubuntu:~$ ls -p
Desktop/ Documents/ Downloads/ Music/ Pictures/ Public/ Templates/ Videos/ snap/
```

8. ls -R

Description: Recursively lists contents of directories and subdirectories.

Syntax: ls -R

Example Output:

```
ubuntu@ubuntu:~$ ls -R
.:
Desktop Documents Downloads Music Pictures Public Templates Videos snap

./Desktop:
ubuntu-desktop-bootstrap_ubuntu-desktop-bootstrap.desktop

./Documents:

./Downloads:

./Music:

./Pictures:
```

9. ls -f

Description: Displays files without sorting and marks executable files with `*`.

Syntax: ls -f

Example Output:

```
ubuntu@ubuntu:~$ ls -f
.  ..  .bash_logout  .bashrc  .profile  Desktop  .cache  .gvfs  .local  .config
```

10. mkdir

Description: Creates a new directory.

Syntax: mkdir[OPTION]DIRECTORY_NAME

Example Output:

```
ubuntu@ubuntu:~$ mkdir new_folder
ubuntu@ubuntu:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos  new_folder  snap
```

11. rmdir

Description: Removes empty directories.

Syntax: rmdir DIRECTORY_NAME

Example Output :

```
ubuntu@ubuntu:~$ rmdir new_folder
ubuntu@ubuntu:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos  snap
```

12. cd

Description: Changes the current directory.

Syntax: cd DIRECTORY_NAME

Example Output :

```
ubuntu@ubuntu:~$ cd Desktop
ubuntu@ubuntu:~/Desktop$ pwd
/home/ubuntu/Desktop
```

13. cd ..

Description: Moves upto the parent directory.

Syntax : cd ..

Example Output:

```
ubuntu@ubuntu:~/Desktop$ cd ..  
ubuntu@ubuntu:~$ pwd  
/home/ubuntu
```

14. pwd

Description: Prints the current working directory.

Syntax : pwd

Example Output:

```
ubuntu@ubuntu:~$ pwd  
/home/ubuntu
```

15. find

Description: Searches for files and directories.

Syntax: find[PATH][OPTIONS][EXPRESSION]

Example Output:

```
ubuntu@ubuntu:~$ find . -name "Documents"  
./Documents
```

16. du

Description: Displays disk usage of files and directories.

Syntax: du [OPTIONS] [PATH]

Example Output:

```
ubuntu@ubuntu:~$ du -h Documents  
0      Documents
```

17. df

Description: Displays available disk space on mounted filesystems.

Syntax : df [OPTIONS]

Example Output:

```
ubuntu@ubuntu:~$ df -h
Filesystem      Size  Used Avail Use% Mounted on
tmpfs           822M  1.8M  820M   1% /run
/dev/sr0        5.8G  5.8G    0 100% /cdrom
/cow            4.1G   56M  4.0G   2% /
tmpfs           4.1G   8.0K  4.1G   1% /dev/shm
tmpfs           5.0M   8.0K  5.0M   1% /run/lock
tmpfs           4.1G    0  4.1G   0% /tmp
tmpfs           822M  156K  822M   1% /run/user/1000
```

(B) File Oriented Commands:

18. cat

Description: Displays the contents of a file.

Syntax : cat [FILE]

Example Output:

```
ubuntu@ubuntu:~$ touch test1.txt
ubuntu@ubuntu:~$ echo "Hello from Roshan"> test1.txt
ubuntu@ubuntu:~$ cat test1.txt
Hello from Roshan
```

19. cp

Description: Copies files and directories.

Syntax: cp[SOURCE][DESTINATION]

Example Output:

```
ubuntu@ubuntu:~$ cp test1.txt test2.txt
ubuntu@ubuntu:~$ cat test2.txt
Currently studying Computer Science and Design
```

20. rm

Description: Deletes files and directories.

Syntax: rm [FILE]

Example Output:

```
ubuntu@ubuntu:~$ rm test1.txt
ubuntu@ubuntu:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos  snap  test2.txt
```

21. mv

Description: Moves or renames files and directories.

Syntax: mv [SOURCE] [DESTINATION]

Example Output:

```
ubuntu@ubuntu:~$ mv test2.txt renamed_file.txt
ubuntu@ubuntu:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos  renamed_file.txt  snap
```

22. wc

Description: Counts lines, words, and characters in a file.

Syntax : wc [FILE]

Example Output:

```
ubuntu@ubuntu:~$ wc renamed_file.txt
 1  6 47 renamed_file.txt
```

23. file

Description: Identifies the file type.

Syntax : file [FILE]

Example Output:

```
ubuntu@ubuntu:~$ file renamed_file.txt
renamed_file.txt: ASCII text
```

24. cmp

Description: Compares two files byte by byte.

Syntax: cmp [FILE1] [FILE2]

Example Output:

```
ubuntu@ubuntu:~$ cmp renamed_file.txt renamed_file2.txt
cmp: EOF on renamed_file2.txt which is empty
```

(C) File Access Permission Commands:

25. chmod

Description: Changes file permissions.

Syntax: chmod[PERMISSIONS][FILE]

Example Output :

```
ubuntu@ubuntu:~$ chmod 644 renamed_file.txt
ubuntu@ubuntu:~$ ls -l renamed_file.txt
-rw-r--r-- 1 ubuntu ubuntu 47 Feb 18 15:04 renamed_file.txt
```

26. touch

Description: Creates an empty file or updates the timestamp of a file.

Syntax : touch [FILE]

Example Output:

```
ubuntu@ubuntu:~$ touch new.txt
ubuntu@ubuntu:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos  new.txt  renamed_file.txt  renamed_file2.txt  snap
```


27. tac

Description: Displays file content in reverse order.

Syntax : tac [FILE]

Example Output:

```
ubuntu@ubuntu:~$ tac renamed_file.txt
Currently studying Computer Science and Design
```

28. head

Description: Displays the first few lines of a file.

Syntax : head [FILE]

Example Output:

```
ubuntu@ubuntu:~$ head -n 2 renamed_file.txt
Currently studying Computer Science and Design \nhey \nmessgyugyugyufgyusdfsdgdg
sdg
```

29. tail

Description: Displays the last few lines of a file.

Syntax : tail [FILE]

Example Output:

```
ubuntu@ubuntu:~$ tail -n 2 renamed_file.txt
dggt
red
```

(D) Process Oriented Commands:

30. ps -a

Description: Displays information about all running processes.

Syntax : `ps -a`

Example Output:

```
ubuntu@ubuntu:~$ ps -a
  PID TTY          TIME CMD
 1797 tty2        00:01:44 Xorg
 1839 tty2        00:00:00 gnome-session-b
 4877 pts/0        00:00:00 ps
```

31. ps -u

Description: Displays processes for a specific user.

Syntax : `ps -u [USER]`

Example Output:

```
ubuntu@ubuntu:~$ ps -u
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
ubuntu    1790   0.0   0.0 244808   6144 tty2    Ssl+  14:39   0:00 /usr/libexec/gdm-x-session --run-script env GNOME_SHELL_SESSION_MODE=ubuntu
ubuntu    1839   0.0   0.1 307296  16640 tty2    Sl+   14:39   0:00 /usr/libexec/gnome-session-binary --session=ubuntu
ubuntu    4191   0.0   0.0  20100   4992 pts/0    Ss    14:46   0:00 bash
ubuntu    4878  100   0.0   22720   4608 pts/0    R+    15:32   0:00 ps -u
```

32. ps -x

Description: Shows processes without a controlling terminal.

Syntax : `ps -x`

Example Output:

```
ubuntu@ubuntu:~$ ps -x
  PID TTY      STAT   TIME COMMAND
 1751 ?        Ss      0:00 /usr/lib/systemd/systemd --user
 1752 ?        S        0:00 (sd-pam)
 1763 ?        S<sl    0:00 /usr/bin/pipewire
 1764 ?        Ssl     0:00 /usr/bin/pipewire -c filter-chain.conf
 1766 ?        S<sl    0:00 /usr/bin/wireplumber
 1767 ?        S<sl    0:00 /usr/bin/pipewire-pulse
 1768 ?        Ssl     0:00 /usr/bin/gnome-keyring-daemon --foreground --components=pkcs11,secrets --control-directory=/run/user/1000/keyring
 1773 ?        Ss      0:00 /usr/bin/dbus-daemon --session --address=systemd: --nofork --nopidfile --systemd-activation --syslog-only
 1790 tty2    Ssl+    0:00 /usr/libexec/gdm-x-session --run-script env GNOME_SHELL_SESSION_MODE=ubuntu /usr/bin/gnome-session --session=ubuntu
 1839 tty2    Sl+     0:00 /usr/libexec/gnome-session-binary --session=ubuntu
```

33. ps -t

Description: Displays processes associated with a terminal.

Syntax: `ps -t [TERMINAL]`

Example Output:

```
ubuntu@ubuntu:~$ ps -t
  PID TTY          STAT       TIME COMMAND
  4191 pts/0        Ss          0:00   bash
  4881 pts/0        R+          0:00   ps -t
```

(E) General Purpose Commands:

34. date

Description: Displays the current date and time.

Syntax : date

Example Output:

```
ubuntu@ubuntu:~$ date
Tue Feb 18 15:32:41 UTC 2025
```

35. who

Description: Displays all list of users currently logged in.

Syntax: who

Example Output:

```
ubuntu@ubuntu:~$ who
ubuntu  seat0          2025-02-18 14:39 (login screen)
ubuntu  :0                2025-02-18 14:39 (:0)
```

36. cal

Description: Displays a calendar for the current month.

Syntax: cal

Example Output:

```
ubuntu@ubuntu:~$ cal
  February 2025
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
```

Practical 2 : Study of Advance commands and filters of Linux/UNIX.

1. head

Description: Displays the first few lines of a file.

Syntax: head [FILE]

Example Output:

```
ubuntu@ubuntu:~$ head -n 2 renamed_file.txt
Currently studying Computer Science and Design \nhey \nmessgyugyugyufgyusdfsdgdg
sdg
```

2. tail

Description: Displays the last few lines of a file.

Syntax: tail [FILE]

Example Output:

```
ubuntu@ubuntu:~$ tail -n 2 renamed_file.txt
dggg
red
```

3. ps -a

Description: Displays information about all running processes.

Syntax: ps -a

Example Output:

```
ubuntu@ubuntu:~$ ps -a
  PID TTY          TIME CMD
 1797 tty2        00:01:44 Xorg
 1839 tty2        00:00:00 gnome-session-b
 4877 pts/0        00:00:00 ps
```

4. ps -u

Description: Displays processes for a specific user.

Syntax: ps -u [USER]

Example Output:

```
ubuntu@ubuntu:~$ ps -u
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
ubuntu    1790  0.0  0.0 244808  6144 tty2    Ssl+  14:39   0:00 /usr/libexec/gdm-x-session --run-script env GNOME_SHELL_SESSION_MODE=ubuntu
ubuntu    1839  0.0  0.1 307296 16640 tty2    Sl+   14:39   0:00 /usr/libexec/gnome-session-binary --session=ubuntu
ubuntu    4191  0.0  0.0  20100  4992 pts/0    Ss   14:46   0:00 bash
ubuntu    4878 100  0.0  22720  4608 pts/0    R+   15:32   0:00 ps -u
```

5. ps -x

Description: Shows processes without a controlling terminal.

Syntax: ps -x

Example Output:

```
ubuntu@ubuntu:~$ ps -x
  PID TTY          STAT TIME  COMMAND
 1751 ?            Ss      0:00 /usr/lib/systemd/systemd --user
 1752 ?            S        0:00 (sd-pam)
 1763 ?           S<sl    0:00 /usr/bin/pipewire
 1764 ?           Ssl     0:00 /usr/bin/pipewire -c filter-chain.conf
 1766 ?           S<sl    0:00 /usr/bin/wireplumber
 1767 ?           S<sl    0:00 /usr/bin/pipewire-pulse
 1768 ?           Ssl     0:00 /usr/bin/gnome-keyring-daemon --foreground --components=pkcs11,secrets --control-directory=/run/user/1000/keyring
 1773 ?            Ss      0:00 /usr/bin/dbus-daemon --session --address=systemd: --nofork --nopidfile --systemd-activation --syslog-only
 1790 tty2        Ssl+    0:00 /usr/libexec/gdm-x-session --run-script env GNOME_SHELL_SESSION_MODE=ubuntu /usr/bin/gnome-session --session=ubuntu
 1839 tty2        Sl+     0:00 /usr/libexec/gnome-session-binary --session=ubuntu
```

6. ps -t

Description: Displays processes associated with a terminal.

Syntax: ps -t [TERMINAL]

Example Output:

```
ubuntu@ubuntu:~$ ps -t
  PID TTY          STAT TIME  COMMAND
 4191 pts/0        Ss      0:00 bash
 4881 pts/0        R+      0:00 ps -t
```

7. date

Description: Displays the current date and time.

Syntax: date

Example Output:

```
ubuntu@ubuntu:~$ date  
Tue Feb 18 15:32:41 UTC 2025
```

8. who

Description: Displays a list of users currently logged in.

Syntax: who

Example Output:

```
ubuntu@ubuntu:~$ who  
ubuntu    seat0      2025-02-18 14:39 (login screen)  
ubuntu    :0         2025-02-18 14:39 (:0)
```

9. whoami

Description: Displays the username and terminal information of the current user session.

Syntax: whoami

Example Output:

```
ubuntu@ubuntu:~$ whoami  
ubuntu
```

10. cal

Description: The cal command in Linux displays a calendar.

Syntax: cal

Example Output:

```
ubuntu@ubuntu:~$ cal  
    February 2025  
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
```


11. kill

Description: Sends signals to processes. By default, it sends a signal to terminate a process, but other signals can be specified to perform different actions

Syntax: kill[OPTION]<PID>... Example Output:

```
ubuntu@ubuntu:~$ kill 1234
bash: kill: (1234) - No such process
```

12. man

Description: The man command in Linux is used to display the manual pages for other commands, providing detailed documentation, including usage, options, and examples.

Syntax: man[COMMAND] Example Output:

```
ubuntu@ubuntu:~$ man ls
```

```
LS(1) User Commands
NAME
  ls - list directory contents
SYNOPSIS
  ls [OPTION]... [FILE]...
DESCRIPTION
  List information about the FILES (the current directory by default). Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.
  Mandatory arguments to long options are mandatory for short options too.
  -a, --all
    do not ignore entries starting with .
  -A, --almost-all
    do not list implied . and ..
  --author
    with -l, print the author of each file
  -b, --escape
    print C-style escapes for nongraphic characters
  --block-size=SIZE
    with -l, scale sizes by SIZE when printing them; e.g., '--block-size=M'; see SIZE format below
  -B, --ignore-backups
    do not list implied entries ending with ~
```

13. tee

Description: The tee command in Linux reads from standard input(stdin) and writes the output to both standard output (stdout) and one or more files. It's often used in combination with other commands via pipes (|)

Syntax: command|tee[OPTION]...[FILE]...

Example Output:

```
ubuntu@ubuntu:~$ ls -l | tee renamed_file.txt
total 4
drwxr-xr-x 2 ubuntu ubuntu 60 Feb 18 14:38 Desktop
drwxr-xr-x 2 ubuntu ubuntu 40 Feb 18 14:39 Documents
drwxr-xr-x 2 ubuntu ubuntu 40 Feb 18 14:39 Downloads
drwxr-xr-x 2 ubuntu ubuntu 40 Feb 18 14:39 Music
drwxr-xr-x 3 ubuntu ubuntu 60 Feb 18 15:17 Pictures
drwxr-xr-x 2 ubuntu ubuntu 40 Feb 18 14:39 Public
drwxr-xr-x 2 ubuntu ubuntu 40 Feb 18 14:39 Templates
drwxr-xr-x 2 ubuntu ubuntu 40 Feb 18 14:39 Videos
-rw-rw-r-- 1 ubuntu ubuntu 0 Feb 18 15:09 new.txt
-rw-r--r-- 1 ubuntu ubuntu 0 Feb 18 15:41 renamed_file.txt
-rw-rw-r-- 1 ubuntu ubuntu 0 Feb 18 15:07 renamed_file2.txt
drwx----- 5 ubuntu ubuntu 100 Feb 18 15:00 snap
-rw-rw-r-- 1 ubuntu ubuntu 62 Feb 18 15:12 test1.txt
```

14. script

Description: The script command in Linux is used to record a terminal session, capturing all inputs and outputs into a file. This is useful for logging or debugging purposes.

Syntax: script[OPTION]...[FILE]

Example Output:

```
ubuntu@ubuntu:~$ script renamed_file.txt
Script started, output log file is 'renamed_file.txt'.
```

15. expr

Description: The expr command in Linux is used for evaluating expressions. It can perform arithmetic operations, string manipulations, and comparisons. It's commonly used for simple calculations in shell scripts.

Syntax: exprEXPR

Example Output:

```
ubuntu@ubuntu:~$ expr 5 + 9
14
ubuntu@ubuntu:~$ expr length "Hello World"
11
```

16. bc

Description: The bc command allows users to perform calculations interactively or execute mathematical operations provided in files or through standard input. It supports floating-point arithmetic, variables, and control structures.

Syntax: bc[expression]

Example Output:

```
ubuntu@ubuntu:~$ 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'.
5 + 3
8
5 - 1
4
```

17. sort

Description: The sort command in Linux is used to sort lines of text files or standard input in various ways, such as alphabetically, numerically, or in reverse order. It is highly versatile and supports a range of options for different sorting needs.

Syntax: sort[option]<filename>

Example Output:

```
ubuntu@ubuntu:~$ cat advance.txt
Cat
Tiger
Elephant
Lion
Bird
ubuntu@ubuntu:~$ sort advance.txt
Bird
Cat
Elephant
Lion
Tiger
```

18. grep

Description: The grep command searches a file or input line by line for matches to a specified pattern. By default, it outputs the lines that match the given pattern. It also supports various options for casesensitivity, recursive searches, line numbers, and more.

Syntax: `grep[OPTIONS]PATTERN[FILE]...`

Example Output:

```
ubuntu@ubuntu:~$ grep "Lion" advance.txt
Lion
```

29. uniq

Description: The uniq command filters out duplicate lines from sorted input, displaying only unique lines or repeated lines based on options.

Syntax: `uniq[OPTIONS][INPUT][OUTPUT]`

Example output:

```
ubuntu@ubuntu:~$ cat advance.txt
Cat
Tiger
Elephant
Lion
Bird
Bird
Bird
Pigeon
Snail
ubuntu@ubuntu:~$ uniq advance.txt
Cat
Tiger
Elephant
Lion
Bird
Pigeon
Snail
```

20. more

Description: The more command is used to view the contents of a file or the output of a command one screen at a time in the terminal. It is especially useful when dealing with long outputs that exceed the visible area of the terminal.

Syntax: more<filename>

Example output:

```
ubuntu@ubuntu:~$ ls | more
Desktop
Documents
Downloads
Music
Pictures
Public
Templates
Videos
advance.txt
new.txt
renamed_file.txt
renamed_file2.txt
snap
test1.txt
```

21. cut

Description: The cut command is used to extract specific sections of text(fields or characters) from a file or input.

Syntax: cut[OPTIONS][FILE]...

Example output:

```
ubuntu@ubuntu:~$ cat advance.txt
Cat
Tiger
Elephant
Lion
Bird
Bird
Bird
Pigeon
Snail
ubuntu@ubuntu:~$ cut -f 1 advance.txt
Cat
Tiger
Elephant
Lion
Bird
Bird
Bird
Pigeon
Snail
```

22. paste

Description: The paste command merges lines of files horizontally by joining them with a delimiter.

Syntax: paste[OPTIONS]FILE... Example output:

```
ubuntu@ubuntu:~$ echo -e "Alice\nBob\nCharlie" > employee1.txt
ubuntu@ubuntu:~$ echo -e "2000\n3000\n9000" > employee2.txt
ubuntu@ubuntu:~$ cat employee1.txt
Alice
Bob
Charlie
ubuntu@ubuntu:~$ cat employee2.txt
2000
3000
9000
ubuntu@ubuntu:~$ paste employee1.txt employee2.txt
Alice 2000
Bob 3000
Charlie 9000
```

Practical 3 : Write a shell script to generate marksheet of a student. Take 3 subjects, calculate and display total marks, percentage and Class obtained by the student

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

```
determine_class() {  
    local percentage=$1  
    if (( $(echo "$percentage >= 75" | bc -l) )); then  
        echo "DISTINCTION"  
    elif (( $(echo "$percentage >= 60" | bc -l) )); then  
        echo "FIRST CLASS"  
    elif (( $(echo "$percentage >= 50" | bc -l) )); then  
        echo "SECOND CLASS"  
    elif (( $(echo "$percentage >= 35" | bc -l) )); then  
        echo "PASS CLASS"  
    else  
        echo "FAIL"  
    fi  
}
```

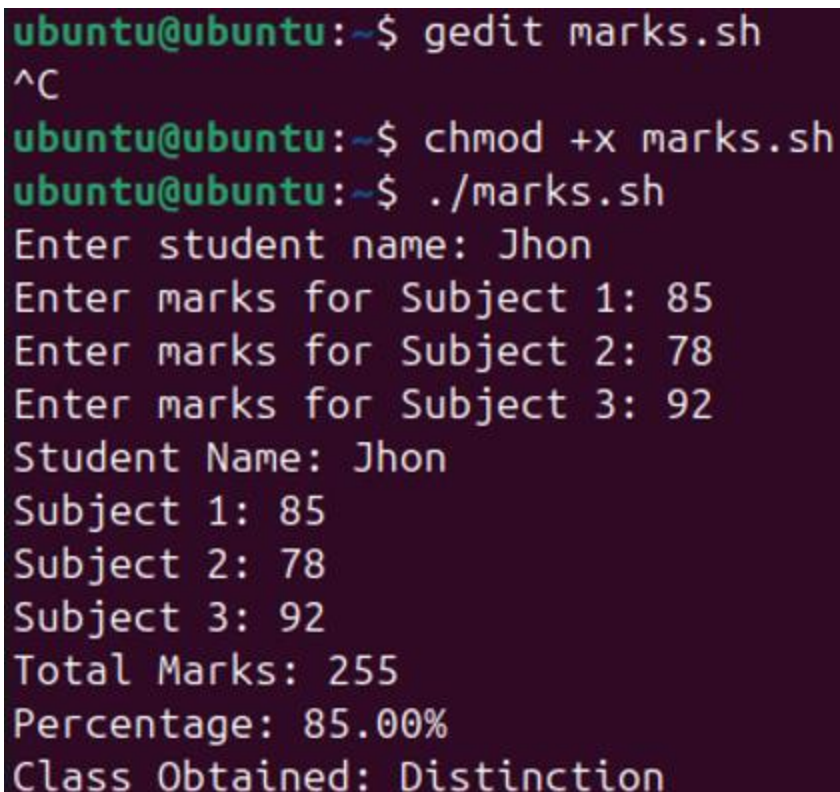
```
read -p "Enter student name: " name  
read -p "Enter marks for Subject 1: " sub1  
read -p "Enter marks for Subject 2: " sub2  
read -p "Enter marks for Subject 3: " sub3
```



```
total_marks=$((sub1 + sub2 + sub3))
percentage=$(echo "scale=2; $total_marks / 3" | bc)

class=$(determine_class $percentage)

echo "Student Name: $name"
echo "Subject 1: $sub1"
echo "Subject 2: $sub2"
echo "Subject 3: $sub3"
echo "Total Marks: $total_marks"
echo "Percentage: $percentage%"
echo "Class Obtained: $class"
```



```
ubuntu@ubuntu:~$ gedit marks.sh
^C
ubuntu@ubuntu:~$ chmod +x marks.sh
ubuntu@ubuntu:~$ ./marks.sh
Enter student name: Jhon
Enter marks for Subject 1: 85
Enter marks for Subject 2: 78
Enter marks for Subject 3: 92
Student Name: Jhon
Subject 1: 85
Subject 2: 78
Subject 3: 92
Total Marks: 255
Percentage: 85.00%
Class Obtained: Distinction
```

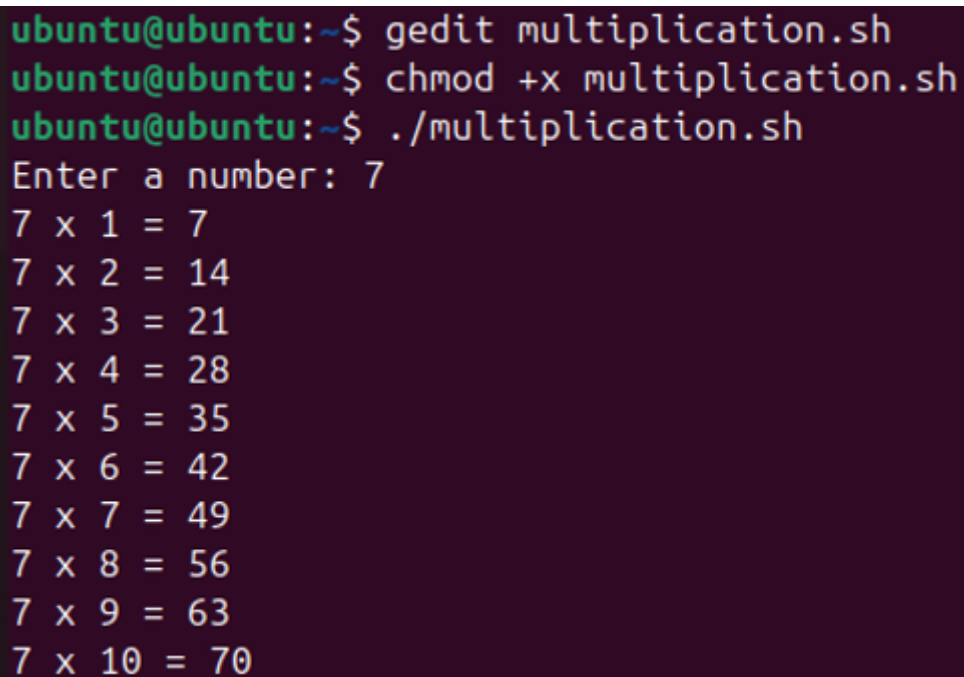
Practical 4 : Write a shell script to display multiplication table of given number

```
#!/bin/bash

read -p "Enter a number: " number

i=1

while [ $i -le 10 ]; do
    echo "$number x $i = $((number * i))"
    i=$((i + 1))
done
```



```
ubuntu@ubuntu:~$ gedit multiplication.sh
ubuntu@ubuntu:~$ chmod +x multiplication.sh
ubuntu@ubuntu:~$ ./multiplication.sh
Enter a number: 7
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70
```

Practical 5 : Write a shell script to find factorial of given number n.

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

```
read -p "Enter a number: " n
```

```
factorial=1
```

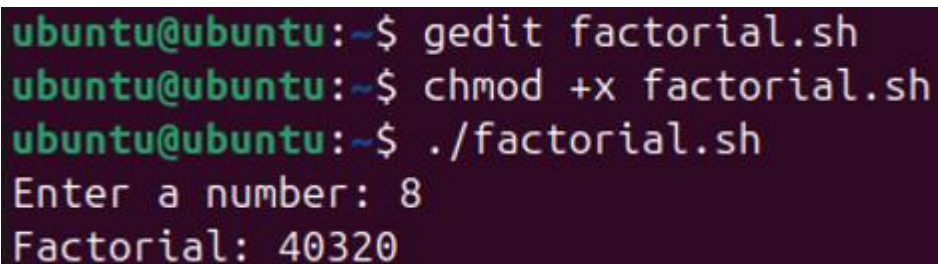
```
while [ $n -gt 1 ]; do
```

```
    factorial=$((factorial * n))
```

```
    n=$((n - 1))
```

```
done
```

```
echo "Factorial: $factorial"
```



```
ubuntu@ubuntu:~$ gedit factorial.sh
ubuntu@ubuntu:~$ chmod +x factorial.sh
ubuntu@ubuntu:~$ ./factorial.sh
Enter a number: 8
Factorial: 40320
```

Practical 6 : Write a menu driven shell script which will print the following menu and execute the given task. a. Display calendar of current month b. Display today's date and time c. Display usernames those are currently logged in the system d. Display your name at given x, y position e. Display your terminal number

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

```
while true; do
```

```
    echo "Menu:"
```

```
    echo "a. Display calendar of current month"
```

```
    echo "b. Display today's date and time"
```

```
    echo "c. Display usernames those are currently logged in the system"
```

```
    echo "d. Display your name at given x, y position"
```

```
    echo "e. Exit"
```

```
    read -p "Enter your choice: " choice
```

```
case $choice in
```

```
    a)
```

```
        cal
```

```
        ;;
```

```
    b)
```

```
        date
```

```
        ;;
```

c)

who

;;

d)

read -p "Enter your name: " name

read -p "Enter x position: " x

read -p "Enter y position: " y

tput cup \$x \$y

echo "\$name"

;;

e)

tty

;;

f)

exit

;;

*)

echo "Invalid choice, please try again."

;;

esac

done

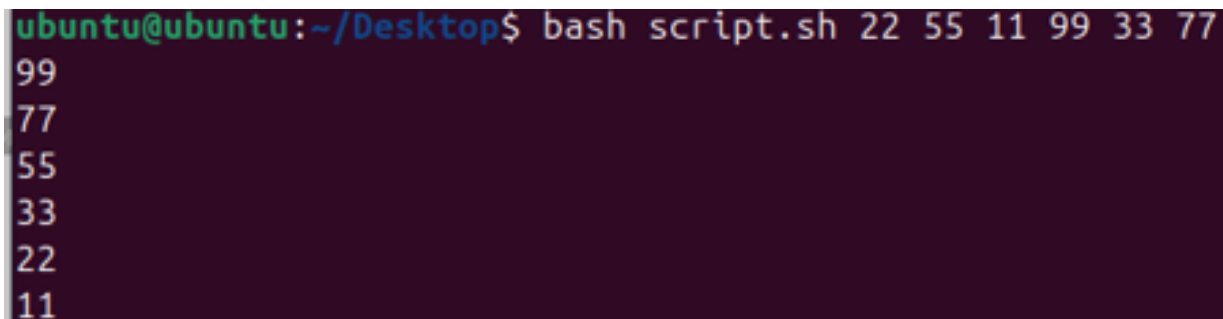
```
ubuntu@ubuntu:~$ gedit menu.sh
ubuntu@ubuntu:~$ chmod +x menu.sh
ubuntu@ubuntu:~$ ./menu.sh
Menu:
a. Display calendar of current month
b. Display today's date and time
c. Display usernames those are currently logged in the system
d. Display your name at given x, y position
e. Display your terminal number
f. Exit
Enter your choice: a
    February 2025
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

Menu:
a. Display calendar of current month
b. Display today's date and time
c. Display usernames those are currently logged in the system
d. Display your name at given x, y position
e. Display your terminal number
f. Exit
Enter your choice: b
Tue Feb 18 18:43:10 UTC 2025
Menu:
a. Display calendar of current month
b. Display today's date and time
c. Display usernames those are currently logged in the system
d. Display your name at given x, y position
e. Display your terminal number
f. Exit
Enter your choice: c
ubuntu  seat0      2025-02-18 18:09 (login screen)
ubuntu  :0         2025-02-18 18:09 (:0)
Menu:
a. Display calendar of current month
b. Display today's date and time
c. Display usernames those are currently logged in the system
d. Display your name at given x, y position
e. Display your terminal number
f. Exit
Enter your choice: f
```

Practical 7 : Write a shell script to read n numbers as command arguments and sort them in descending order

Shellscript:

```
if [ $# -eq 0 ]; then
    echo "No numbers provided"
    exit 1
fi
echo "$@" | tr ' ' '\n' | sort -nr
exit 0
```



A terminal window screenshot showing the execution of a shell script. The prompt is `ubuntu@ubuntu:~/Desktop$`. The command `bash script.sh 22 55 11 99 33 77` is entered. The output shows the numbers sorted in descending order: `99`, `77`, `55`, `33`, `22`, and `11`.

Practical 8 : Shell programming using filters (including grep, egrep, fgrep)

Shellscript:

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

```
# Check if files are provided as arguments
```

```
if [ $# -eq 0 ]; then
```

```
    echo "Please provide at least one file to process"
```

```
    exit 1
```

```
fi
```

```
echo "1. Using grep to find lines with 'ram' or 'sam':"
```

```
grep "\(r|s\)am" "$@"
```

```
echo "-----"
```

```
echo "2. Using egrep to find lines with 'ram' or 'sam':"
```

```
egrep "(r|s)am" "$@"
```

```
echo "-----"
```

```
echo "3. Using fgrep to find literal 'hello world':"
```

```
fgrep "hello world" "$@"
```

```
echo "-----"
```

```
echo "4. Chaining filters to extract and sort numbers:"
```

```
cat "$@" | egrep "^[0-9]+$" | sort -n
```

```
echo "-----"
```

```
ubuntu@ubuntu:~/Desktop$ echo -e "ram is here\nsam I am\ntam tam\nhello worl\n123\n45\nhello there" >greping.txt
ubuntu@ubuntu:~/Desktop$ echo -e "goodbye\nhello world again\nram agai" > another.txt
ubuntu@ubuntu:~/Desktop$ ./filter_script.sh greping.txt another.txt
1. Using grep to find lines with 'ram' or 'sam':
-----
2. Using egrep to find lines with 'ram' or 'sam':
greping.txt:ram is here
greping.txt:sam I am
another.txt:ram agai
-----
3. Using fgrep to find literal 'hello world':
another.txt:hello world again
-----
4. Chaining filters to extract and sort numbers:
45
123
-----
```

Practical 9 : Write a shell script to display all executable files, directories and zero sized files from current directory

Shellscript:

```
echo "----- List of executable files-----"
```

```
for sort in * ; do
```

```
if [ -f "$sort" ] && [ -x "$sort" ] ; then
```

```
echo "$sort"
```

```
fi
```

```
done
```

```
echo "----- List of directories-----"
```

```
for sort in * ; do
```

```
if [ -d "$sort" ]; then
```

```
echo "$sort"
```

```
fi
```

```
done
```

```
echo "----- For zero size file-----"
```

```
for sort in * ; do
```

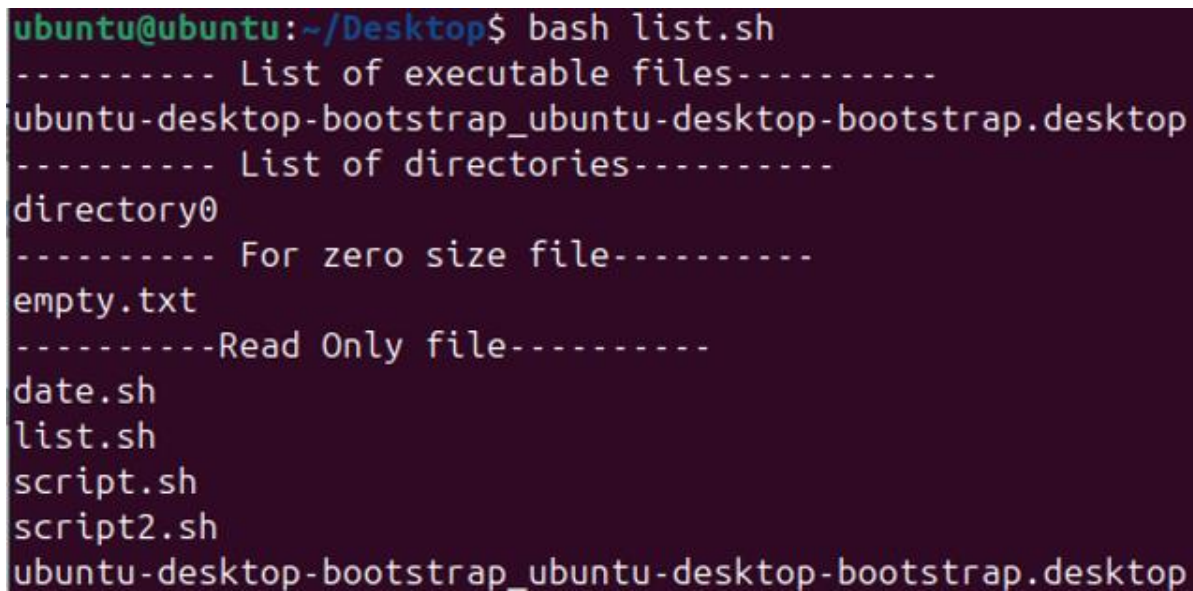
```
if [ -f "$sort" ] && [ ! -s "$sort" ]; then
```

```
echo "$sort"
```

```
fi
```

```
done
```

```
echo "-----Read Only file-----"
for sort in * ; do
if [ -r "$sort" ] && [ -f "$sort" ] && [ -s "$sort" ]; then
echo "$sort"
fi
done
```



```
ubuntu@ubuntu:~/Desktop$ bash list.sh
----- List of executable files-----
ubuntu-desktop-bootstrap_ubuntu-desktop-bootstrap.desktop
----- List of directories-----
directory0
----- For zero size file-----
empty.txt
-----Read Only file-----
date.sh
list.sh
script.sh
script2.sh
ubuntu-desktop-bootstrap_ubuntu-desktop-bootstrap.desktop
```

Practical 10 : Write a shell script to check entered string is palindrome or not.

Shellscript:

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

```
read -p "Enter a string: " str
```

```
clean_str=$(echo "$str" | tr -d ' ' | tr '[:upper:]' '[:lower:]')
```

```
rev_str=$(echo "$clean_str" | rev)
```

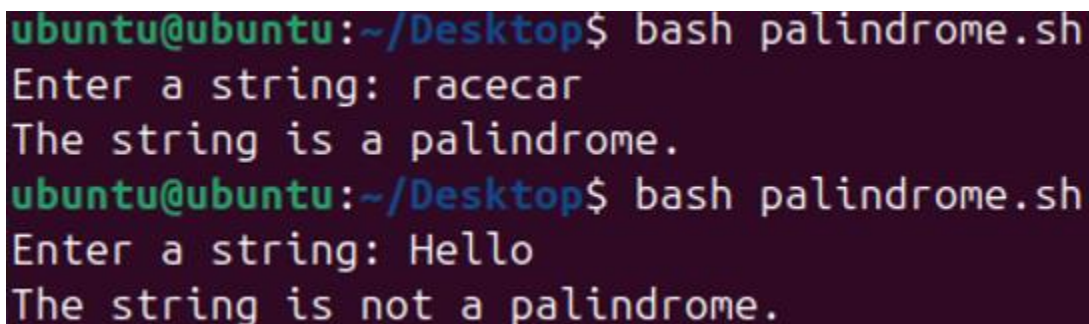
```
if [ "$clean_str" = "$rev_str" ]; then
```

```
    echo "The string is a palindrome."
```

```
else
```

```
    echo "The string is not a palindrome."
```

```
fi
```



```
ubuntu@ubuntu:~/Desktop$ bash palindrome.sh
Enter a string: racecar
The string is a palindrome.
ubuntu@ubuntu:~/Desktop$ bash palindrome.sh
Enter a string: Hello
The string is not a palindrome.
```

Practical 11 : Write a shell script to validate the entered date. (eg. Date format is : dd-mm-yyyy).

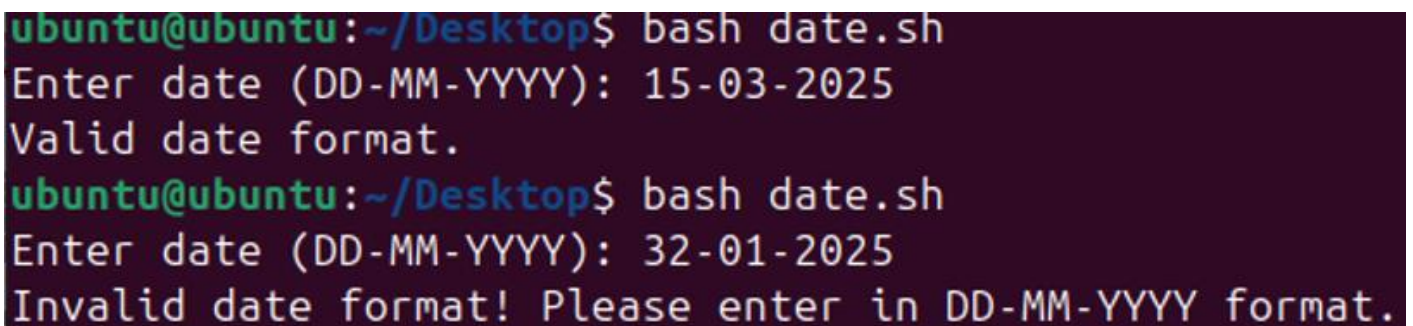
Shellscript:

```
#!/bin/bash

# Regular expression for date validation (DD-MM-YYYY format)
DATE_REGEX="^(0[1-9]|[12][0-9]|3[01])-(0[1-9]|1[0-2])-[0-9]{4}$"

# Read date from user
read -p "Enter date (DD-MM-YYYY): " date

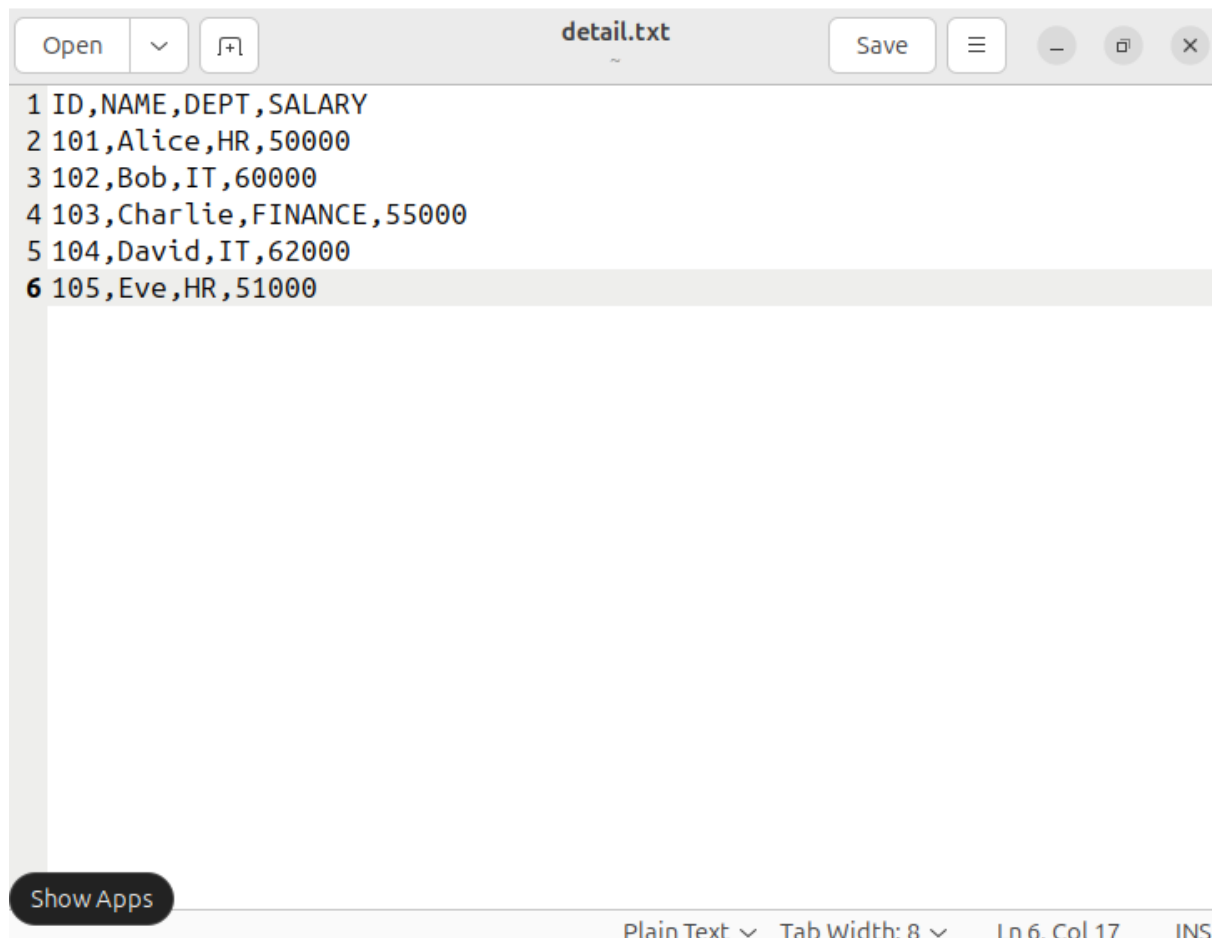
# Validate using regex
if [[ $date =~ $DATE_REGEX ]]; then
    echo "Valid date format."
else
    echo "Invalid date format! Please enter in DD-MM-YYYY format."
fi
```



```
ubuntu@ubuntu:~/Desktop$ bash date.sh
Enter date (DD-MM-YYYY): 15-03-2025
Valid date format.
ubuntu@ubuntu:~/Desktop$ bash date.sh
Enter date (DD-MM-YYYY): 32-01-2025
Invalid date format! Please enter in DD-MM-YYYY format.
```

Practical 12 : Write an awk program using function, which convert each word in a given text into capital.

Text file:



The screenshot shows a text editor window titled "detail.txt". The window contains a CSV file with 6 lines of data. The first line is a header: "1 ID,NAME,DEPT,SALARY". The following lines are data rows: "2 101,Alice,HR,50000", "3 102,Bob,IT,60000", "4 103,Charlie,FINANCE,55000", "5 104,David,IT,62000", and "6 105,Eve,HR,51000". The window has a toolbar with "Open", "Save", and other icons. At the bottom, there is a status bar showing "Plain Text", "Tab Width: 8", "Ln 6, Col 17", and "INS".

```
1 ID,NAME,DEPT,SALARY
2 101,Alice,HR,50000
3 102,Bob,IT,60000
4 103,Charlie,FINANCE,55000
5 104,David,IT,62000
6 105,Eve,HR,51000
```

awk file code:

```
# AWK program to convert each word to uppercase using a function
```

```
# Define a function to convert a string to uppercase
```

```
function to_upper(str) {
```

```
    return toupper(str)
```



```
}
```

```
# Begin block to print a header
```

```
BEGIN {
```

```
    FS = "," # Set field separator to comma
```

```
    OFS = "," # Set output field separator to comma
```

```
    print "ID,NAME,DEPT,SALARY" # Print header
```

```
}
```

```
# Main block to process each line
```

```
{
```

```
    # Skip the header line (if it matches the pattern)
```

```
    if (NR == 1) next
```

```
    # Process each field and convert to uppercase
```

```
    $1 = to_upper($1) # ID
```

```
    $2 = to_upper($2) # NAME
```

```
    $3 = to_upper($3) # DEPT
```

```
    $4 = to_upper($4) # SALARY
```

```
    # Print the modified line
```

```
    print $0
```

```
}
```

In terminal:

```
ubuntu@ubuntu:~$ awk -f convertword.awk detail.txt
ID,NAME,DEPT,SALARY
101,ALICE,HR,50000
102,BOB,IT,60000
103,CHARLIE,FINANCE,55000
104,DAVID,IT,62000
105,EVE,HR,51000
```

Practical 13 : Write a program which demonstrate the use of fork, join, and exec and wait system calls.

Fork :

```
#include <stdio.h>

#include <unistd.h>

int main() {

    printf("Before fork, PID: %d\n", getpid());

    // Create a new process

    pid_t pid = fork();

    if (pid < 0) {

        // Fork failed

        perror("Fork failed");

        return 1;

    } else if (pid == 0) {

        // Child process

        printf("Child process: PID = %d, Parent PID = %d\n", getpid(), getppid());

    } else {

        // Parent process

        printf("Parent process: PID = %d, Child PID = %d\n", getpid(), pid);

    }

    // Code executed by both parent and child
```

```
printf("This message is from PID = %d\n", getpid());

return 0;
}
```

```
Before fork, PID: 14489
Parent process: PID = 14489, Child PID = 14490
This message is from PID = 14489
Child process: PID = 14490, Parent PID = 14489
This message is from PID = 14490

=== Code Execution Successful ===
```

join :

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

void *thread_function(void *arg) {
    printf("Thread %ld started.\n", pthread_self());
    sleep(2);
    printf("Thread %ld finished.\n", pthread_self());
    return NULL;
}

int main() {
    pthread_t thread;
```

```
// Creating a thread
if (pthread_create(&thread, NULL, thread_function, NULL) != 0) {
    perror("Thread creation failed");
    return 1;
}

// Waiting for the thread to complete
printf("Main thread waiting for child thread...\n");
pthread_join(thread, NULL);
printf("Main thread exiting.\n");

return 0;
}
```

```
Main thread waiting for child thread...
Thread 140564004746944 started.
Thread 140564004746944 finished.
Main thread exiting.
```

```
=== Code Execution Successful ===
```

exec :

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
```

```
int main() {
    printf("Executing `ls -l` using execvp()\n");
```

```
// Arguments for execvp (running "ls -l")
char *args[] = {"/bin/ls", "-l", NULL};

// Replacing current process with ls -l
execvp(args[0], args);

// This line is only reached if exec fails
perror("Exec failed");
return 1;
}
```

```
Executing `ls -l` using execvp()
total 0

=== Code Execution Successful ===
```

wait :

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
```

```
int main() {
    pid_t pid = fork();

    if (pid < 0) {
        perror("Fork failed");
    }
}
```

```
    return 1;
}
else if (pid == 0) {

    // Child process
    printf("Child process: PID = %d\n", getpid());
    sleep(2); // Simulating some work
    printf("Child process exiting...\n");
    exit(42); // Exiting with status 42
}
else {
    // Parent process waits for child to finish
    int status;
    printf("Parent waiting for child to complete...\n");
    wait(&status);

    if (WIFEXITED(status)) {
        printf("Child exited with status %d\n", WEXITSTATUS(status));
    }
    printf("Parent process finished.\n");
}
return 0;
}
```

```
Parent waiting for child to complete...
Child process: PID = 19433
Child process exiting...
Child exited with status 42
Parent process finished.
```

```
=== Code Execution Successful ===
```

Practical 14 : Write a C program to simulate FCFS CPU scheduling algorithm.

```
#include <stdio.h>
```

```
struct Process {  
    int pid;  
    int burst_time;  
    int arrival_time;  
    int waiting_time;  
    int turnaround_time;  
    int completion_time;  
};
```

```
void sort_by_arrival(struct Process p[], int n) {  
    struct Process temp;  
    for(int i = 0; i < n; i++) {  
        for(int j = i + 1; j < n; j++) {  
            if(p[i].arrival_time > p[j].arrival_time) {  
                temp = p[i];  
                p[i] = p[j];  
                p[j] = temp;  
            }  
        }  
    }  
}
```

```
void calculate_completion_time(struct Process p[], int n) {  
    p[0].completion_time = p[0].arrival_time + p[0].burst_time;
```



```
for(int i = 1; i < n; i++) {

    if(p[i].arrival_time > p[i-1].completion_time) {
        p[i].completion_time = p[i].arrival_time + p[i].burst_time;
    } else {
        p[i].completion_time = p[i-1].completion_time + p[i].burst_time;
    }
}

}

void calculate_turnaround_time(struct Process p[], int n) {
    for(int i = 0; i < n; i++) {
        p[i].turnaround_time = p[i].completion_time - p[i].arrival_time; // TAT = CT - AT
    }
}

void calculate_waiting_time(struct Process p[], int n) {
    for(int i = 0; i < n; i++) {
        p[i].waiting_time = p[i].turnaround_time - p[i].burst_time; // WT = TAT - BT
    }
}

float calculate_average_waiting_time(struct Process p[], int n) {
    float sum = 0.0;
    for(int i = 0; i < n; i++) {
        sum += p[i].waiting_time;
    }

    return sum / n;
```

```
}

float calculate_average_turnaround_time(struct Process p[], int n) {
    float sum = 0.0;
    for(int i = 0; i < n; i++) {
        sum += p[i].turnaround_time;
    }
    return sum / n;
}

void display_results(struct Process p[], int n) {
    printf("\nProcess  Arrival Time  Burst Time  Completion Time  Turnaround Time  Waiting Time\n");

    for(int i = 0; i < n; i++) {
        printf("Process %d  %d      %d      %d      %d      %d\n",
            p[i].pid, p[i].arrival_time, p[i].burst_time, p[i].completion_time, p[i].turnaround_time,
            p[i].waiting_time);
    }

    printf("\nAverage Waiting Time: %.2f", calculate_average_waiting_time(p, n));
    printf("\nAverage Turnaround Time: %.2f\n", calculate_average_turnaround_time(p, n));
}

int main() {
    int n;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    struct Process processes[n];
```

```
for(int i = 0; i < n; i++) {  
    processes[i].pid = i + 1;  
    printf("Enter Burst Time for Process %d: ", i + 1);  
    scanf("%d", &processes[i].burst_time);  
    printf("Enter Arrival Time for Process %d: ", i + 1);  
    scanf("%d", &processes[i].arrival_time);  
}  
  
sort_by_arrival(processes, n);  
calculate_completion_time(processes, n);  
calculate_turnaround_time(processes, n);  
calculate_waiting_time(processes, n);  
display_results(processes, n);  
  
return 0;  
}
```

```
Enter number of processes: 6  
Enter Burst Time for Process 1: 6  
Enter Arrival Time for Process 1: 0  
Enter Burst Time for Process 2: 8  
Enter Arrival Time for Process 2: 2  
Enter Burst Time for Process 3: 7  
Enter Arrival Time for Process 3: 2  
Enter Burst Time for Process 4: 3  
Enter Arrival Time for Process 4: 3  
Enter Burst Time for Process 5: 4  
Enter Arrival Time for Process 5: 5  
Enter Burst Time for Process 6: 2  
Enter Arrival Time for Process 6: 6
```

Process	Arrival Time	Burst Time	Completion Time	Turnaround Time	Waiting Time
Process 1	0	6	6	6	0
Process 2	2	8	14	12	4
Process 3	2	7	21	19	12
Process 4	3	3	24	21	18
Process 5	5	4	28	23	19
Process 6	6	2	30	24	22

```
Average Waiting Time: 12.50  
Average Turnaround Time: 17.50
```