

Department of Computer Science & Engineering

**SUBJECT: Operating System Lab
(21CSH-242)**

**B.E. II Year – III Semester
(Branch: CSE-IBM)
(Batch: 2021-25)**



LAB FILE

**Prepared By:
Dr. Madan Lal Saini
E13485
Session- July-Dec, 2022**

Table of Contents

Exp. No.	Name of the Experiments
UNIT-I	
1.	1.1 Installation of Linux operating system. 1.2 Study of basic Linux commands.
2.	2.1 Programs using the following system calls of Linux operating system: fork, getpid, getppid, exit, wait, and close. 2.2 Use of Linux file related commands like ls, grep, cat, etc.
3.	3.1 Programs using the I/O system calls of Linux operating system (open, read, write etc). 3.2 Study of basics of shell programming.
UNIT-II	
4.	4.1 Write a program to show the use of echo. 4.2 Write a program to read the keywords in shell programming.
5.	5.1 Write a program using arithmetic operators in shell programming. 5.2 Write a program using relational operators in shell programming.
6.	6.1 Write a program using Boolean operators in shell programming. 6.2 Write programs using control structures in shell programming.
7.	Write a program to demonstrate the difference between while and until statement.
UNIT-III	
8.	8.1 Simulation of First come first serve CPU scheduling algorithm. 8.2 Simulation of Shortest job first CPU scheduling algorithm.
9.	9.1 Simulation of Round Robin CPU scheduling algorithm. 9.2 Simulation of Priority based CPU scheduling algorithm.
10.	Simulate the Bankers algorithm for deadlock avoidance and deadlock prevention.

Experiment-1

Exp1.1

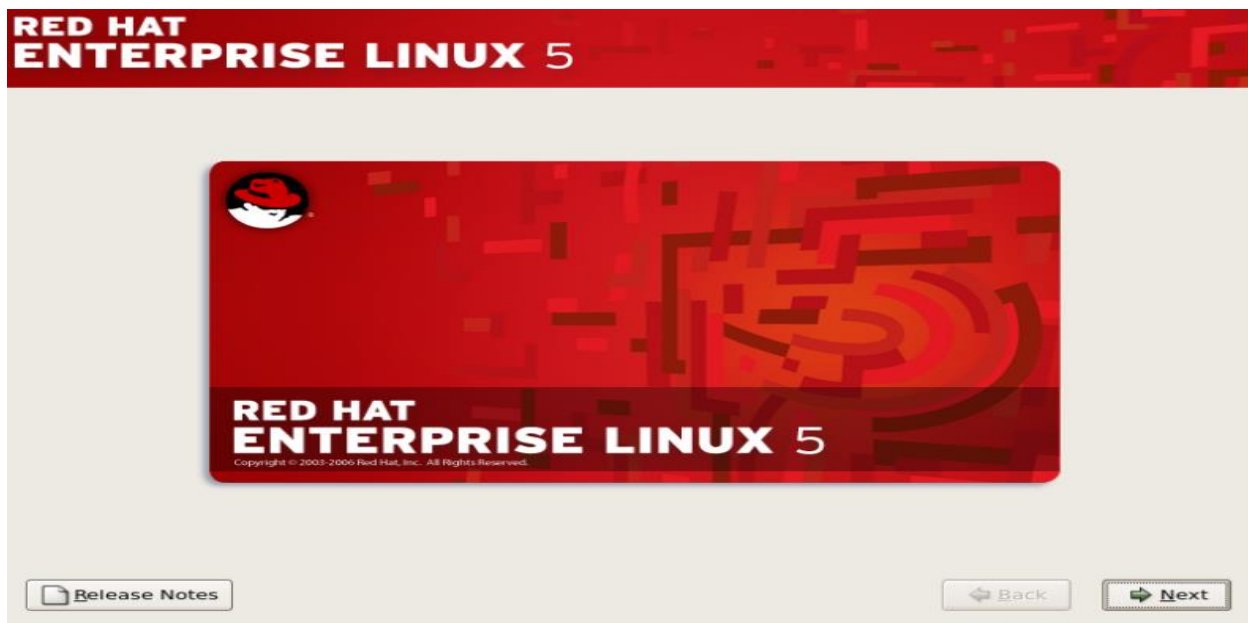
AIM: Installation of Linux operating system.

Step 1: Firstly empty any one drive from our computer and right click on the computer select manage and then select disk fragmentation and then delete that drive which is empty.

Step 2: Then put the Red Hat DVD into the DVD drive and reboot the system, check the BIOS setting the computer from DVD is selected if not selected. Then DVD runs on it.

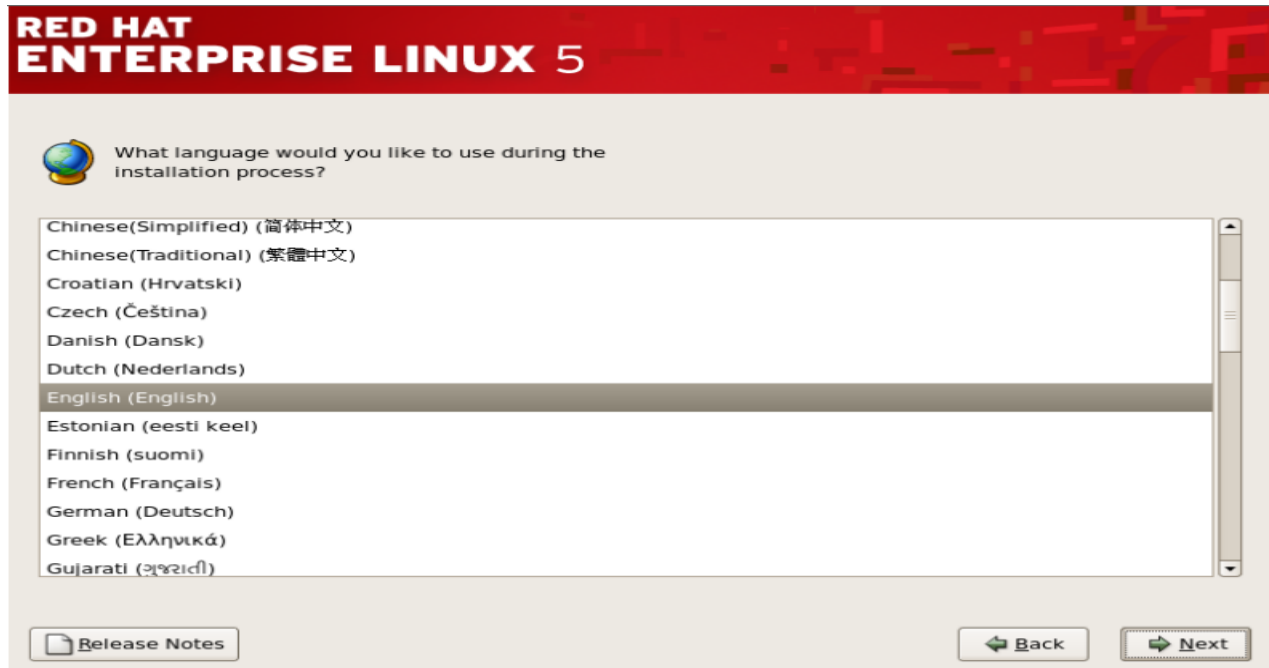
Step 3: Then seen a) In text mode we have to need to type Linux ascue.
b) In Graphical mode we have to press enter.

Step 4: Welcome to Red Hat Enterprise Linux: The welcome screen does not prompt you for any input. From this screen you can access the release notes for Red Hat Enterprise 5.0.0 by clicking on the release button.

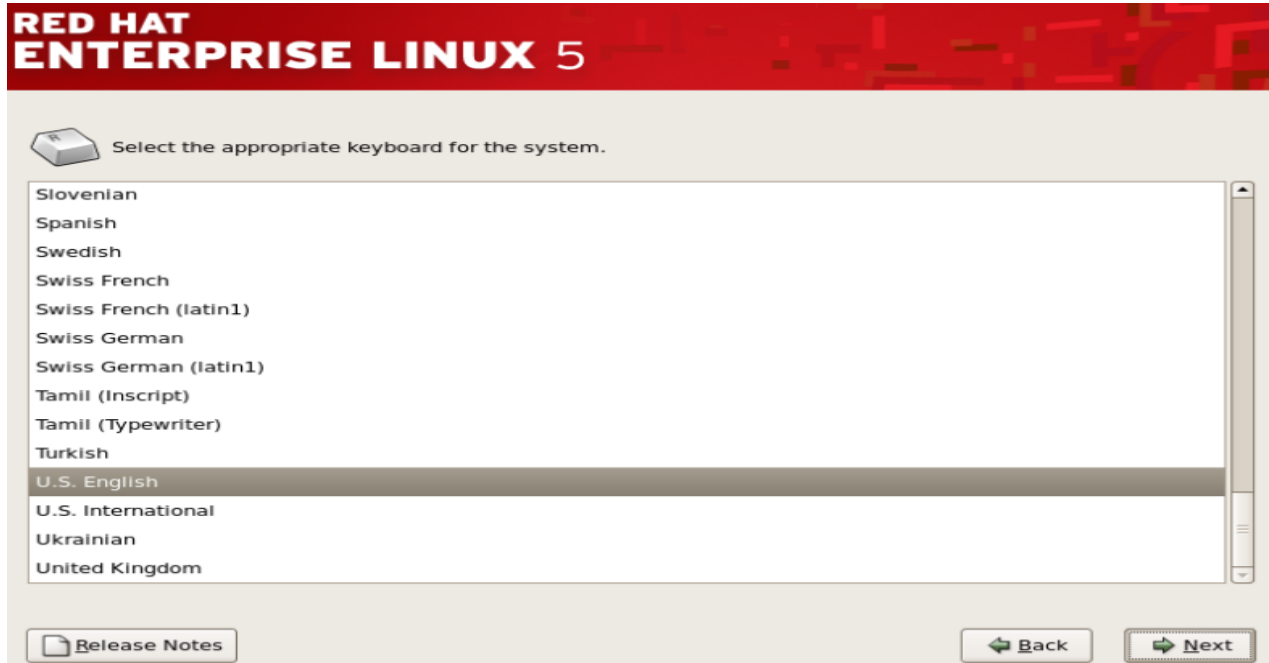


Click on the next button to continue

Step 5: Language Selection: The language you select here will become the default language for the operating system once it is installed. Selecting the appropriate language also helps target your time zone configuration later in the installation. The installation program tries to define the appropriate time zone based on what you specify on this screen.



Step 6: Keyboard Configuration: Using your mouse, select the correct layout type for example U.S. English for the keyboard. You would prefer to use for the installation and as the system default.

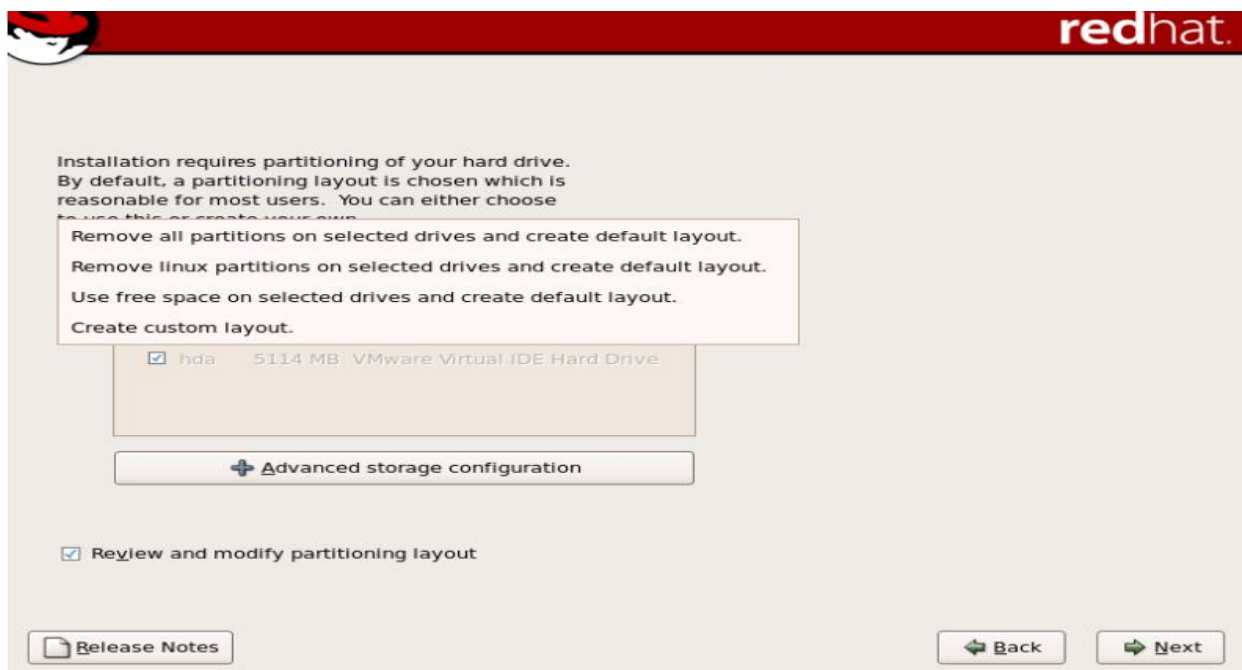


Step 7: Enter the Installation Number: If you have a number then put on it otherwise skip this option



Step 8: Create Default Layout options are

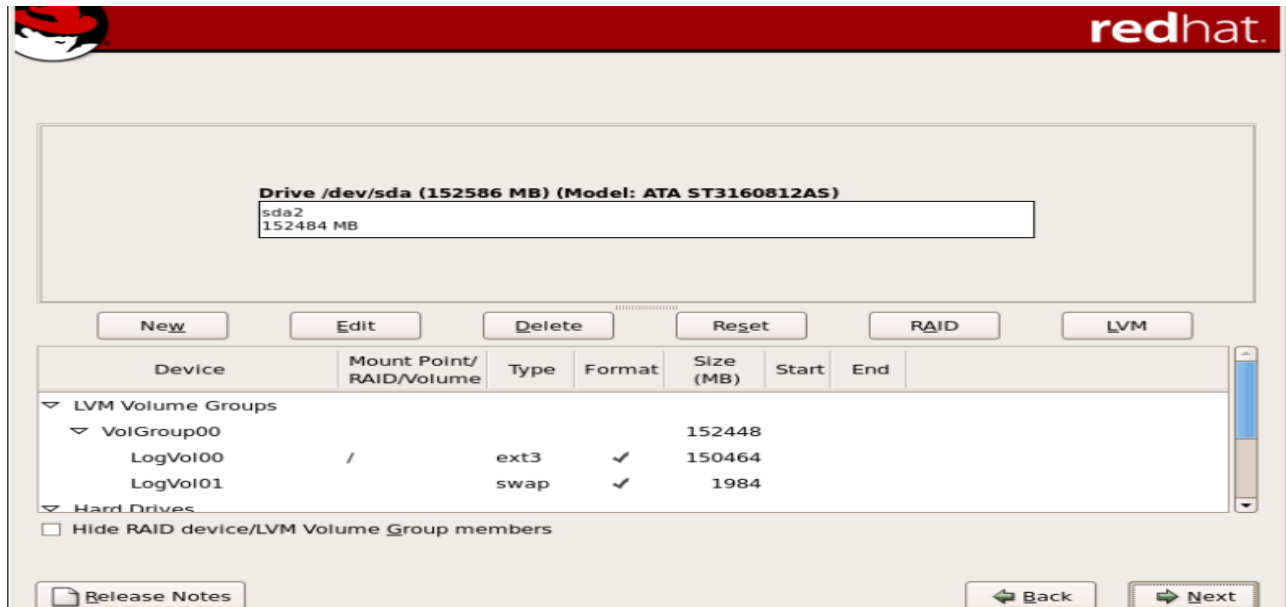
- Remove all partitions on selected drives and create default layout: Select this option to remove all partitions on your hard drives
- Remove Linux partitions on selected drives and create default layout: Select this option to remove Linux partitions on your hard drives
- Use free space on selected drives and create default layout: Select this option to retain your current data and partitions, assuming you have enough free space.
- Create custom layout: If you delete free drive then select create custom layout.



CHANDIGARH UNIVERSITY, GHARUAN (MOHALI)

Select the Create Custom Layout. Then click next button. If you have two hard disk then select which one.

Step 9: Choose custom layout, you must tell the installation program where to install Red Hat 5.0. This is done by defining mount point for one or more disk partitions in which Red Hat is installed. You may also need to create or delete partition at this time.



Mount Point: Enter the partition's mount point. For example, if this partition should be the root partition, enter / for the root

/boot for the /boot partition, and
/swap files double of ram.

Pull-down menu to choose the correct mount point for your partition. For a swap partition the Mount point should not be set - setting the files system type to swap is sufficient.

File System Type: Using the pull-down menu, select the appropriate file system type for this partition. For more information on file system types, “File System Types”.

Allowable Drives: This field contains a list of the hard disks installed on your system. If a Hard disk's box is highlighted, then a desired partition can be created on that hard disk. If the box is not checked, then the partition will never be created on that hard disk. By using different checkbox settings, you can have Disk Druid place partitions where you need them, or let Disk Druid decide where partitions should go.

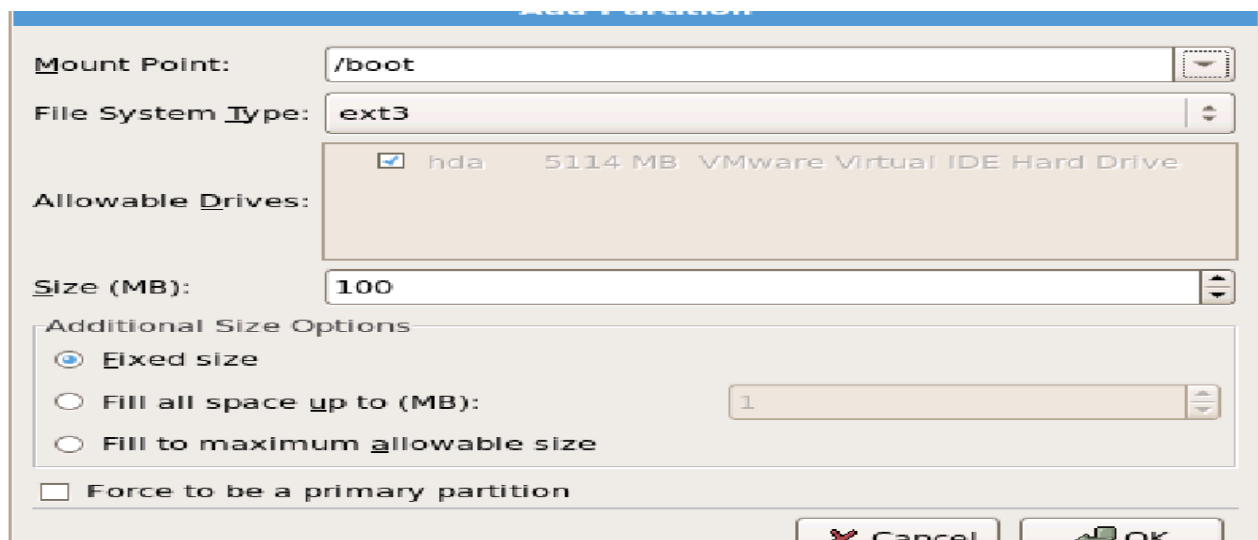
Size (MB): Enter the size (in megabytes) of the partition. Note, this field starts with 100 MB; unless changed; only a 100 MB partition will be created.

The file systems are:

ext2 — An ext2 file system supports standard Unix file types (regular files, directories, symbolic links, etc). It provides the ability to assign long file names, up to 255 characters.

ext3 — The ext3 file system is based on the ext2 file system and has one main advantage — journaling. Using a journaling file system reduces time spent recovering a file system after a crash as there is no need to fsck the file system. The ext3 file system is selected by default and is highly recommended.


Physical volume (LVM) — Creating one or more physical volume (LVM) partitions allows you to create an LVM logical volume. LVM can improve performance when using physical disks. For more information regarding LVM, refer to the Red Hat Enterprise Linux Deployment Guide.



Step 10: To boot the system without boot media, you usually need to install a boot loader. A boot loader

is the first software program that runs when a computer starts. It is responsible for loading and transferring control to the operating system kernel software. The kernel, in turn, initializes the rest of the operating system.

GRUB (Grand Unified Boot loader), which is installed by default, is a very powerful boot loader. GRUB can load a variety of free operating systems, as well as proprietary operating system. With chain-loading (the mechanism for loading unsupported operating systems, such as DOS or Windows, by loading another boot loader).



redhat.

☒ The GRUB boot loader will be installed on /dev/hda.
☐ No boot loader will be installed.

You can configure the boot loader to boot other operating systems. It will allow you to select an operating system to boot from the list. To add additional operating systems, which are not automatically detected, click 'Add.' To change the operating system booted by default, select 'Default' by the desired operating system.

Default	Label	Device
<input checked="" type="checkbox"/>	Red Hat Enterprise Linux Server	/dev/VolGroup00/LogVol00

A boot loader password prevents users from changing options passed to the kernel. For greater system security, it is recommended that you set a password.


☐ Use a boot loader password

☒ Configure advanced boot loader options

Click the next button

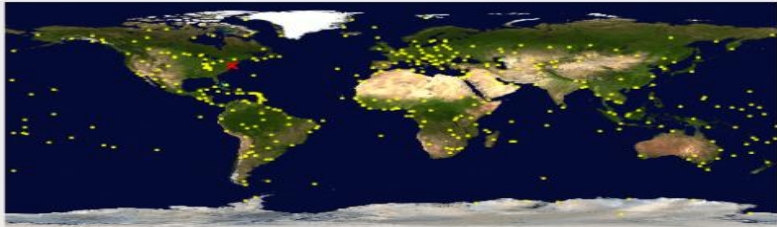
Step 11: Time Zone Configuration: Set your time zone by selecting the city closest to your computer's physical location. Click on the map to zoom in to a particular geographical region of the world. From here there are two ways for you to select your time zone:

- Using your mouse, click on the interactive map to select a specific city (represented by a yellow dot). A red X appears indicating your selection.
- You can also scroll through the list at the bottom of the screen to select your time zone. Using your mouse, click on a location to highlight your selection.



redhat.

Please click into the map to choose a region:



☒ System clock uses UTC

Step 12: Set Root Password: Setting up a root account and password is one of the most important steps during your installation. Your root account is similar to the administrator account used on Windows NT



**RED HAT
ENTERPRISE LINUX 5**


 The root account is used for administering the system. Enter a password for the root user.

Root Password:

Confirm:

[Release Notes](#) [Back](#) [Next](#)

Step 13: You want to select web server and software development and choose custom now and choose custom later, any one of them.



**RED HAT
ENTERPRISE LINUX 5**

The default installation of Red Hat Enterprise Linux Server includes a set of software applicable for general internet usage. What additional tasks would you like your system to include support for?

☐ Software Development

☐ Web server

You can further customize the software selection now, or after install via the software management application.

☐ Customize later ☒ Customize now

Step 14: And installing the Linux.

Step 15: And rebooting the system.

Step 16: Firewall disabled or enabled.

Step 17: SELN disabled or enabled.

Step 18: Then continue means Linux is installed.

Experiment-1

Exp1.2

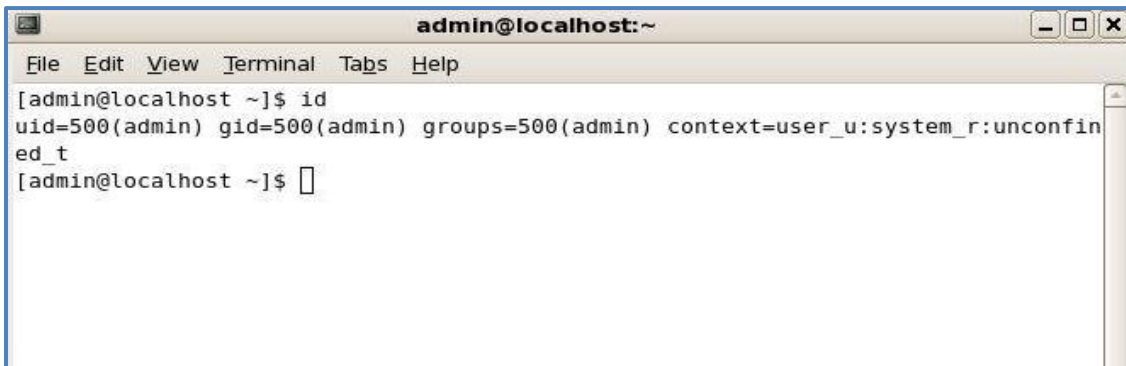
AIM: Study of basic LINUX commands.

List of some of the LINUX COMMANDS are given below:

1. **Id**

Syntax: id

Id command is used to find a user's UID (user ID) or GID (group ID) and other information in Linux. The main purpose of id command is to **displays the system identifications of a specified user.**

A terminal window titled 'admin@localhost:~' with a menu bar (File, Edit, View, Terminal, Tabs, Help). The command 'id' has been executed, showing the output: 'uid=500(admin) gid=500(admin) groups=500(admin) context=user_u:system_r:unconfined_t'. The prompt '[admin@localhost ~]\$' is shown at the end of the line.

```
admin@localhost:~  
File Edit View Terminal Tabs Help  
[admin@localhost ~]$ id  
uid=500(admin) gid=500(admin) groups=500(admin) context=user_u:system_r:unconfined_t  
[admin@localhost ~]$
```

2. **Who**

Syntax: who

The who command prints information about all users who are currently logged in.

A terminal window titled 'admin@localhost:~' with a menu bar (File, Edit, View, Terminal, Tabs, Help). The command 'who' has been executed, showing the output: 'admin pts/0 2015-08-14 02:04 (:0.0)'. The prompt '[admin@localhost ~]\$' is shown at the end of the line.

```
admin@localhost:~  
File Edit View Terminal Tabs Help  
[admin@localhost ~]$ who  
admin pts/0 2015-08-14 02:04 (:0.0)  
[admin@localhost ~]$
```

3. **Who am i**

Syntax: who am i

The who am i command prints the user name (i.e., login name) of the owner of the current login session on the monitor screen.



```
admin@localhost:~  
File Edit View Terminal Tabs Help  
[admin@localhost ~]$ who i am  
admin pts/0 2015-08-14 02:04 (:0.0)  
[admin@localhost ~]$
```

4. Date

Syntax: date

Date command is helpful to display date in several formats. It also allows you to set systems date and time. When you execute date command without any option, it will display the current date and time.



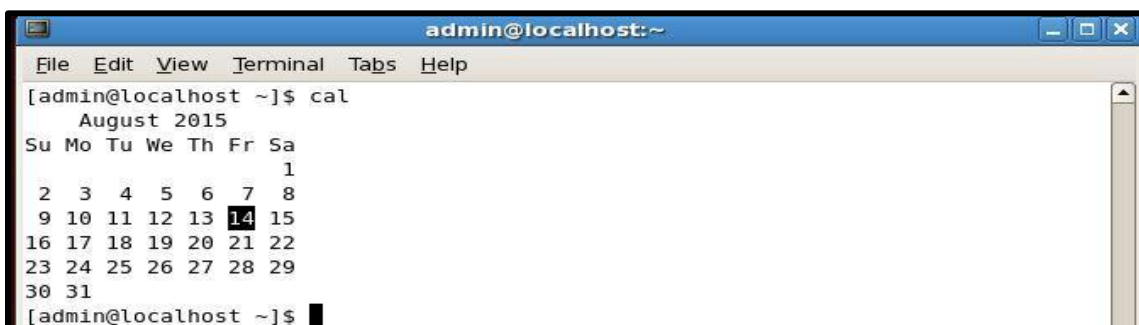
```
[admin@localhost ~]$ date  
Fri Aug 14 02:08:29 PDT 2015  
[admin@localhost ~]$
```

5. Cal

Display a conveniently-formatted calendar from the command line.

i. **Syntax:** cal

If no options are given, cal displays the current month at the command line.



```
admin@localhost:~  
File Edit View Terminal Tabs Help  
[admin@localhost ~]$ cal  
August 2015  
Su Mo Tu We Th Fr Sa  
1  
2 3 4 5 6 7 8  
9 10 11 12 13 14 15  
16 17 18 19 20 21 22  
23 24 25 26 27 28 29  
30 31  
[admin@localhost ~]$
```

ii. **Syntax:** cal [year]

Display a calendar for that entire year.

File Edit View Terminal Tabs Help

				1	2	3	4																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																								
--	--	--	--	---	---	---	---	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

6. Clear

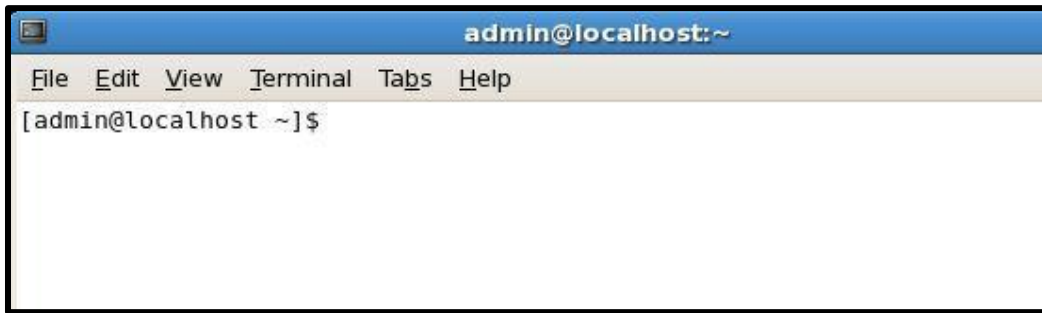
Syntax: clear

Clear command is used to clear the screen. The clear command does not affect files or jobs, it simply clears the clutter from your terminal screen. `CLEAR` doesn't erase your terminal scroll buffer, so if you need to see something that was on your screen a while ago, you can still use your mouse wheel to scroll back.

```

admin@localhost:~
File Edit View Terminal Tabs Help
[admin@localhost ~]$ id
uid=500(admin) gid=500(admin) groups=500(admin) context=user_u:system_r:unconfined_t
[admin@localhost ~]$ who i am
admin pts/0 2015-08-14 02:04 (:0.0)
[admin@localhost ~]$ cal
August 2015
Su Mo Tu We Th Fr Sa
1
2 3 4 5 6 7 8
9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
[admin@localhost ~]$ clear

```

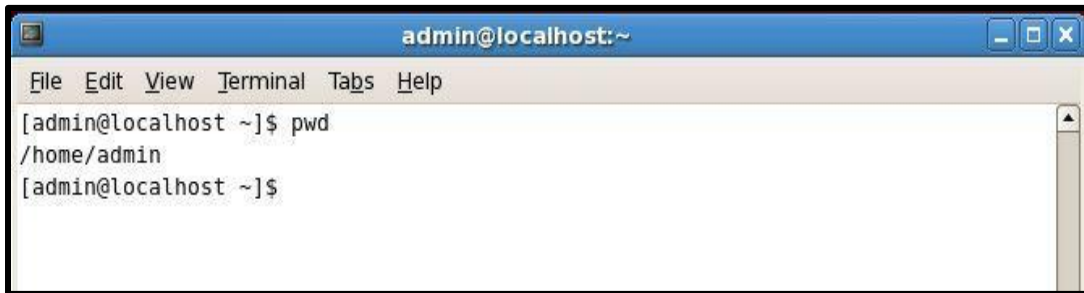


```
admin@localhost:~  
File Edit View Terminal Tabs Help  
[admin@localhost ~]$
```

7. Pwd

Syntax: pwd

‘pwd’ stands for ‘Print Working Directory’. As the name states, command ‘pwd’ prints the current working directory or simply the directory user is, at present. It prints the current directory name with the complete path starting from root (/).

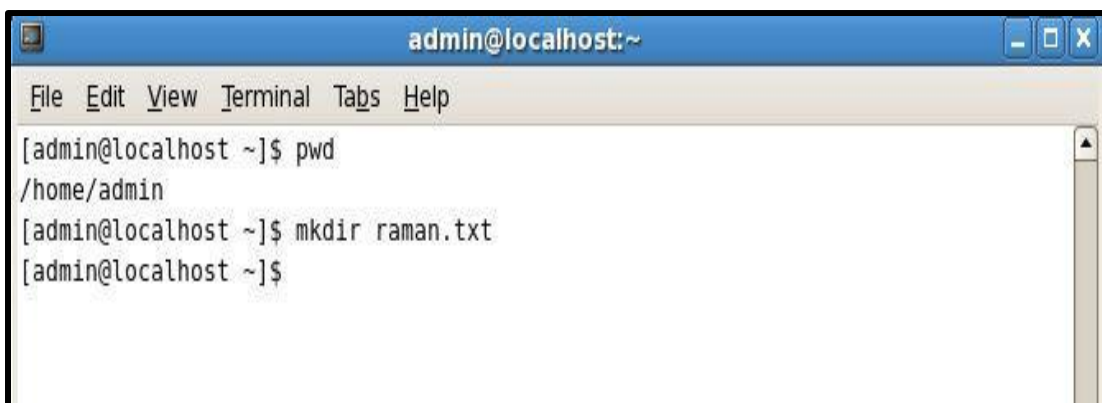


```
admin@localhost:~  
File Edit View Terminal Tabs Help  
[admin@localhost ~]$ pwd  
/home/admin  
[admin@localhost ~]$
```

8. Mkdir

Syntax: mkdir directory_name

It is a short for "make directory", mkdir is used to create directories on a file system. If the specified DIRECTORY does not already exist, mkdir creates it. More than one DIRECTORY may be specified when calling mkdir.



```
admin@localhost:~  
File Edit View Terminal Tabs Help  
[admin@localhost ~]$ pwd  
/home/admin  
[admin@localhost ~]$ mkdir raman.txt  
[admin@localhost ~]$
```

9. Cd

The cd command is used to change the current directory (i.e., the directory in which the user is currently working) in Linux and other Unix-like operating systems.

i. **Syntax:** cd [directory]

When a directory name is provided, cd changes the current directory to it. When used without specifying any directory name, cd returns the user to the previous current directory.

A terminal window titled 'admin@localhost:~/raman.txt' with a menu bar (File, Edit, View, Terminal, Tabs, Help). The terminal shows the following commands and output:

```
[admin@localhost ~]$ pwd
/home/admin
[admin@localhost ~]$ mkdir raman.txt
[admin@localhost ~]$ cd raman.txt
[admin@localhost raman.txt]$
```

ii. **Syntax:** cd ..

This command helps the user to change to the parent of the current directory. On LINUX like operating system parent directory of the current directory is represented by two consecutive dots.

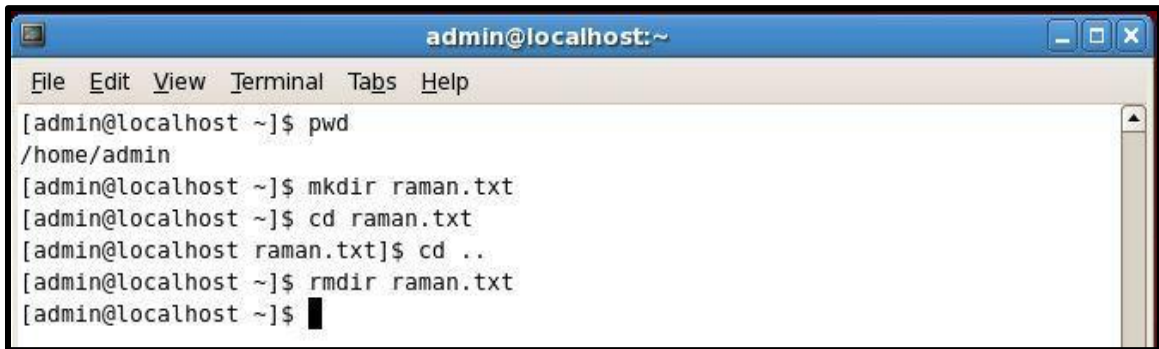
A terminal window titled 'admin@localhost:~' with a menu bar (File, Edit, View, Terminal, Tabs, Help). The terminal shows the following commands and output:

```
[admin@localhost ~]$ pwd
/home/admin
[admin@localhost ~]$ mkdir raman.txt
[admin@localhost ~]$ cd raman.txt
[admin@localhost raman.txt]$ cd ..
[admin@localhost ~]$
```

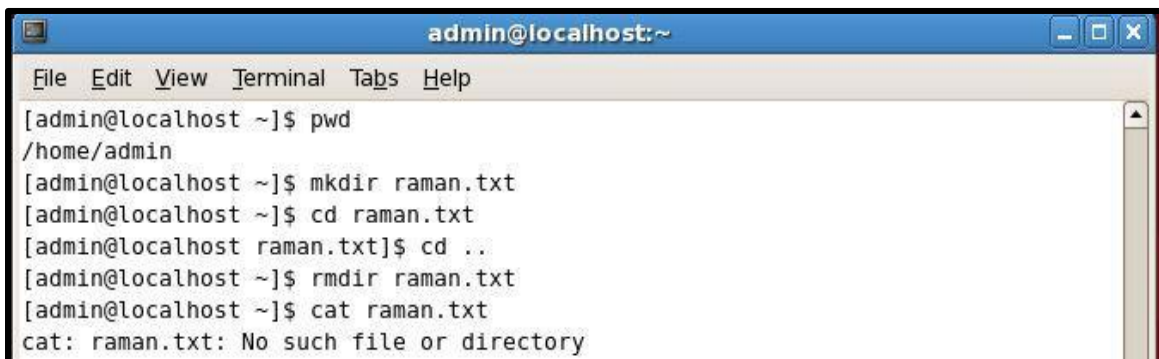
10. Rmdir

Syntax: rmdir directory_name

The rmdir command is used to remove empty directories in Linux. If a directory contains files or sub directories, then the rmdir command fails.



```
admin@localhost:~  
File Edit View Terminal Tabs Help  
[admin@localhost ~]$ pwd  
/home/admin  
[admin@localhost ~]$ mkdir raman.txt  
[admin@localhost ~]$ cd raman.txt  
[admin@localhost raman.txt]$ cd ..  
[admin@localhost ~]$ rmdir raman.txt  
[admin@localhost ~]$
```



```
admin@localhost:~  
File Edit View Terminal Tabs Help  
[admin@localhost ~]$ pwd  
/home/admin  
[admin@localhost ~]$ mkdir raman.txt  
[admin@localhost ~]$ cd raman.txt  
[admin@localhost raman.txt]$ cd ..  
[admin@localhost ~]$ rmdir raman.txt  
[admin@localhost ~]$ cat raman.txt  
cat: raman.txt: No such file or directory
```

11. Ls

Syntax: ls

This command is usually used to view the contents of current directory. So, when you run this command, the files and sub-directories included under the current directory will be listed before you.



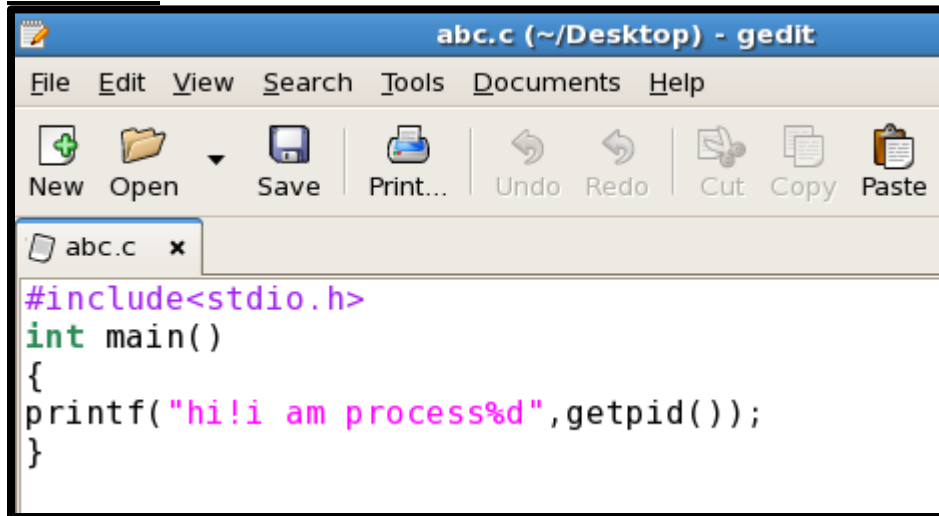
```
admin@localhost:~  
File Edit View Terminal Tabs Help  
[admin@localhost ~]$ pwd  
/home/admin  
[admin@localhost ~]$ mkdir raman.txt  
[admin@localhost ~]$ cd raman.txt  
[admin@localhost raman.txt]$ cd ..  
[admin@localhost ~]$ rmdir raman.txt  
[admin@localhost ~]$ cat raman.txt  
cat: raman.txt: No such file or directory  
[admin@localhost ~]$ ls  
a.txt Desktop  
[admin@localhost ~]$
```


Experiment-2

Exp2.1

AIM: Programs using the following system calls of Linux operating system: fork, getpid, getppid, exit, wait, and close.

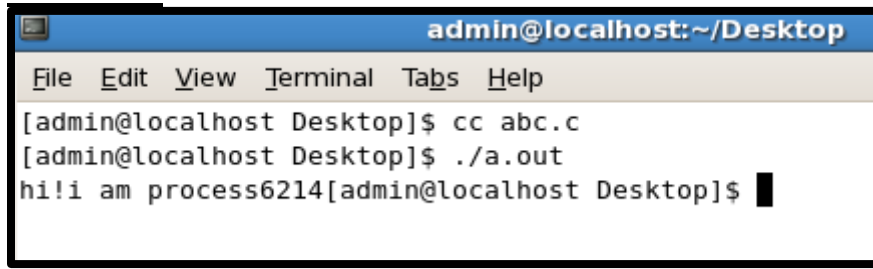
GET PID



The screenshot shows a gedit editor window titled 'abc.c (~/Desktop) - gedit'. The menu bar includes File, Edit, View, Search, Tools, Documents, and Help. The toolbar contains icons for New, Open, Save, Print..., Undo, Redo, Cut, Copy, and Paste. The code editor displays the following C code:

```
#include<stdio.h>
int main()
{
printf("hi!i am process%d",getpid());
}
```

OUTPUT



The screenshot shows a terminal window titled 'admin@localhost:~/Desktop'. The menu bar includes File, Edit, View, Terminal, Tabs, and Help. The terminal output is as follows:

```
[admin@localhost Desktop]$ cc abc.c
[admin@localhost Desktop]$ ./a.out
hi!i am process6214[admin@localhost Desktop]$
```

CODE

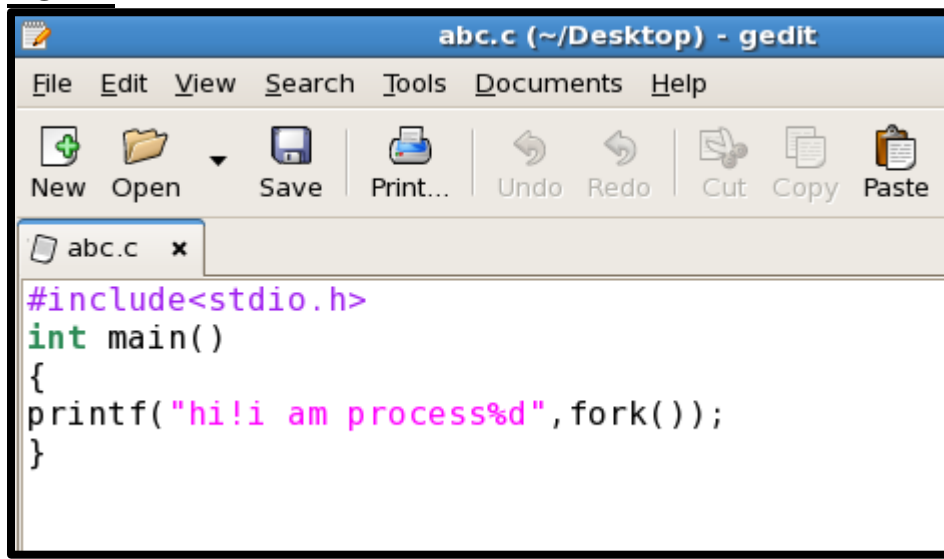
```
#include<stdio.h>
main(intarc,char*ar[])
{
intpid;
char s[100];
pid=fork();
if(pid<0)
printf("error");
else if(pid>0)
{
wait(NULL);
```

```
printf("\n Parent Process:\n");
printf("\n\n\tParent Process id:%d\n\t\n",getpid());
execlp("cat","cat",ar[1],(char*)0);
error("can't execute cat %s",ar[1]);
}
else
{
printf("\nChild process:");
printf("\n\n\tChildprocess parent id:\n\t%d",getppid());
sprintf(s,"
\n\tChild process id :
\n\t%d",getpid());
write(1,s,strlen(s));
printf(" ");
printf(" ");
printf(" ");
execvp(ar[2],&ar[2]);
error("can't execute %s",ar[2]);
}
}
```

OUTPUT

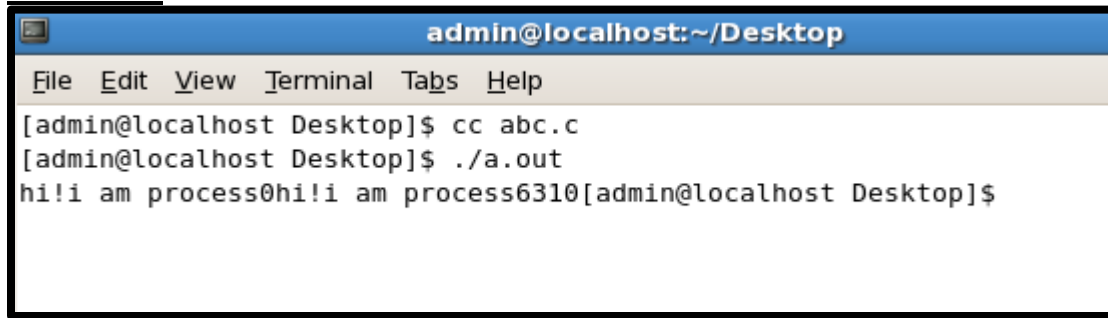
```
[root@localhost ~]# ./a.outtst date
Child process:
Child process id :
3137 Sat A
pr 10 02:45:32 IST 2010
Parent Process:
Parent Process id:3136
sd
dsaASD[root@localhost ~]# cat tst
sd
dsaASD
```

FORK



```
abc.c (~/Desktop) - gedit
File Edit View Search Tools Documents Help
New Open Save Print... Undo Redo Cut Copy Paste
abc.c x
#include<stdio.h>
int main()
{
printf("hi!i am process%d", fork());
}
```

OUTPUT



```
admin@localhost:~/Desktop
File Edit View Terminal Tabs Help
[admin@localhost Desktop]$ cc abc.c
[admin@localhost Desktop]$ ./a.out
hi!i am process0hi!i am process6310[admin@localhost Desktop]$
```

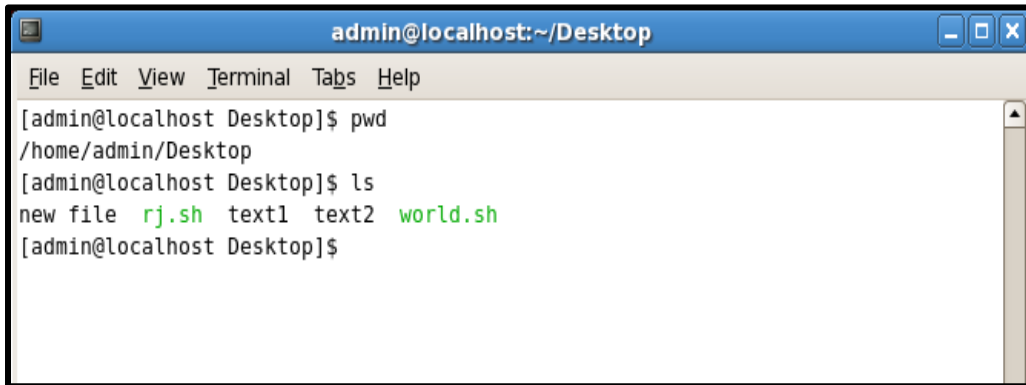
EXPERIMENT-2

Exp2.2

AIM: Use of Linux file related commands like ls, grep, cat, etc.

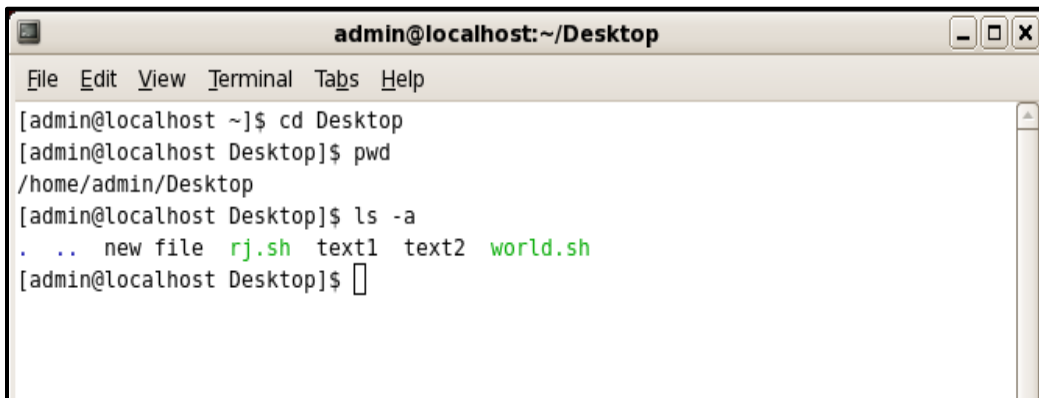
1. **Ls**: Lists the contents of a directory. List information about the Files (the current directory by default).

Syntax: ls[OPTIONS]...[FILE]..



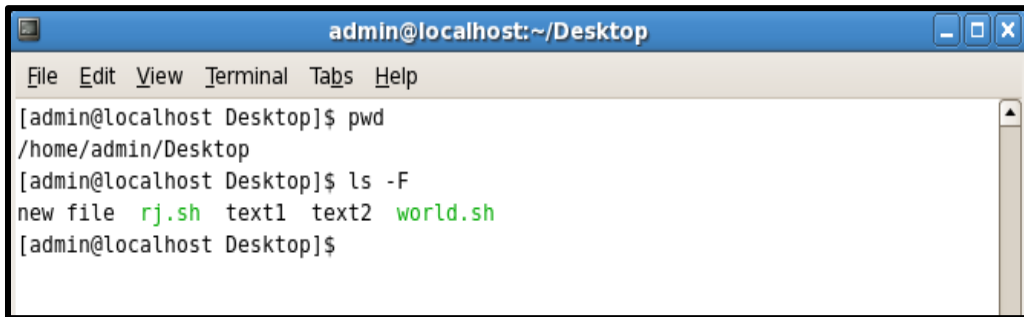
```
admin@localhost:~/Desktop
File Edit View Terminal Tabs Help
[admin@localhost Desktop]$ pwd
/home/admin/Desktop
[admin@localhost Desktop]$ ls
new file  rj.sh  text1  text2  world.sh
[admin@localhost Desktop]$
```

ls -a: List all files and folders including hidden file starting with ‘.’.



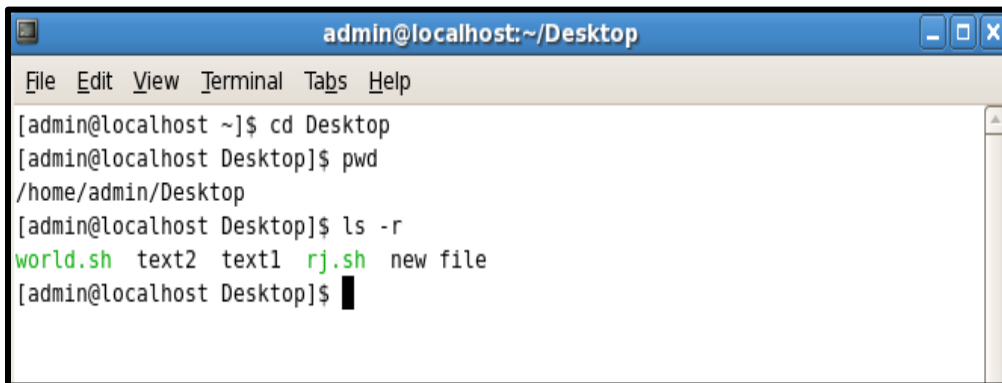
```
admin@localhost:~/Desktop
File Edit View Terminal Tabs Help
[admin@localhost ~]$ cd Desktop
[admin@localhost Desktop]$ pwd
/home/admin/Desktop
[admin@localhost Desktop]$ ls -a
.  ..  new file  rj.sh  text1  text2  world.sh
[admin@localhost Desktop]$
```

ls -F: Using -F option with ls command, will add the ‘/’ Character at the end of each directory.



```
admin@localhost:~/Desktop
File Edit View Terminal Tabs Help
[admin@localhost Desktop]$ pwd
/home/admin/Desktop
[admin@localhost Desktop]$ ls -F
new file  rj.sh  text1  text2  world.sh
[admin@localhost Desktop]$
```

ls -r: Using **ls -r** option with **ls** command will display files and directories in reverse order.



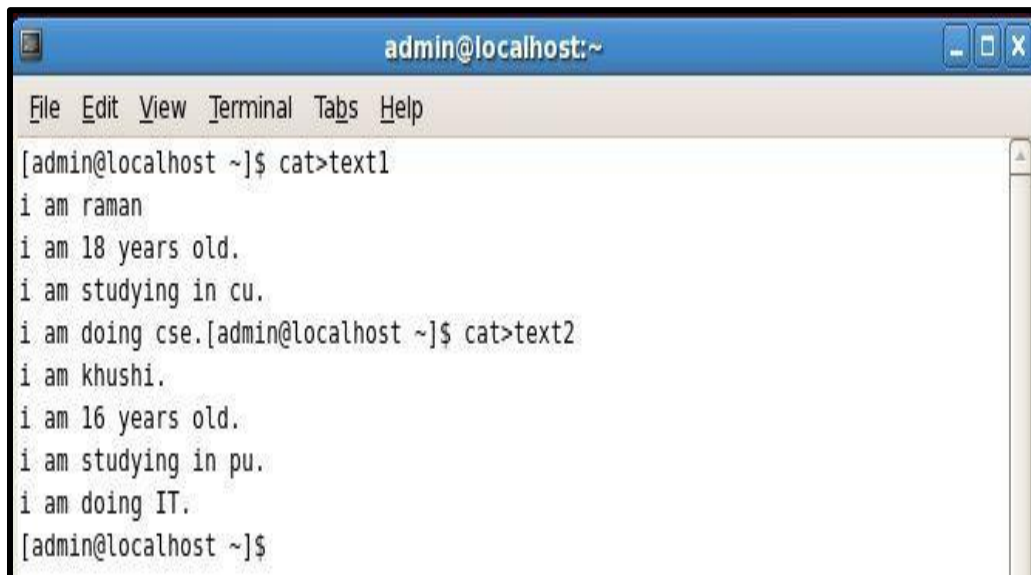
```
admin@localhost:~/Desktop
File Edit View Terminal Tabs Help
[admin@localhost ~]$ cd Desktop
[admin@localhost Desktop]$ pwd
/home/admin/Desktop
[admin@localhost Desktop]$ ls -r
world.sh  text2  text1  rj.sh  new file
[admin@localhost Desktop]$
```

2. **Cat:** The **cat** (short for “**concatenate**”) command is one of the most frequently used command in Linux/Unix like operating systems. **cat** command allows us to create single or multiple files, view contain of file, concatenate files and redirect output in terminal or files.

Syntax: **cat** [options] [filenames] [-] [filenames]

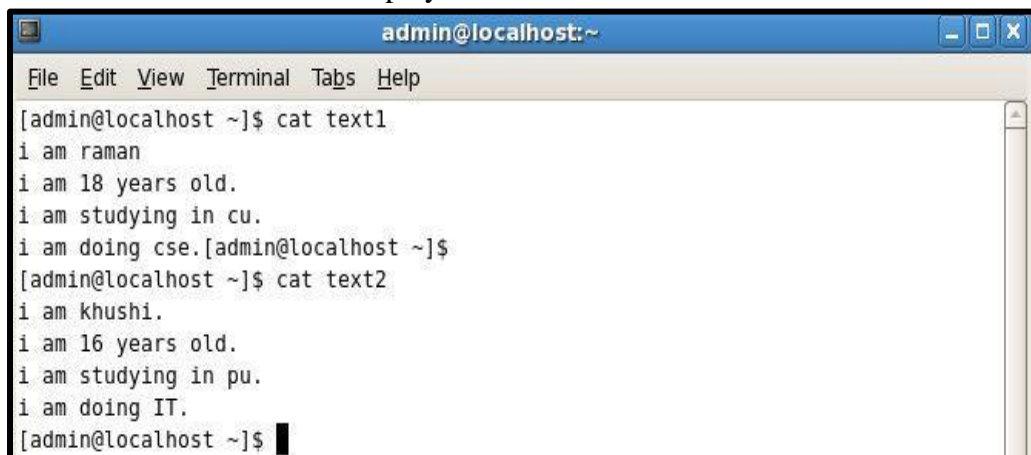
cat > filename: Typing the command **cat** followed by the output redirection operator and a file name on the same line, pressing ENTER to move to the next line, then typing some text and finally pressing ENTER again causes the text to be written to that file. This command create and open file in writing mode.

Awaits input from user, type desired text and press **CTRL+D** (hold down **Ctrl Key** and type ‘**d**’) to exit.



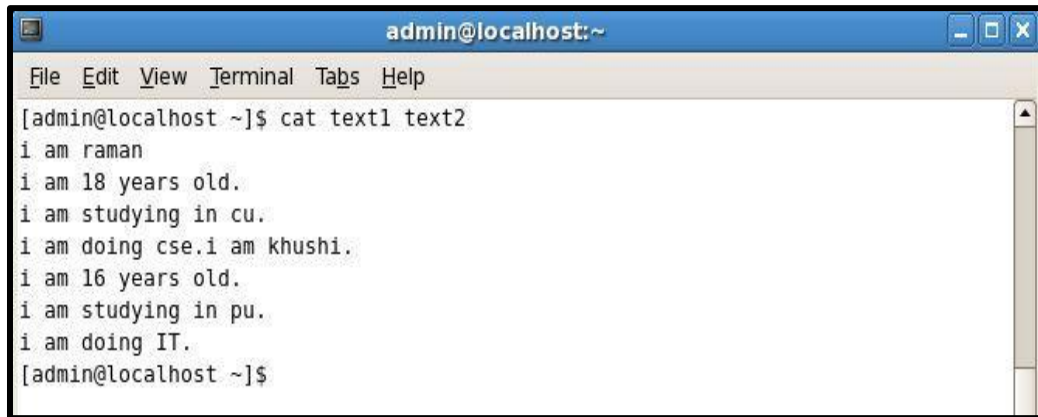
```
admin@localhost:~  
File Edit View Terminal Tabs Help  
[admin@localhost ~]$ cat>text1  
i am raman  
i am 18 years old.  
i am studying in cu.  
i am doing cse.[admin@localhost ~]$ cat>text2  
i am khushi.  
i am 16 years old.  
i am studying in pu.  
i am doing IT.  
[admin@localhost ~]$
```

cat filename: It is used to display the content of a file on the monitor screen.



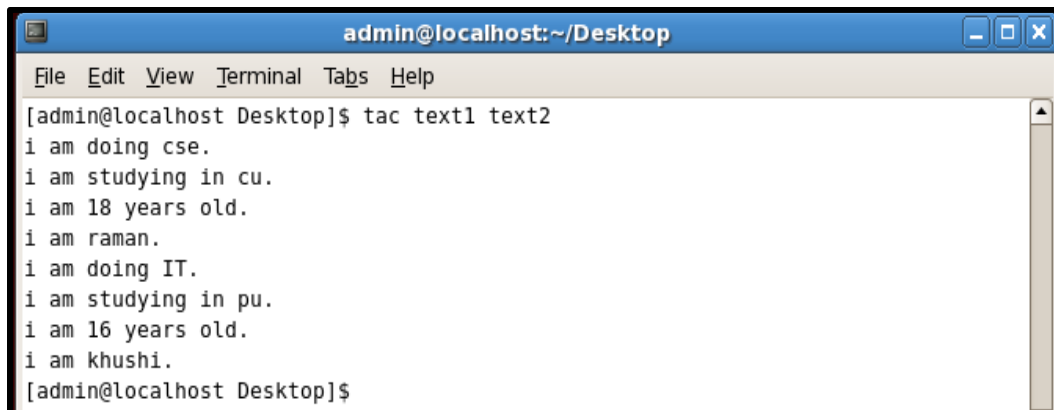
```
admin@localhost:~  
File Edit View Terminal Tabs Help  
[admin@localhost ~]$ cat text1  
i am raman  
i am 18 years old.  
i am studying in cu.  
i am doing cse.[admin@localhost ~]$  
[admin@localhost ~]$ cat text2  
i am khushi.  
i am 16 years old.  
i am studying in pu.  
i am doing IT.  
[admin@localhost ~]$
```

cat file1 file2: This command is used to concatenate the contents of files. The contents of each file will be displayed on the monitor screen (which, again, is standard output, and thus the destination of the output in the absence of redirection) starting on a new line and in the order that the file names appear in the command.

A terminal window titled 'admin@localhost:~' with a menu bar (File, Edit, View, Terminal, Tabs, Help). The command '[admin@localhost ~]\$ cat text1 text2' has been executed. The output is concatenated text from two files: 'i am raman', 'i am 18 years old.', 'i am studying in cu.', 'i am doing cse.i am khushi.', 'i am 16 years old.', 'i am studying in pu.', and 'i am doing IT.'. The prompt '[admin@localhost ~]\$' is visible at the bottom.

```
admin@localhost:~  
File Edit View Terminal Tabs Help  
[admin@localhost ~]$ cat text1 text2  
i am raman  
i am 18 years old.  
i am studying in cu.  
i am doing cse.i am khushi.  
i am 16 years old.  
i am studying in pu.  
i am doing IT.  
[admin@localhost ~]$
```

tac file1 file2: This command is used to display the contents of files after concatenating in the order that firstly the contents of last entered file will be shown and the contents of first entered file will be shown at end.

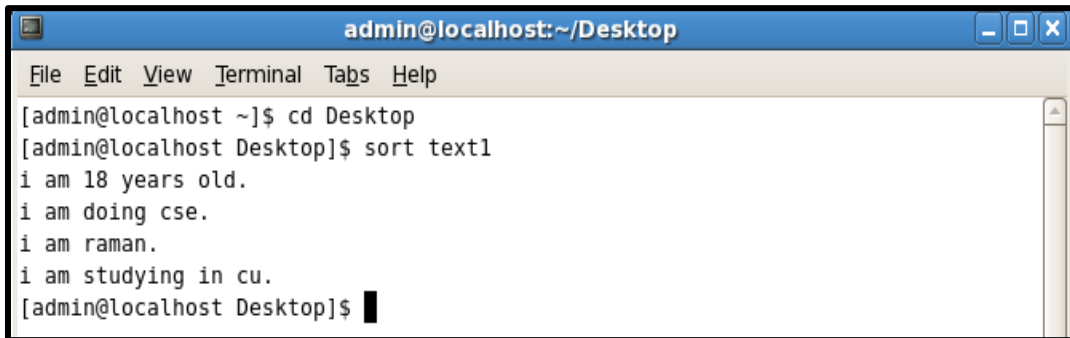
A terminal window titled 'admin@localhost:~/Desktop' with a menu bar (File, Edit, View, Terminal, Tabs, Help). The command '[admin@localhost Desktop]\$ tac text1 text2' has been executed. The output shows the contents of the files in reverse order: 'i am doing cse.', 'i am studying in cu.', 'i am 18 years old.', 'i am raman.', 'i am doing IT.', 'i am studying in pu.', 'i am 16 years old.', and 'i am khushi.'. The prompt '[admin@localhost Desktop]\$' is visible at the bottom.

```
admin@localhost:~/Desktop  
File Edit View Terminal Tabs Help  
[admin@localhost Desktop]$ tac text1 text2  
i am doing cse.  
i am studying in cu.  
i am 18 years old.  
i am raman.  
i am doing IT.  
i am studying in pu.  
i am 16 years old.  
i am khushi.  
[admin@localhost Desktop]$
```

3. **Sort:** **sort** is a simple and very useful command which will rearrange the lines in a text file so that they are sorted, numerically and alphabetically. By default, the rules for sorting are:
- lines starting with a number will appear before lines starting with a letter;
 - lines starting with a letter that appears earlier in the alphabet will appear before lines starting with a letter that appears later in the alphabet;
 - lines starting with a lowercase letter will appear before lines starting with the same letter in uppercase.

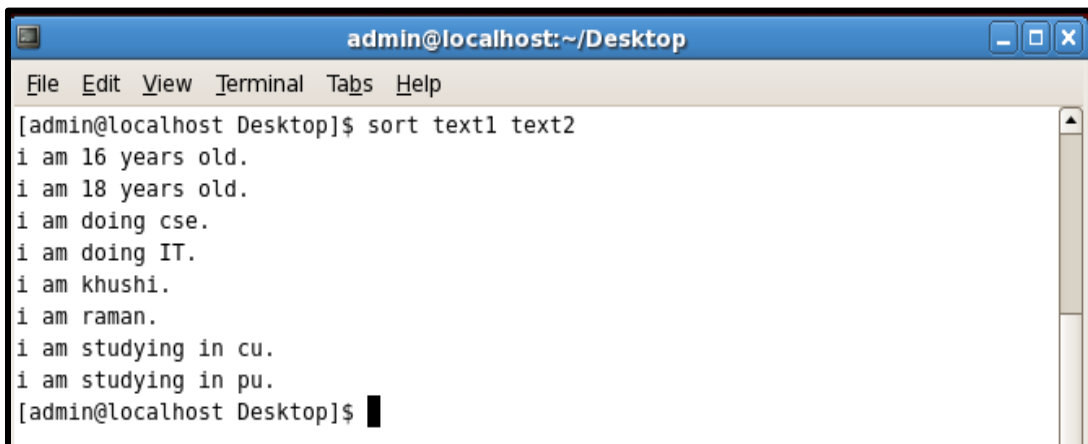
Syntax: **sort** [OPTION]... [FILE]...

sort filename: This command sort the contents of the given file and display it on the monitor screen.



```
admin@localhost:~/Desktop
File Edit View Terminal Tabs Help
[admin@localhost ~]$ cd Desktop
[admin@localhost Desktop]$ sort text1
i am 18 years old.
i am doing cse.
i am raman.
i am studying in cu.
[admin@localhost Desktop]$
```

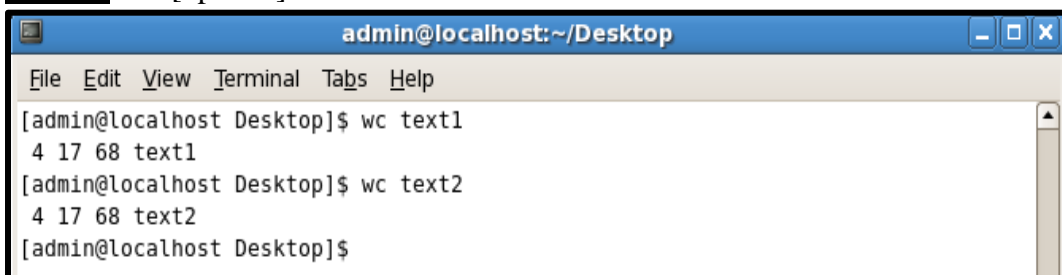
sort file1 file2: This command display the contents on the screen after sorting the contents of both the files.



```
admin@localhost:~/Desktop
File Edit View Terminal Tabs Help
[admin@localhost Desktop]$ sort text1 text2
i am 16 years old.
i am 18 years old.
i am doing cse.
i am doing IT.
i am khushi.
i am raman.
i am studying in cu.
i am studying in pu.
[admin@localhost Desktop]$
```

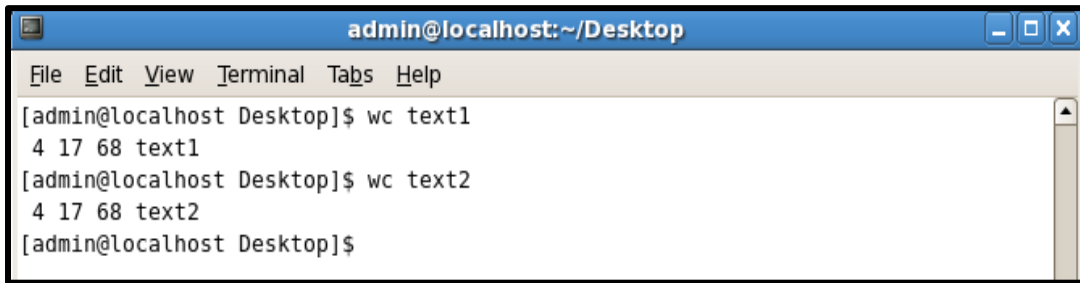
4. **wc:** The **wc (word count)** command in Unix/Linux operating systems is used to find out number of **newline count, word count, byte and characters** count in a files specified by the file arguments.

Syntax: wc [options] filenames



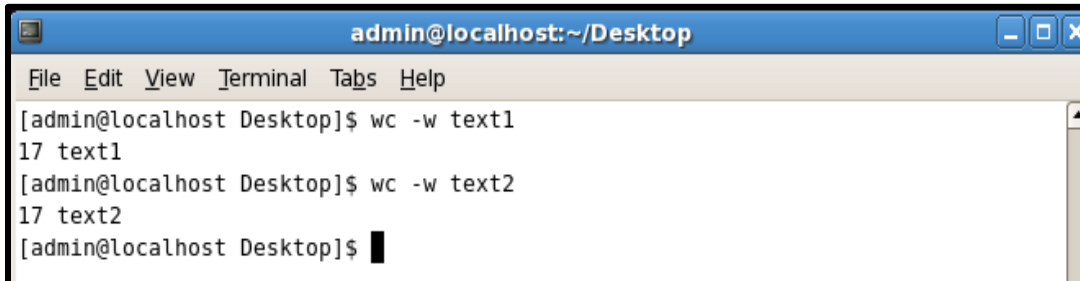
```
admin@localhost:~/Desktop
File Edit View Terminal Tabs Help
[admin@localhost Desktop]$ wc text1
 4 17 68 text1
[admin@localhost Desktop]$ wc text2
 4 17 68 text2
[admin@localhost Desktop]$
```

wc -l filename: Prints the number of lines in a file.



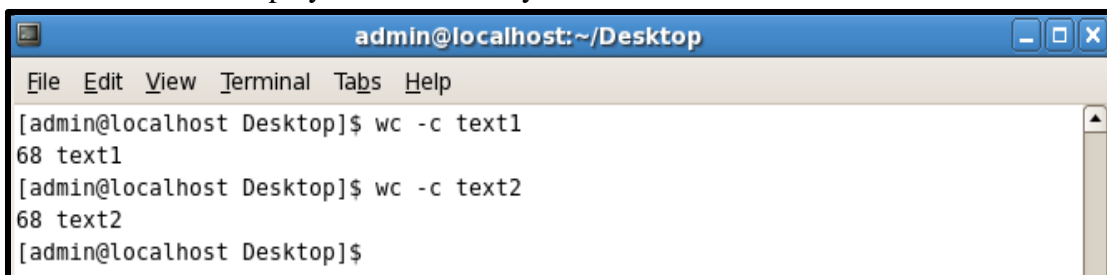
```
admin@localhost:~/Desktop
File Edit View Terminal Tabs Help
[admin@localhost Desktop]$ wc text1
 4 17 68 text1
[admin@localhost Desktop]$ wc text2
 4 17 68 text2
[admin@localhost Desktop]$
```

wc -w filename: Prints the number of words in a file.



```
admin@localhost:~/Desktop
File Edit View Terminal Tabs Help
[admin@localhost Desktop]$ wc -w text1
17 text1
[admin@localhost Desktop]$ wc -w text2
17 text2
[admin@localhost Desktop]$
```

wc -c filename: Displays the count of bytes in a file.

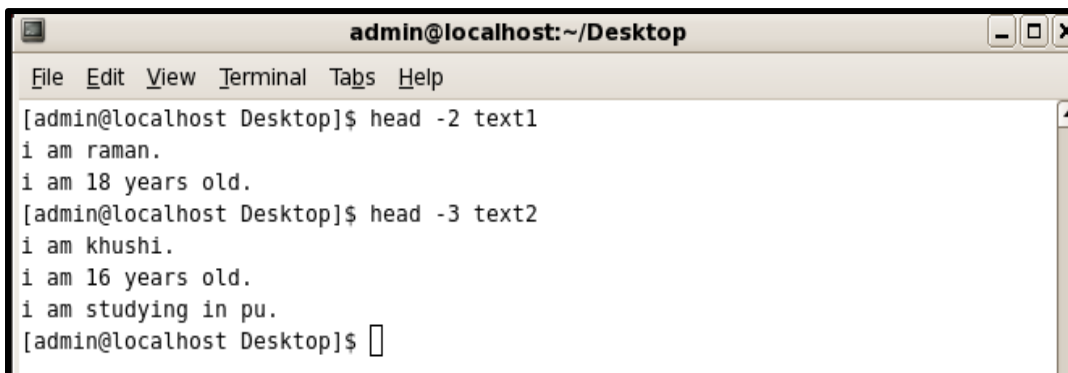


```
admin@localhost:~/Desktop
File Edit View Terminal Tabs Help
[admin@localhost Desktop]$ wc -c text1
68 text1
[admin@localhost Desktop]$ wc -c text2
68 text2
[admin@localhost Desktop]$
```

5. **Head:** Print the first 10 lines of each FILE to standard output. With more than one FILE, precede each with a header giving the file name.

Syntax: head [OPTION]... [FILE]...

head -n filename: Print the first N lines instead of the first 10.

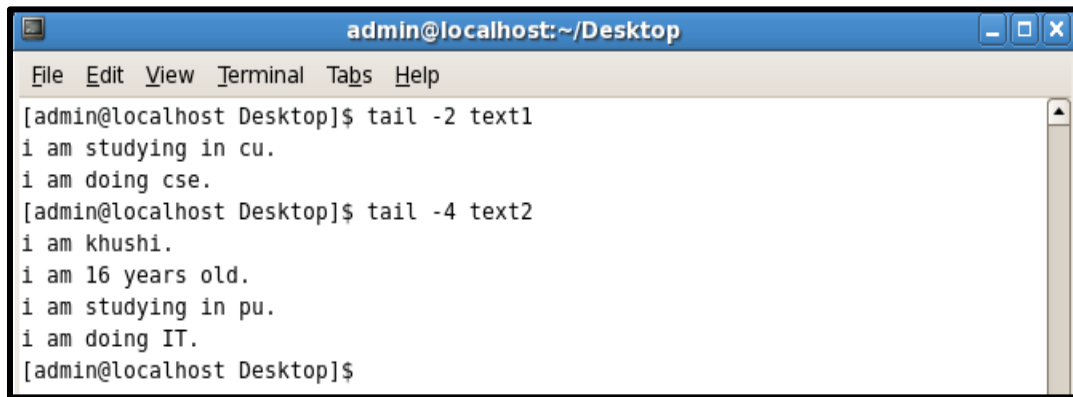


```
admin@localhost:~/Desktop
File Edit View Terminal Tabs Help
[admin@localhost Desktop]$ head -2 text1
i am raman.
i am 18 years old.
[admin@localhost Desktop]$ head -3 text2
i am khushi.
i am 16 years old.
i am studying in pu.
[admin@localhost Desktop]$
```

6. **Tail:** Print the last 10 lines of each FILE to standard output. With more than one FILE, precede each with a header giving the file name.

Syntax: tail [OPTION]... [FILE]...

tail -n filename: Output the last N lines, instead of the last 10.

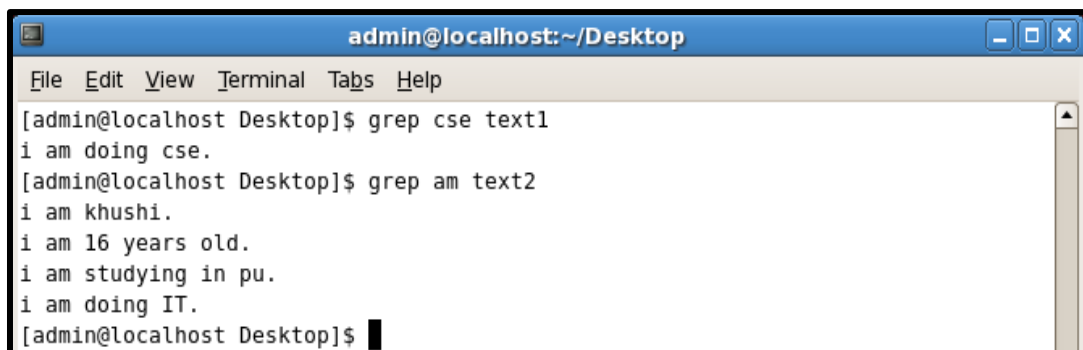


```
admin@localhost:~/Desktop
File Edit View Terminal Tabs Help
[admin@localhost Desktop]$ tail -2 text1
i am studying in cu.
i am doing cse.
[admin@localhost Desktop]$ tail -4 text2
i am khushi.
i am 16 years old.
i am studying in pu.
i am doing IT.
[admin@localhost Desktop]$
```

7. **Grep:** The grep command is used to search text or searches the given file for lines containing a match to the given strings or words. By default, grep displays the matching lines. Use grep to search for lines of text that match one or many regular expressions, and outputs only the matching lines.

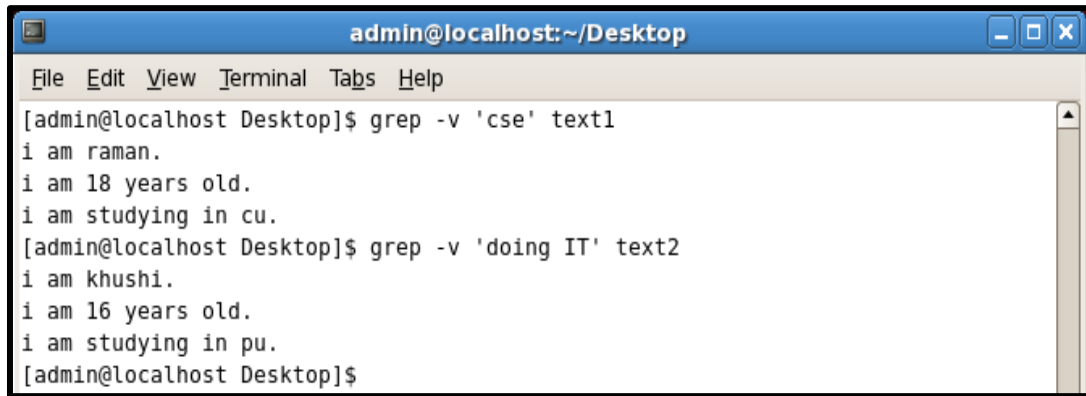
Syntax: grep [options] PATTERN [FILE...]

grep pattern filename: This command searches the named input FILES (or standard input if no files are named, or the file name -is given) for lines containing a match to the given PATTERN. By default, grep prints the matching lines.



```
admin@localhost:~/Desktop
File Edit View Terminal Tabs Help
[admin@localhost Desktop]$ grep cse text1
i am doing cse.
[admin@localhost Desktop]$ grep am text2
i am khushi.
i am 16 years old.
i am studying in pu.
i am doing IT.
[admin@localhost Desktop]$
```

grep -v filename: -v option is used to print inverts the match; that is, it matches only those lines that do not contain the given word.



A terminal window titled "admin@localhost:~/Desktop" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The terminal shows two grep commands and their outputs. The first command is `grep -v 'cse' text1`, which outputs three lines: `i am raman.`, `i am 18 years old.`, and `i am studying in cu.`. The second command is `grep -v 'doing IT' text2`, which outputs three lines: `i am khushi.`, `i am 16 years old.`, and `i am studying in pu.`. The prompt `[admin@localhost Desktop]$` is shown at the end of each command and at the bottom of the window.

```
admin@localhost:~/Desktop
File Edit View Terminal Tabs Help
[admin@localhost Desktop]$ grep -v 'cse' text1
i am raman.
i am 18 years old.
i am studying in cu.
[admin@localhost Desktop]$ grep -v 'doing IT' text2
i am khushi.
i am 16 years old.
i am studying in pu.
[admin@localhost Desktop]$
```

EXPERIMENT-3

Exp 3.1

AIM: Programs using the I/O system calls of Linux operating system (open, read, write etc).

```
#include<stdio.h>

#include<sys/stat.h>

#include<time.h>
main(intag,char*arg[])
{
charbuf[100];
struct stat s;
int fd1,fd2,n;
fd1=open(arg[1],0);
fd2=creat(arg[2],0777);
stat(arg[2],&s);
if(fd2==
-
1)
printf("ERROR IN CREATION");
while((n=read(fd1,buf,sizeof(buf)))>0)
{
if(write(fd2,buf,n)!=n)
{
close(fd1);
close(fd2);
}
}
printf("
\
t
\
n UID FOR FILE.....>%d
\
n FILE
ACCESS TIME.....>%s
\
n FILE
MODIFIED TIME.....>%s
\
n FILE I
-
NODE NUMBER.....>%d
\
```

```
n
PERMISSION FOR
FILE.....>%o
\
n
\
n",s.st_uid,ctime
(&s.st_atime),ctime(&s.st_mtime),s.st_mode);
close(fd1);
close(fd2);
}
```

OUTPUT

```
[root@localhost ~]# cc iosys.c
[roo
t@localhost ~]# ./a.out
UID FOR FILE.....>0
FILE ACCESS TIME.....>Thu Apr 8 01:23:54 2011
FILE MODIFIED TIME.....>Thu Apr 8 01:23:54 2011
FILE I
-
NODE NUMBER.....>33261
PERMISSION FOR FILE.....>1001101014
```

EXPERIMENT-3

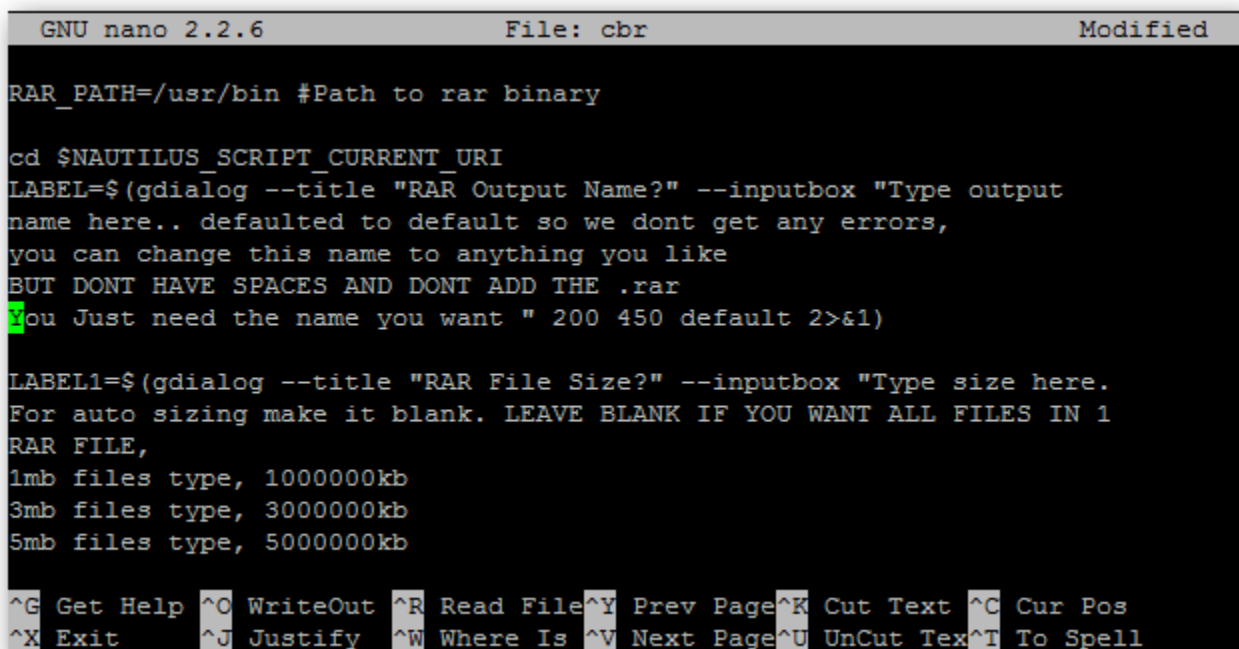
Exp 3.2

AIM: Study of basics of shell programming.

The term “shell scripting” gets mentioned often in Linux forums, but many users aren’t familiar with it. Learning this easy and powerful programming method can help you save time, learn the command-line better, and banish tedious file management tasks.

Being a Linux user means you play around with the command-line. Like it or not, there are just some things that are done much more easily via this interface than by pointing and clicking. The more you use and learn the command-line, the more you see its potential. Well, the command-line itself is a program: the shell. Most Linux distros today use Bash, and this is what you’re really entering commands into.

Now, some of you who used Windows before using Linux may remember batch files. These were little text files that you could fill with commands to execute and Windows would run them in turn. It was a clever and neat way to get some things done, like run games in your high school computer lab when you couldn’t open system folders or create shortcuts. Batch files in Windows, while useful, are a cheap imitation of shell scripts.



```

GNU nano 2.2.6                               File: cbr                               Modified

RAR_PATH=/usr/bin #Path to rar binary

cd $NAUTILUS_SCRIPT_CURRENT_URI
LABEL=$(gdialog --title "RAR Output Name?" --inputbox "Type output
name here.. defaulted to default so we dont get any errors,
you can change this name to anything you like
BUT DONT HAVE SPACES AND DONT ADD THE .rar
You Just need the name you want " 200 450 default 2>&1)

LABEL1=$(gdialog --title "RAR File Size?" --inputbox "Type size here.
For auto sizing make it blank. LEAVE BLANK IF YOU WANT ALL FILES IN 1
RAR FILE,
1mb files type, 1000000kb
3mb files type, 3000000kb
5mb files type, 5000000kb

^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify    ^W Where Is ^V Next Page ^U UnCut Tex ^T To Spell

```

Shell scripts allow us to program commands in chains and have the system execute them as a scripted event, just like batch files. They also allow for far more useful functions, such as command

substitution. You can invoke a command, like `date`, and use its output as part of a file-naming scheme. You can automate backups and each copied file can have the current date appended to the end of its name. Scripts aren't just invocations of commands, either. They're programs in their own right. Scripting allows you to use programming functions – such as 'for' loops, if/then/else statements, and so forth – directly within your operating system's interface. And, you don't have to learn another language because you're using what you already know: the command-line.

That's really the power of scripting, I think. You get to program with commands you already know, while learning staples of most major programming languages. Need to do something repetitive and tedious? Script it! Need a shortcut for a really convoluted command? Script it! Want to build a really easy to use command-line interface for something? Script it!

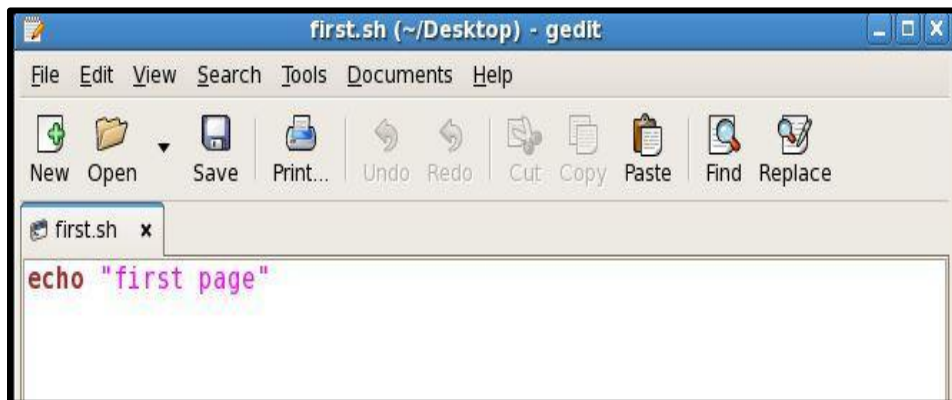
EXPERIMENT-4

Exp 4.1

AIM: Write a program to show the use of echo.

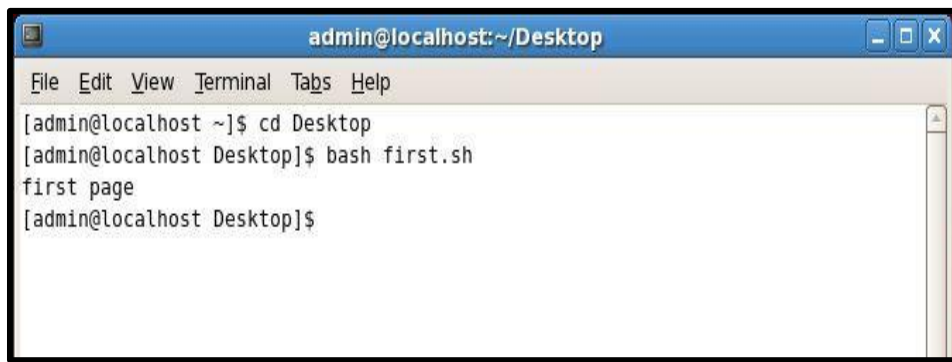
Program to print “first page” on the screen.

Source code:



The screenshot shows a gedit editor window titled "first.sh (~/Desktop) - gedit". The menu bar includes File, Edit, View, Search, Tools, Documents, and Help. The toolbar contains icons for New, Open, Save, Print..., Undo, Redo, Cut, Copy, Paste, Find, and Replace. A single tab labeled "first.sh" is open. The editor area contains the text `echo "first page"` in a syntax-highlighted font.

Output:



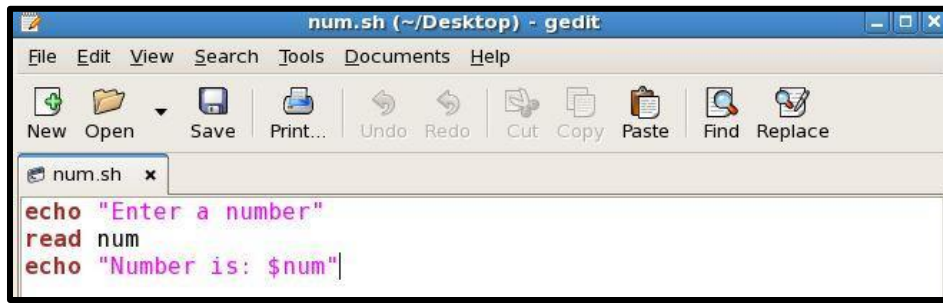
The screenshot shows a terminal window titled "admin@localhost:~/Desktop". The menu bar includes File, Edit, View, Terminal, Tabs, and Help. The terminal displays the following commands and output:
[admin@localhost ~]\$ cd Desktop
[admin@localhost Desktop]\$ bash first.sh
first page
[admin@localhost Desktop]\$

EXPERIMENT-4

Exp 4.2

AIM: Write a program to read the keywords in shell programming.

Source code:



```
num.sh (~/Desktop) - gedit
File Edit View Search Tools Documents Help
New Open Save Print... Undo Redo Cut Copy Paste Find Replace
num.sh x
echo "Enter a number"
read num
echo "Number is: $num"
```

Output:



```
admin@localhost:~/Desktop
File Edit View Terminal Tabs Help
[admin@localhost ~]$ cd Desktop
[admin@localhost Desktop]$ bash num.sh
Enter a number
12
Number is: 12
[admin@localhost Desktop]$
```

Source code:



```
name.sh (~/Desktop) - gedit
File Edit View Search Tools Documents Help
New Open Save Print... Undo Redo Cut Copy Paste Find Replace
name.sh x
echo "Enter your name"
read name
echo "name is: $name"
```

Output:

A terminal window titled "admin@localhost:~/Desktop" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The terminal shows the following sequence of commands and output:

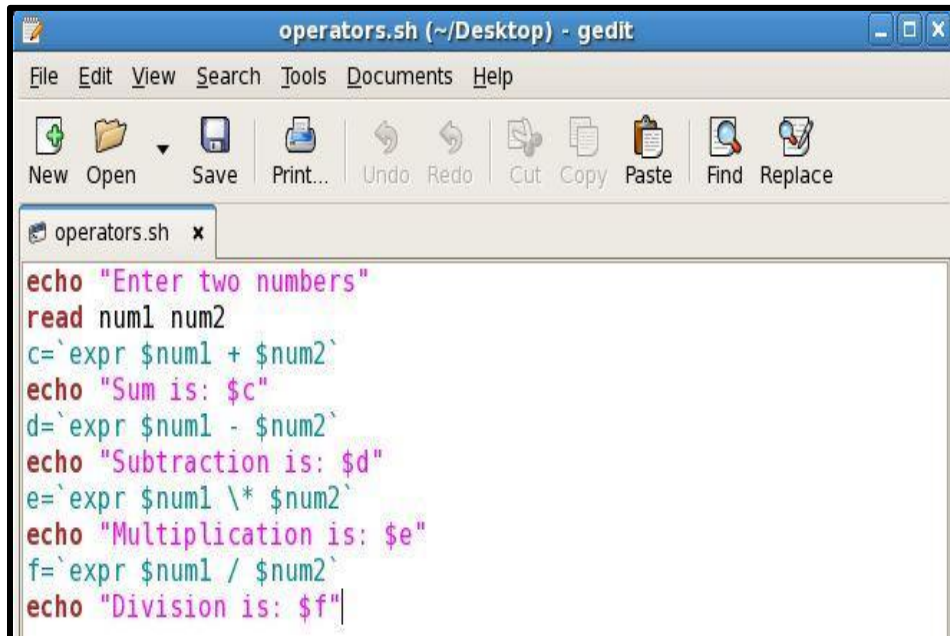
```
[admin@localhost ~]$ cd Desktop
[admin@localhost Desktop]$ bash name.sh
Enter your name
Raman
name is: Raman
[admin@localhost Desktop]$
```

EXPERIMENT-5

Exp 5.1

AIM: Write a program using arithmetic operators in shell programming.

Source code:



```
operators.sh (~/Desktop) - gedit
File Edit View Search Tools Documents Help
New Open Save Print... Undo Redo Cut Copy Paste Find Replace
operators.sh x
echo "Enter two numbers"
read num1 num2
c=`expr $num1 + $num2`
echo "Sum is: $c"
d=`expr $num1 - $num2`
echo "Subtraction is: $d"
e=`expr $num1 \* $num2`
echo "Multiplication is: $e"
f=`expr $num1 / $num2`
echo "Division is: $f"
```

Output:

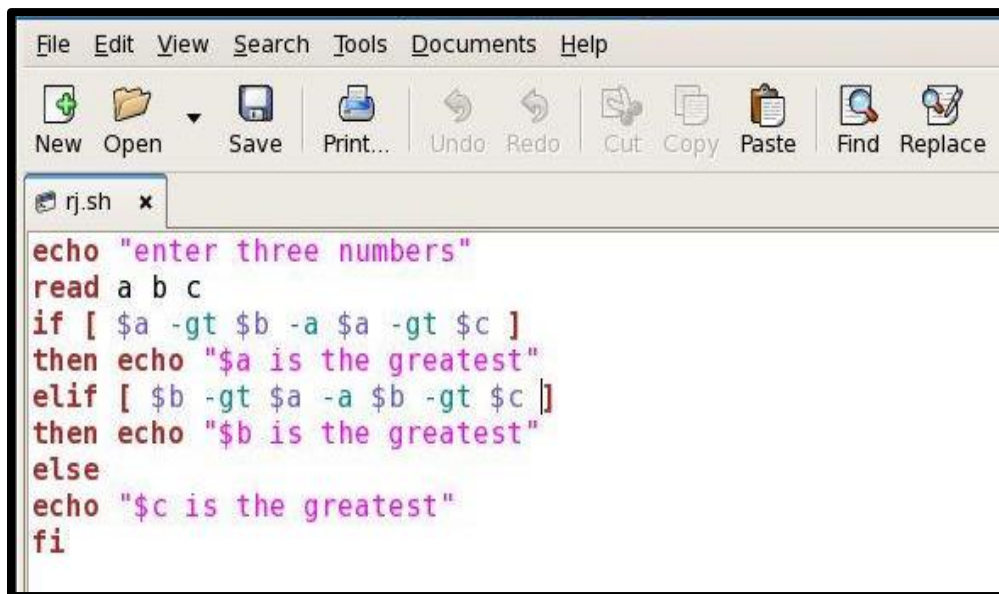


```
admin@localhost:~/Desktop
File Edit View Terminal Tabs Help
[admin@localhost ~]$ cd Desktop
[admin@localhost Desktop]$ bash operators.sh
Enter two numbers
7 5
Sum is: 12
Subtraction is: 2
Multiplication is: 35
Division is: 1
[admin@localhost Desktop]$
```

EXPERIMENT-5

Exp 5.2

AIM: Write a program using relational operators in shell programming.



```
File Edit View Search Tools Documents Help
New Open Save Print... Undo Redo Cut Copy Paste Find Replace

rj.sh x
echo "enter three numbers"
read a b c
if [ $a -gt $b -a $a -gt $c ]
then echo "$a is the greatest"
elif [ $b -gt $a -a $b -gt $c ]
then echo "$b is the greatest"
else
echo "$c is the greatest"
fi
```

Output:

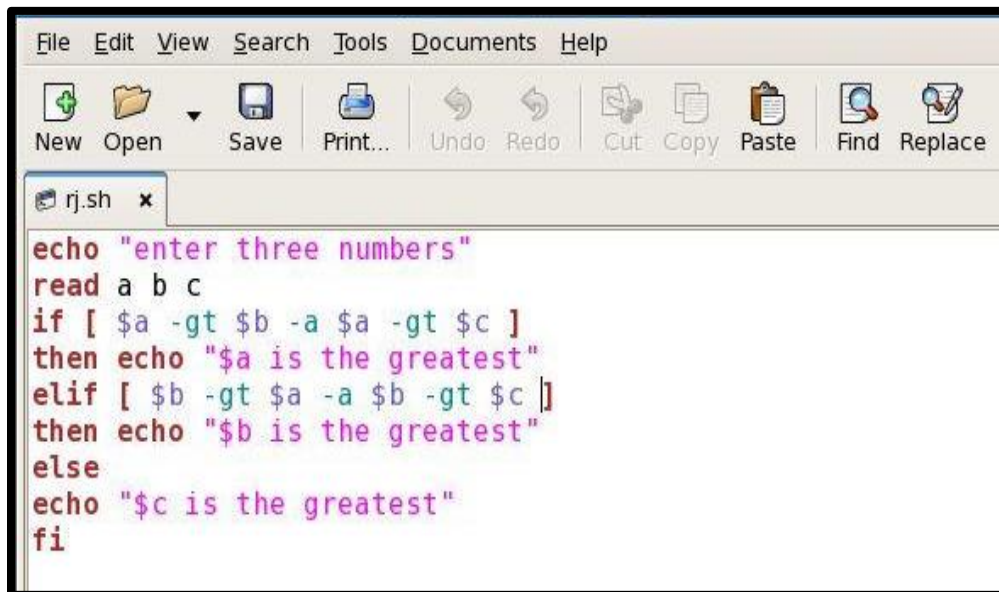


```
admin@localhost:~/Desktop
File Edit View Terminal Tabs Help
[admin@localhost ~]$ cd Desktop
[admin@localhost Desktop]$ bash rj.sh
enter three numbers
15 34 2
34 is the greatest
[admin@localhost Desktop]$
```

EXPERIMENT-6

Exp 6.1

AIM: Write a program using Boolean operators in shell programming.



```
File Edit View Search Tools Documents Help
New Open Save Print... Undo Redo Cut Copy Paste Find Replace
rj.sh x
echo "enter three numbers"
read a b c
if [ $a -gt $b -a $a -gt $c ]
then echo "$a is the greatest"
elif [ $b -gt $a -a $b -gt $c ]
then echo "$b is the greatest"
else
echo "$c is the greatest"
fi
```

Output:



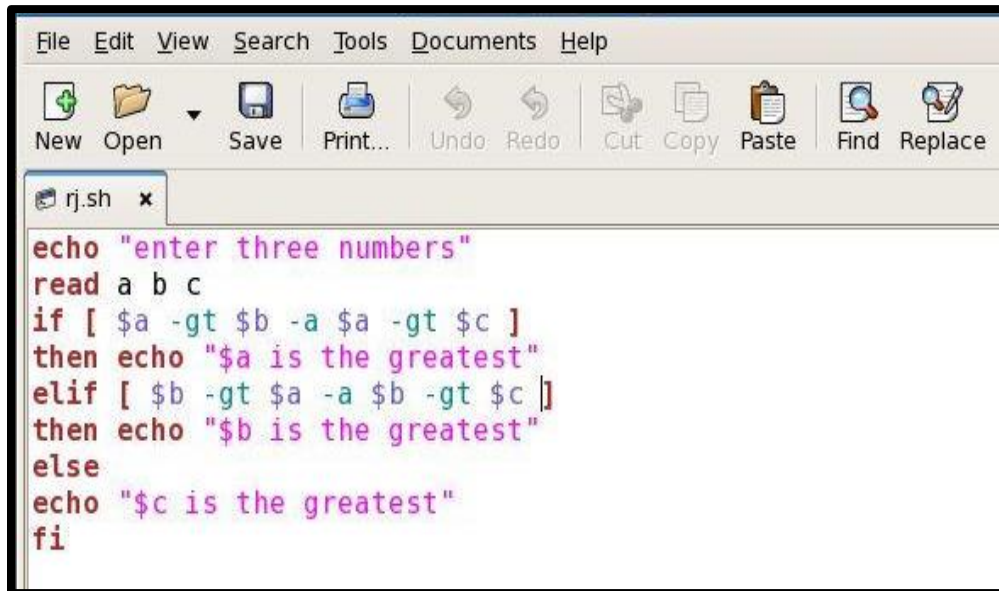
```
admin@localhost:~/Desktop
File Edit View Terminal Tabs Help
[admin@localhost ~]$ cd Desktop
[admin@localhost Desktop]$ bash rj.sh
enter three numbers
15 34 2
34 is the greatest
[admin@localhost Desktop]$
```

EXPERIMENT-6

Exp 6.2

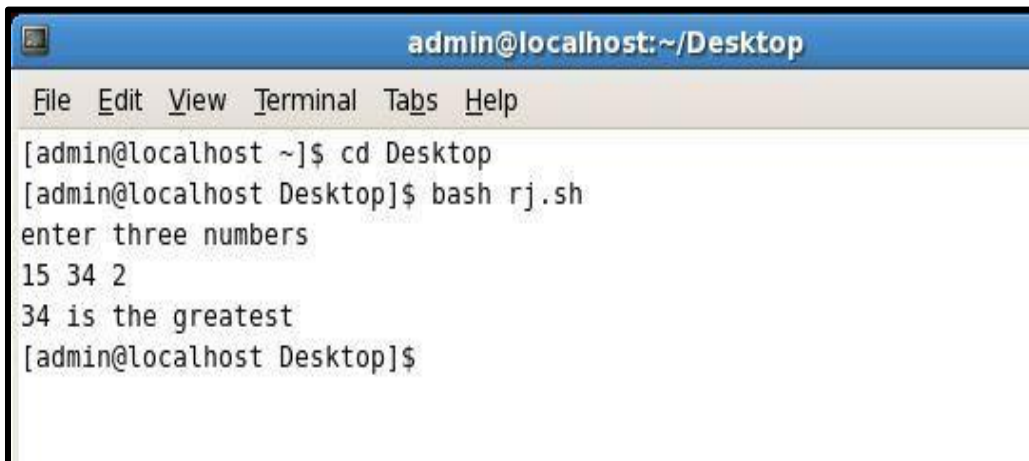
AIM: Write programs using control structures in shell programming.

Program using if-else statements



```
File Edit View Search Tools Documents Help
New Open Save Print... Undo Redo Cut Copy Paste Find Replace
rj.sh x
echo "enter three numbers"
read a b c
if [ $a -gt $b -a $a -gt $c ]
then echo "$a is the greatest"
elif [ $b -gt $a -a $b -gt $c ]
then echo "$b is the greatest"
else
echo "$c is the greatest"
fi
```

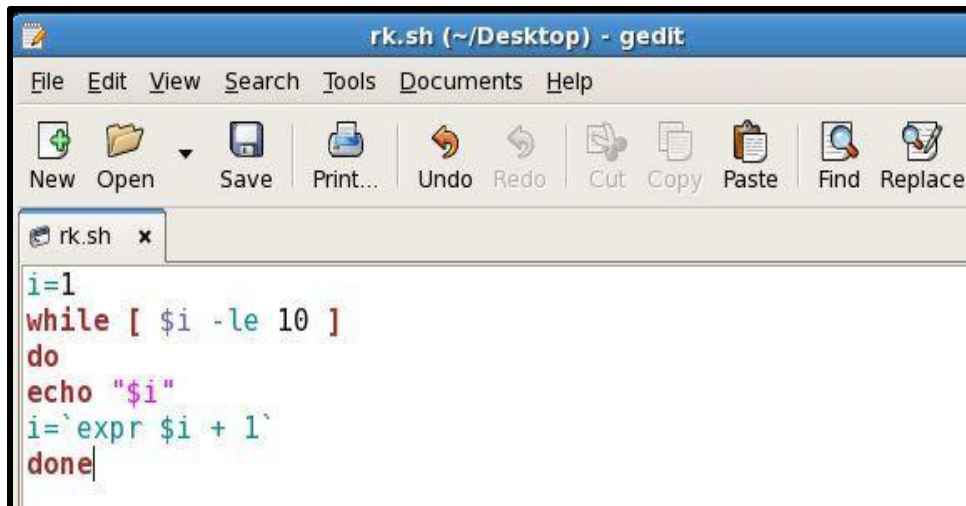
Output:



```
admin@localhost:~/Desktop
File Edit View Terminal Tabs Help
[admin@localhost ~]$ cd Desktop
[admin@localhost Desktop]$ bash rj.sh
enter three numbers
15 34 2
34 is the greatest
[admin@localhost Desktop]$
```

Program to display first ten natural numbers using while loop.

Source code:



The screenshot shows a gedit editor window titled "rk.sh (~/Desktop) - gedit". The menu bar includes File, Edit, View, Search, Tools, Documents, and Help. The toolbar contains icons for New, Open, Save, Print..., Undo, Redo, Cut, Copy, Paste, Find, and Replace. The script content is as follows:

```
i=1
while [ $i -le 10 ]
do
echo "$i"
i=`expr $i + 1`
done
```

Output:

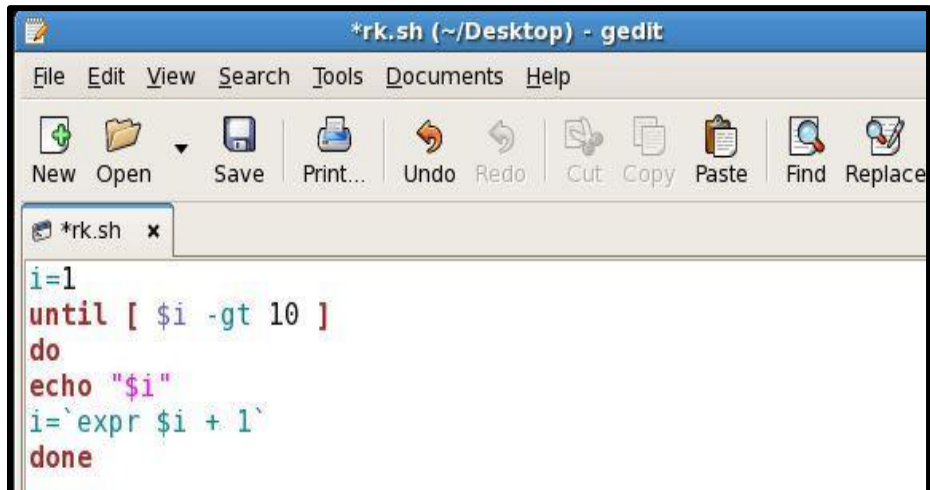


The screenshot shows a terminal window titled "admin@localhost:~/Desktop". The menu bar includes File, Edit, View, Terminal, Tabs, and Help. The terminal shows the following commands and output:

```
[admin@localhost ~]$ cd Desktop
[admin@localhost Desktop]$ bash rk.sh
1
2
3
4
5
6
7
8
9
10
[admin@localhost Desktop]$
```

Program to display first ten natural numbers using until loop.

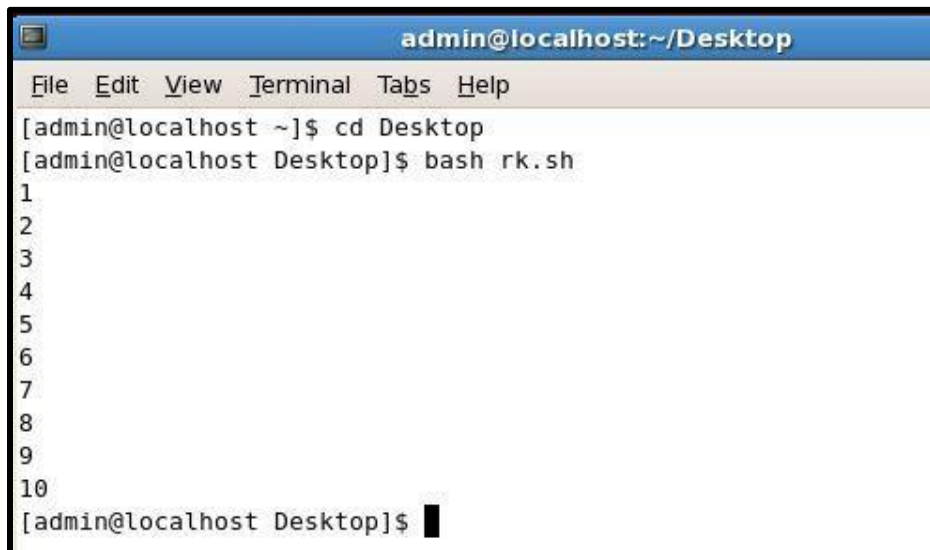
Source code:



The screenshot shows a gedit editor window titled '*rk.sh (~/Desktop) - gedit'. The menu bar includes File, Edit, View, Search, Tools, Documents, and Help. The toolbar contains icons for New, Open, Save, Print..., Undo, Redo, Cut, Copy, Paste, Find, and Replace. The script content is as follows:

```
i=1
until [ $i -gt 10 ]
do
echo "$i"
i=`expr $i + 1`
done
```

Output:



The screenshot shows a terminal window titled 'admin@localhost:~/Desktop'. The menu bar includes File, Edit, View, Terminal, Tabs, and Help. The terminal output is as follows:

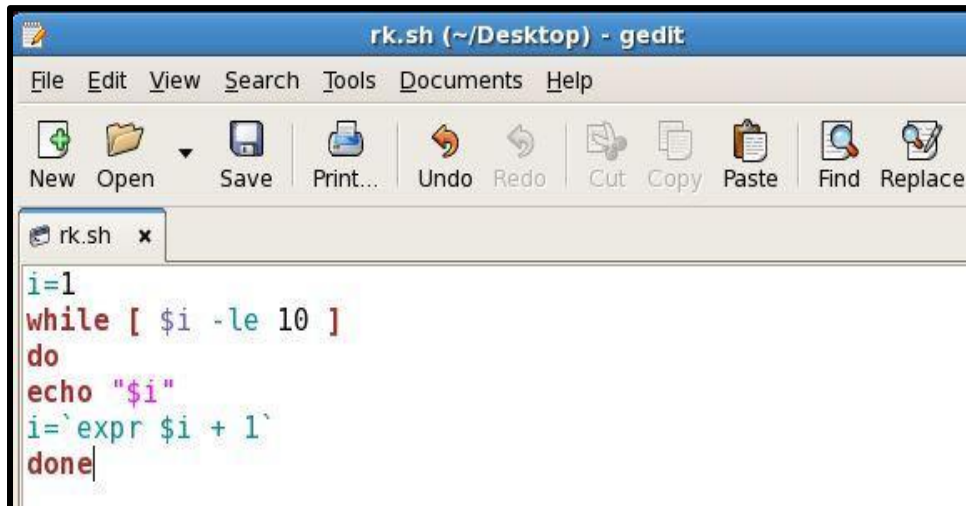
```
[admin@localhost ~]$ cd Desktop
[admin@localhost Desktop]$ bash rk.sh
1
2
3
4
5
6
7
8
9
10
[admin@localhost Desktop]$
```


EXPERIMENT-7

AIM: Write a program to demonstrate the difference between while and until statement.

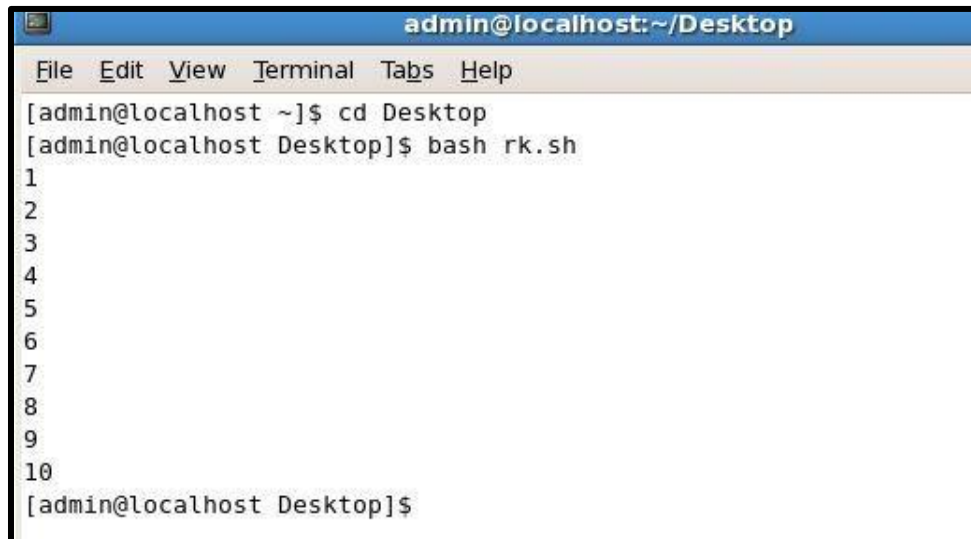
1. Program to display first ten natural numbers using while loop.

Source code:



```
rk.sh (~/Desktop) - gedit
File Edit View Search Tools Documents Help
New Open Save Print... Undo Redo Cut Copy Paste Find Replace
rk.sh x
i=1
while [ $i -le 10 ]
do
echo "$i"
i=`expr $i + 1`
done
```

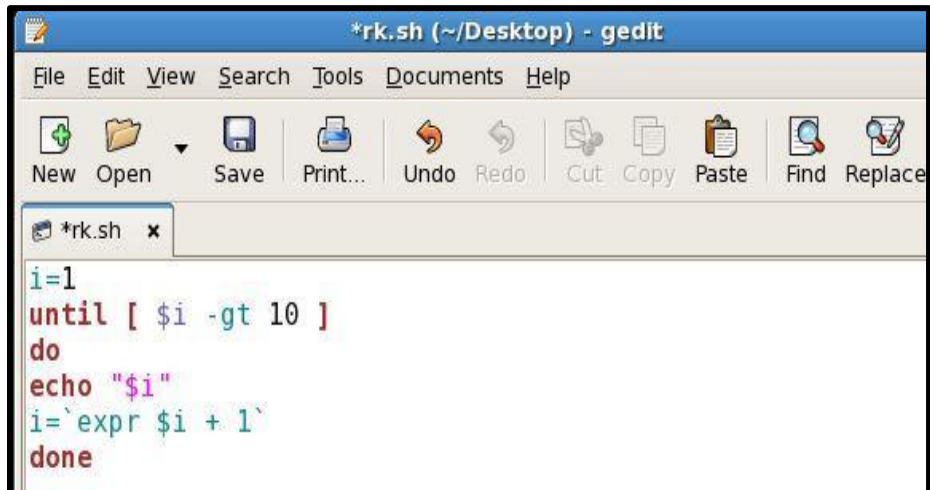
Output:



```
admin@localhost:~/Desktop
File Edit View Terminal Tabs Help
[admin@localhost ~]$ cd Desktop
[admin@localhost Desktop]$ bash rk.sh
1
2
3
4
5
6
7
8
9
10
[admin@localhost Desktop]$
```

2. Program to display first ten natural numbers using until loop.

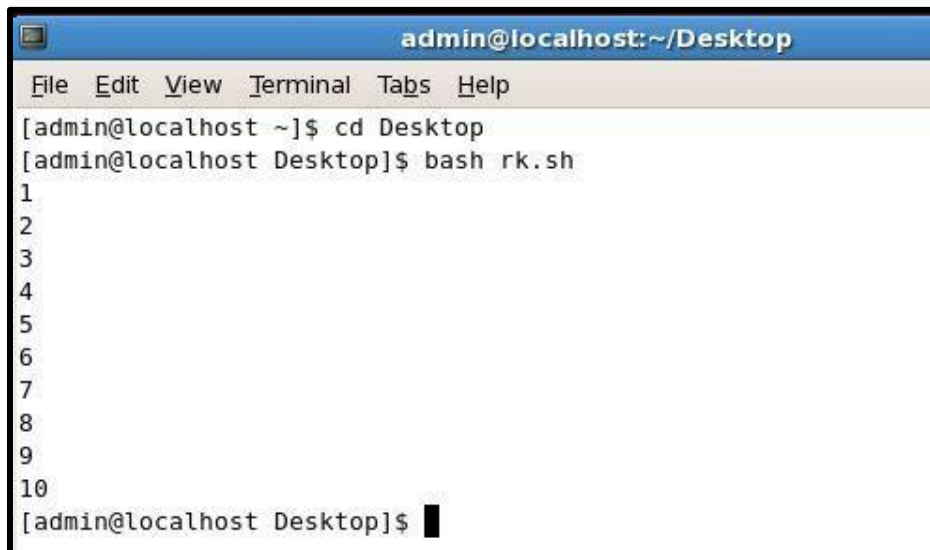
Source code:



The screenshot shows a gedit editor window titled '*rk.sh (~/Desktop) - gedit'. The menu bar includes File, Edit, View, Search, Tools, Documents, and Help. The toolbar contains icons for New, Open, Save, Print..., Undo, Redo, Cut, Copy, Paste, Find, and Replace. The script content is as follows:

```
i=1
until [ $i -gt 10 ]
do
echo "$i"
i=`expr $i + 1`
done
```

Output:



The screenshot shows a terminal window titled 'admin@localhost:~/Desktop'. The menu bar includes File, Edit, View, Terminal, Tabs, and Help. The terminal output is as follows:

```
[admin@localhost ~]$ cd Desktop
[admin@localhost Desktop]$ bash rk.sh
1
2
3
4
5
6
7
8
9
10
[admin@localhost Desktop]$
```

EXPERIMENT-8

Exp 8.1

AIM: Simulation of First come first serve CPU scheduling algorithm.

Source code:

```
#include<iostream.h>

#include<conio.h>

void main()

{

    clrscr();

    int n,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;

    cout<<"Enter total number of processes:";

    cin>>n;


    cout<<"\nEnter Process Burst Time\n";

    for(i=0;i<n;i++)

    {

        cout<<"P["<<i+1<<"]:";

        cin>>bt[i];

    }


    wt[0]=0;  //waiting time for first process is 0


    //calculating waiting time

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

    {

        wt[i]=0;
```

CHANDIGARH UNIVERSITY, GHARUAN (MOHALI)

```
for(j=0;j<i;j++)
    wt[i]+=bt[j];
}

cout<<"\nProcess\tBurst Time\tWaiting Time\tTurnaround Time";

//calculating turnaround time
for(i=0;i<n;i++)
{
    tat[i]=bt[i]+wt[i];
    avwt+=wt[i];
    avtat+=tat[i];
    cout<<"\nP["<<i+1<<"]"<<"\t\t"<<bt[i]<<"\t\t"<<wt[i]<<"\t\t"<<tat[i];
}

avwt/=i;
avtat/=i;

cout<<"\nAverage Waiting Time:"<<avwt;
cout<<"\nAverage Turn around Time:"<<avtat;

getch();
}
```

Output:

Enter total number of processes:3

Enter Process Burst Time

P[1]:4

P[2]:1

P[3]:2

Process	Burst Time	Waiting Time	Turnaround Time
P[1]	4	0	4
P[2]	1	4	5
P[3]	2	5	7

Average Waiting Time:3

Average Turn around Time:5

EXPERIMENT-8

Exp 8.2

AIM: Simulation of Shortest job first CPU scheduling algorithm.

Source code:

```
#include<stdio.h>

void main()

{

    int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;

    float avg_wt,avg_tat;

    clrscr();

    printf("Enter number of process: ");

    scanf("%d",&n);

    printf("\nEnter Burst Time:\n");

    for(i=0;i<n;i++)

    {

        printf("p%d:",i+1);

        scanf("%d",&bt[i]);

        p[i]=i+1;        //contains process number

    }

    //sorting burst time in ascending order using selection sort

    for(i=0;i<n;i++)

    {

        pos=i;

        for(j=i+1;j<n;j++)

        {
```

CHANDIGARH UNIVERSITY, GHARUAN (MOHALI)

```
        if(bt[j]<bt[pos])
            pos=j;
    }

    temp=bt[i];
    bt[i]=bt[pos];
    bt[pos]=temp;

    temp=p[i];
    p[i]=p[pos];
    p[pos]=temp;
}

wt[0]=0;          //waiting time for first process will be zero

//calculate waiting time
for(i=1;i<n;i++)
{
    wt[i]=0;
    for(j=0;j<i;j++)
        wt[i]+=bt[j];

    total+=wt[i];
}
```

CHANDIGARH UNIVERSITY, GHARUAN (MOHALI)

```
avg_wt=(float)total/n;    //average waiting time

total=0;

printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");

for(i=0;i<n;i++)

{

    tat[i]=bt[i]+wt[i];    //calculate turnaround time

    total+=tat[i];

    printf("\np%d\t\t %d\t\t %d\t\t%d",p[i],bt[i],wt[i],tat[i]);

}

avg_tat=(float)total/n;    //average turnaround time

printf("\n\nAverage Waiting Time=%f",avg_wt);

printf("\n\nAverage Turnaround Time=%f\n",avg_tat);

getch();

}
```

Output:

Enter number of process: 3

Enter Burst Time:

p1:5

p2:2

p3:1

Process	Burst Time	Waiting Time	Turnaround Time
p3	1	0	1
p2	2	1	3
p1	5	3	8

Average Waiting Time=1.333333

Average Turnaround Time=4.000000

EXPERIMENT-9

Exp 9.1

AIM: Simulation of Round Robin CPU scheduling algorithm.

RRS(Round robin Scheduling):

Source code:

```
#include<iostream.h>

#include<conio.h>

struct process
{
    int no;

    int at,et,wt,tt;

    int tet;

    int t;
};

void main()
{
    clrscr();

    process p[99];

    int i,j,k;

    cout<<"\n Enter No of Processes:";

    int np;

    cin>>np;

    for (i=0;i<np;i++)
```

```
{
    cout<<"\n Enter Execution time of process "<<i+1<<":";
    cin>>p[i].et;
    p[i].tet=p[i].et;
    p[i].at=p[i].t=p[i].tt=p[i].wt=0;
    p[i].no=i+1;
}

cout<<"\n Enter Time Quantum:";

int q;

cin>>q;

cout<<"\n Entered Data";

cout<<"\nProcess ExecutionTime";

for(i=0;i<np;i++)
{
    cout<<"\n "<<p[i].no<<"\t"<<p[i].et;
}

int totaltime=0;

for(i=0;i<np;i++)
{
    totaltime+=p[i].et;
}

i=0;

k=0;

int rrg[99];

for(j=0;j<totaltime;j++)
```

```
{
    if((k==0)&&(p[i].et!=0))
    {
        p[i].wt=j;
        if((p[i].t!=0))
        {
            p[i].wt-=q*p[i].t;
        }
    }
    if((p[i].et!=0)&&(k!=q))
    {
        rrg[j]=p[i].no;
        p[i].et-=1;
        k++;
    }
    else
    {
        if((k==q)&&(p[i].et!=0))
        {
            p[i].t+=1;
        }
        i=i+1;
        if(i==np)
        {
            i=0;
        }
    }
}
```

```
    }  
    k=0;  
    j=j-1;  
    }  
}  
int twt=0;  
int ttt=0;  
cout<<"\n \t\tResult Of Round Robin";  
cout<<"\n Processes\tExecution Time\tWaitng Time\tTurn around Time";  
for(i=0;i<np;i++)  
{  
    p[i].tt=p[i].wt+p[i].tet;  
    ttt+=p[i].tt;  
    twt+=p[i].wt;  
    cout<<"\n "<<p[i].no<<"\t"<<"\t\t"<<p[i].tet<<"\t"<<p[i].wt<<"\t\t"<<p[i].tt;  
}  
  
cout<<"\n Average Waiting Time:"<<(float)twt/np;  
cout<<"\n Average Turn Around Time:"<<(float)ttt/np;  
getch();  
}
```

Output:

Enter No of Processes:3

Enter Execution time of process 1:5

Enter Execution time of process 2:2

Enter Execution time of process 3:6

Enter Time Quantum:3

Entered Data

Process ExecutionTime

1 5

2 2

3 6

Result Of Round Robin

Processes	Execution Time	Waitng Time	Turn around Time
1	5	5	10
2	2	3	5
3	6	7	13

Average Waiting Time:5

Average Turn Around Time:9.333333

EXPERIMENT-9

Exp 9.2

AIM: Simulation of Priority based CPU scheduling algorithm.

Source code:

```
#include<iostream.h>

#include<conio.h>

void main()

{

clrscr();

int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg_wt,avg_tat;

cout<<"Enter Total Number of Process:";

cin>>n;


cout<<"\nEnter Burst Time and Priority\n";

for(i=0;i<n;i++)

{

cout<<"\nP["<<i+1<<"]\n";

cout<<"Burst Time:";

cin>>bt[i];

cout<<"Priority:";

cin>>pr[i];

p[i]=i+1;

}

for(i=0;i<n;i++)

{

pos=i;
```

CHANDIGARH UNIVERSITY, GHARUAN (MOHALI)

```
for(j=i+1;j<n;j++)  
{  
    if(pr[j]<pr[pos])  
        pos=j;  
}
```

```
temp=pr[i];  
pr[i]=pr[pos];  
pr[pos]=temp;
```

```
temp=bt[i];  
bt[i]=bt[pos];  
bt[pos]=temp;
```

```
temp=p[i];  
p[i]=p[pos];  
p[pos]=temp;  
}
```

```
wt[0]=0;  
for(i=1;i<n;i++)  
{  
    wt[i]=0;  
    for(j=0;j<i;j++)  
        wt[i]+=bt[j];
```



```
        total+=wt[i];
    }

    avg_wt=total/n;
    total=0;

    cout<<"\nProcess\t Burst Time \tWaiting Time\tTurn around Time";
    for(i=0;i<n;i++)
    {
        tat[i]=bt[i]+wt[i];    //calculate turnaround time
        total+=tat[i];
    }
    cout<<"\nP["<<p[i]<<"]\t\t "<<bt[i]<<"\t\t "<<wt[i]<<"\t\t"<<tat[i];

    }

    avg_tat=total/n;    //average turnaround time
    cout<<"\n\nAverage Waiting Time="<<avg_wt;
    cout<<"\nAverage Turnaround Time="<<avg_tat;

    getch();
}
```

Output:

Enter Total Number of Process:3

Enter Burst Time and Priority

P[1]

Burst Time:4

Priority:6

P[2]

Burst Time:2

Priority:3

P[3]

Burst Time:5

Priority:4

Process	Burst Time	Waiting Time	Turn around Time
P[2]	2	0	2
P[3]	5	2	7
P[1]	4	7	11

Average Waiting Time=3

Average Turnaround Time=6_

EXPERIMENT-10

AIM: Simulate the Bankers algorithm for deadlock avoidance and deadlock prevention.

Source code:

```
#include<iostream.h>

#include<conio.h>

#define MAX 20

class bankers

{

private:

int al[MAX][MAX],m[MAX][MAX],n[MAX][MAX],avail[MAX];

int nop,nor,k,result[MAX],pnum,work[MAX],finish[MAX];

public:

bankers();

void input();

void method();

int search(int);

void display();

};

bankers::bankers()

{

k=0;

for(int i=0;i<MAX;i++)

{

for(int j=0;j<MAX;j++)

{
```

```
al[i][j]=0;

m[i][j]=0;

n[i][j]=0;

}

avail[i]=0;

result[i]=0;

finish[i]=0;

}

}

void bankers::input()

{

int i,j;

cout<<"Enter the number of processes:";

cin>>nop;

cout<<"Enter the number of resources:";

cin>>nor;

cout<<"Enter the allocated resources for each process: "<<endl;

for(i=0;i<nop;i++)

{

cout<<"\nProcess "<<i;

for(j=0;j<nor;j++)

{

cout<<"\nResource "<<j<<":";

cin>>al[i][j];

}
```

```
}

cout<<"Enter the maximum resources that are needed for each process: "<<endl;

for(i=0;i<nop;i++)

{

cout<<"\nProcess "<<i;

for(j=0;j<nor;j++)

{

cout<<"\nResouce "<<j<<":";

cin>>m[i][j];

n[i][j]=m[i][j]-al[i][j];

}

}

cout<<"Enter the currently available resources in the system: ";

for(j=0;j<nor;j++)

{

cout<<"Resource "<<j<<":";

cin>>avail[j];

work[j]=-1;

}

for(i=0;i<nop;i++)

finish[i]=0;

}

void bankers::method()

{

int i=0,j,flag;
```

```
while(1)
{
if(finish[i]==0)
{
pnum =search(i);
if(pnum!=-1)
{
result[k++]=i;
finish[i]=1;
for(j=0;j<nor;j++)
{
avail[j]=avail[j]+al[i][j];
}
}
}
i++;
if(i==nop)
{
flag=0;
for(j=0;j<nor;j++)
if(avail[j]!=work[j])
flag=1;
for(j=0;j<nor;j++)
work[j]=avail[j];
if(flag==0)
```

```
break;

else

i=0;

}

}

}

int bankers::search(int i)

{

int j;

for(j=0;j<nor;j++)

if(n[i][j]>avail[j])

return -1;

return 0;

}

void bankers::display()

{

int i,j;

cout<<endl<<"OUTPUT:";

cout<<endl<<"=====";

cout<<endl<<"PROCESS\t ALLOTTED\t MAXIMUM\t NEED";

for(i=0;i<nop;i++)

{

cout<<"\nP"<<i+1<<"\t ";

for(j=0;j<nor;j++)

{
```

```
cout<<al[i][j]<<" ";

}

cout<<"\t  ";

for (j=0;j<nor;j++)

{

cout<<m[i][j]<<" ";

}

cout<<"\t";

for(j=0;j<nor;j++ )

{

cout<<n[i][j]<<" ";

}

}

cout<<"\nThe sequence of the safe processes are: \n";

for(i=0;i<k;i++)

{

int temp = result[i]+1 ;

cout<<"P"<<temp<<" ";

}

cout<<"\nThe sequence of unsafe processes are: \n";

int flg=0;

for (i=0;i<nop;i++)

{

if(finish[i]==0)

{
```



```
    flg=1;
}
cout<<"P"<<i<<" ";
}
cout<<endl<<"RESULT:";
cout<<endl<<"=====";
if(flg==1)
cout<<endl<<"The system is not in safe state and deadlock may occur!!";
else
cout<<endl<<"The system is in safe state and deadlock will not occur!!";
}
void main()
{
clrscr();
cout<<" DEADLOCK BANKER'S ALGORITHM "<<endl;
bankers B;
B.input ( );
B.method ( );
B.display ( );
getch();
}
```

Output:

DEADLOCK BANKER'S ALGORITHM

Enter the number of processes:3

Enter the number of resources:1

Enter the allocated resources for each process:

Process 0

Resource 0:1

Process 1

Resource 0:2

Process 2

Resource 0:3

Enter the maximum resources that are needed for each process:

Process 0

Resource 0:4

Process 1

Resource 0:5

Process 2

Resource 0:_

Enter the maximum resources that are needed for each process:

Process 0

Resource 0:4

Process 1

Resource 0:5

Process 2

Resource 0:6

Enter the currently available resources in the system: Resource 0:7

OUTPUT:

=====

PROCESS	ALLOTTED	MAXIMUM	NEED
P1	1	4	3
P2	2	5	3
P3	3	6	3

The sequence of the safe processes are:

P1 P2 P3

The sequence of unsafe processes are:

P0 P1 P2

RESULT:

=====

The system is in safe state and deadlock will not occur!!

*****Happy Learning*****

CHANDIGARH UNIVERSITY, GHARUAN (MOHALI)