LINUX / UNIX

What is Unix?

Basically, the **Unix** is an operating system which is a set of programs that act as a connection between the computer and the user.

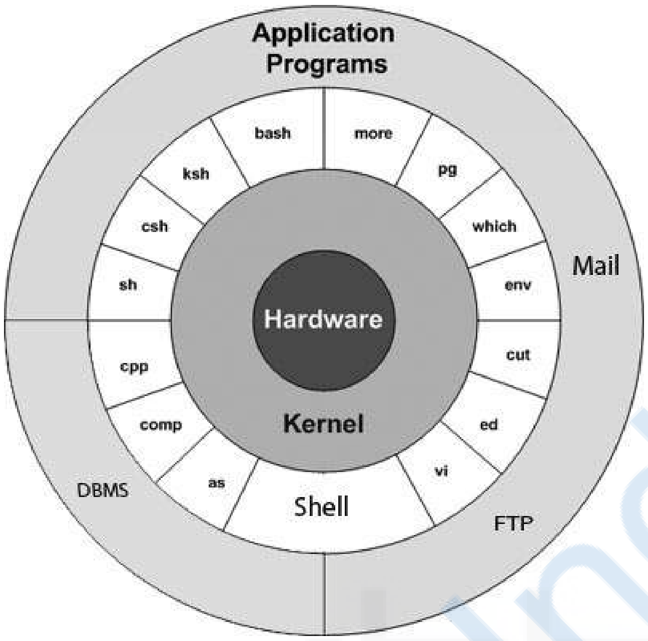
So, what is an Operating system here?

The computer programs that allocate the system resources and coordinate all the details of the computer's internals is called the operating system or the kernel.

Users communicate with the kernel through a program known as the shell. The shell is a command-line interpreter; it accepts and translates commands entered by the user and converts them into a language that is understood by the kernel.

- Unix was originally developed in 1969 by a group of AT&T employees Ken Thompson, Dennis Ritchie, Douglas McIlroy, and Joe Ossanna at Bell Labs.
- There are various Unix variants available in the market. Solaris Unix, AIX, HP Unix and BSD are a few examples. Linux is also a flavour of Unix which is freely available.
- Several people can use a Unix computer at the same time; hence Unix is called a **multiuser** system.
- A user can also run multiple programs at the same time; hence Unix is a multitasking environment.

Unix Architecture



Above is a basic block diagram of a Unix system;

The main concept that unites all the versions of Unix is the following four basics −

- **Kernel** − The kernel is the heart of the operating system. It interacts with the hardware and most of the tasks like memory management, task scheduling and file management.
- **Shell** − The shell is the utility that processes your requests. When you type in a command at your terminal, the shell interprets the command and calls the program that you want. The shell uses standard syntax for all commands. C Shell, Bourne Shell and Korn Shell are the most famous shells which are available with most of the Unix variants.
- **Commands and Utilities** − There are various commands and utilities which you can make use of in your day to day activities. cp, mv, cat and grep, etc. are a few examples of commands and utilities. There are over 250 standard

commands plus numerous others provided through 3rd party software. All the commands come along with various options.

- **Files and Directories** − All the data of Unix is organized into files. All files are then organized into directories. These directories are further organized into a tree-like structure called the filesystem.

Resources :

https://opensource.com/article/18/5/differences-between-linux-and-unix

Let's get started with few Linux Commands

login

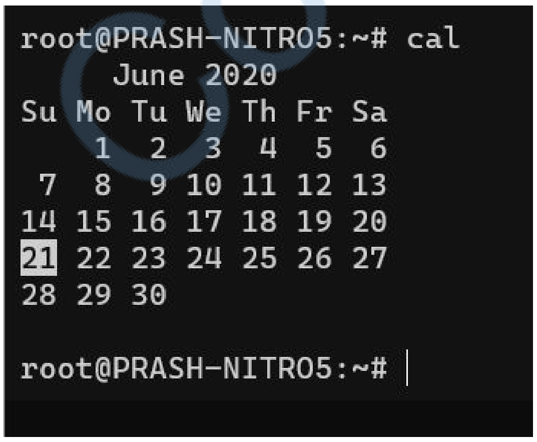
The command for logging into the Linux

```
$ sudo login
```

cal

To display the calendar

```
$ cal
```



```
root@PRASH-NITRO5:~# cal
     June 2020
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

root@PRASH-NITRO5:~#
```

passwd

To Change Password

```
$ sudo passwd
```

ls

To List the directories and files

```
$ ls
```

whoami

To Display the “Logged in” User Name

```
$ whoami
```

Most frequently used commands in Linux

init

To Shut down

```
$ sudo init 0
```

To Restart

```
$ sudo init 6
```

Man pages (also called manual pages) on your Unix or Linux computer.

man

To know more about any command type:

Syntax-

```
$ man [command]
```

Examples-

```
$ man ls
```

```
$ man whoami
```

whatis

To see just the description of a manual page, use **“whatis”** followed by a command.

```
$ whatis [command]
```

```
$ whatis ls
```

whereis

whereis The location of a manpage can be revealed with **whereis**.

```
$ whereis [command]
```

Example-

```
$ whereis ls
```

LINUX DIRECTORIES

pwd

The pwd (Print Working Directory) command displays your current directory

```
root@DESKTOP-ICC1VK0:~$ pwd
/home/username
```

cd

You can change your current directory with the cd command (Change Directory).

```
root@DESKTOP-ICC1VK0:~$ cd /etc
root@DESKTOP-ICC1VK0:/etc$ pwd
```

cd ~

This is also a shortcut to get back into your home directory.

```
root@DESKTOP-ICC1VK0:~$ cd /etc
root@DESKTOP-ICC1VK0:/etc$ cd ~
root@DESKTOP-ICC1VK0:~$
```

cd ..

To go to the parent directory (the one just above your current directory in the directory tree).

```
root@DESKTOP-ICC1VK0:~$ pwd
/home/root
root@DESKTOP-ICC1VK0:~$ cd ..
root@DESKTOP-ICC1VK0:/home$ pwd
/home
```

cd -

cd - is a command to go to the previous directory in your history of commands.

```
root@DESKTOP-ICC1VK0:~$ cd /etc
root@DESKTOP-ICC1VK0:/etc$ pwd
/etc
root@DESKTOP-ICC1VK0:/etc$ cd -
/home/root
root@DESKTOP-ICC1VK0:~$ cd -
/etc
root@DESKTOP-ICC1VK0:/etc$
```

Absolute and Relative paths

You should be aware of the absolute and relative paths in the file tree. When you type a path starting with a slash (/), then the root of the file tree is assumed. If you don't start your path with a slash, then the current directory is the assumed starting point.

The Output below first shows the current directory **/home/username**. From within this directory, you have to type **cd /home** instead of **cd home** to go to the **/home** directory.

```
root@DESKTOP-ICC1VK0:~$ pwd
/home/root
root@DESKTOP-ICC1VK0:~$ cd home
-bash: cd: home: No such file or directory
root@DESKTOP-ICC1VK0:~$ cd /home
root@DESKTOP-ICC1VK0:/home$ pwd
/home
```

When inside **/home**, you have to type **cd username** instead of **cd /username** to enter the subdirectory username of the current directory **/home.**

```
root@DESKTOP-ICC1VK0:/home$ pwd
/home
root@DESKTOP-ICC1VK0:/home$ cd /username
-bash: cd: /username: No such file or directory
root@DESKTOP-ICC1VK0:/home$ cd username
root@DESKTOP-ICC1VK0:~$ pwd
/home/username
```

In case your current directory is the root directory /, then both **cd /home** and **cd home** will get you in the **/home** directory.

```
root@DESKTOP-ICC1VK0:/$ pwd
/
root@DESKTOP-ICC1VK0:/$ cd home
```

```
root@DESKTOP-ICC1VK0:/home$ pwd
/home
root@DESKTOP-ICC1VK0:/home$ cd /
root@DESKTOP-ICC1VK0:/$ cd /home
root@DESKTOP-ICC1VK0:/home$ pwd
/home
```

Path Completion

The **tab key** can help you in typing a path without errors. Typing **cd /et** followed by the **tab key** will expand the command line to **cd /etc/**. When typing **cd /Et** followed by the **tab key**, nothing will happen because you typed the wrong path (upper case E). You will need fewer keystrokes when using the tab key, and you will be sure your typed path is correct!

ls

This command lists the contents of a directory with **ls**.

```
username@DESKTOP-ICC1VK0:~$ ls
file.txt scan.txt
```

ls -a

A frequently used option with **ls** is **-a** to show all files. Showing all files means including the hidden files. When a file name on a Linux file system starts with a dot, it is considered a hidden file and it doesn't show up in regular file listings.

```
username@DESKTOP-ICC1VK0:~$ ls -a
file.txt scan.txt .hidden.txt
```

ls -l

Many times you will be using options with ls to display the contents of the directory in different formats or to display different parts of the directory. Typing just ls gives you a list of files in the directory. Typing ls **-l** (that is a letter L, not the number 1) gives you a long listing.

```
username@DESKTOP-ICC1VK0:~$ ls -l
file.txt scan.txt .hidden.txt
drwxr-xr-x 1 username username 4096 Jun 18 18:47 file.txt
-rw-r--r-- 1 username username  256 Jun 18 18:47 scan.txt
```

ls -lh

Another frequently used ls option is **-h**. It shows the numbers (file sizes) in a more human-readable format. Also shown below is some variation in the way you can give the options to ls. We will explain the details of the output later in this book.

Note that we use the letter L as an option in this screenshot, not the number 1.

```
username@DESKTOP-ICC1VK0:~/server$ ls -l -h
total 0
-rw-rw-rw- 1 username username 0 Jun 18 18:51 hey.txt
-rw-rw-rw- 1 username username 0 Jun 18 18:51 index.html
```

```
username@DESKTOP-ICC1VK0:~/server$ ls -lh
total 0
-rw-rw-rw- 1 username username 0 Jun 18 18:51 hey.txt
-rw-rw-rw- 1 username username 0 Jun 18 18:51 index.html
```

```
username@DESKTOP-ICC1VK0:~/server$ ls -hl
total 0
-rw-rw-rw- 1 username username 0 Jun 18 18:51 hey.txt
-rw-rw-rw- 1 username username 0 Jun 18 18:51 index.html
```

```
username@DESKTOP-ICC1VK0:~/server$ ls -h -l
total 0
-rw-rw-rw- 1 username username 0 Jun 18 18:51 hey.txt
-rw-rw-rw- 1 username username 0 Jun 18 18:51 index.html
```

mkdir

We can create our own directories with the help of **mkdir.** Make sure you give at least one parameter to mkdir, the name of the new directory to be created.

```
username@DESKTOP-ICC1VK0:~$ mkdir new
username@DESKTOP-ICC1VK0:~$ cd new
username@DESKTOP-ICC1VK0:~/new$ ls -al
total 0
drwxrwxrwx 1 username username 4096 Jun 21 23:10 .
drwxr-xr-x 1 username username 4096 Jun 21 23:10 ..
username@DESKTOP-ICC1VK0:~/new$ mkdir new2
username@DESKTOP-ICC1VK0:~/new$ ls -l
total 0
drwxrwxrwx 1 username username 4096 Jun 21 23:11 new2
```

mkdir -p

The following command will fail, because the parent directory of threedirsdeep does not exist.

```
username@DESKTOP-ICC1VK0:~/new$ mkdir create1/create2/create3
mkdir: cannot create directory 'create1/create2/create3': No such file or
directory
```

When given the option -p, then mkdir will create parent directories as needed

```
username@DESKTOP-ICC1VK0:~/new$ mkdir -p create1/create2/create3
username@DESKTOP-ICC1VK0:~/new$ cd create1
```

```
username@DESKTOP-ICC1VK0:~/new/create1$ cd create2
username@DESKTOP-ICC1VK0:~/new/create1/create2$ cd create3
username@DESKTOP-ICC1VK0:~/new/create1/create2/create3$ pwd
/home/username/new/create1/create2/create3
```

rmdir

When a directory is empty, you can use **rmdir** to remove the directory.

```
username@DESKTOP-ICC1VK0:~/new/create1/create2/create3$ cd ..
username@DESKTOP-ICC1VK0:~/new/create1/create2$ rmdir create3
username@DESKTOP-ICC1VK0:~/new/create1/create2$ cd ..
username@DESKTOP-ICC1VK0:~/new/create1$ rmdir create2
username@DESKTOP-ICC1VK0:~/new/create1$ cd ..
username@DESKTOP-ICC1VK0:~/new$ rmdir create1
```

rmdir -p

And similar to the **mkdir -p** option, you can also use **rmdir** to recursively remove directories.

```
username@DESKTOP-ICC1VK0:~/new$ mkdir -p create1/create2/create3
username@DESKTOP-ICC1VK0:~/new$ ls
create1  new2
username@DESKTOP-ICC1VK0:~/new$ rmdir -p create1/create2/create3
username@DESKTOP-ICC1VK0:~/new$ ls
new2
```

pushd, popd

The Bash shell has two built-in commands called pushd and popd. Both commands work with a common stack of previous directories. Pushd adds a directory to the stack and changes to a new directory, popd removes a directory from the stack and sets the current directory.

```
root@DESKTOP-ICC1VK0:~$ pushd /home/
/home ~
root@DESKTOP-ICC1VK0:/home$ pushd /etc
/etc /home ~
root@DESKTOP-ICC1VK0:/etc$ pushd /mnt/
/mnt /etc /home ~
root@DESKTOP-ICC1VK0:/mnt$ popd
/etc /home ~
root@DESKTOP-ICC1VK0:/etc$ popd
/home ~
root@DESKTOP-ICC1VK0:/home$ popd
~
```

Exercise:

1. Display your current directory.

```
$ pwd
```

2. Change to the /etc directory.

```
$ cd /etc
```

5. Go to the parent directory of the current directory.

```
$ cd ..
```

6. Go to the root directory.

```
$ cd /
```

7. List the contents of the root directory.

```
$ ls
```

8. List a long listing of the root directory.

```
$ ls -l
```

9. Stay where you are, and list the contents of /etc.

```
$ ls /etc
```

10. Stay where you are, and list the contents of /bin and /sbin.

```
$ ls /bin/sbin
```

11. Stay where you are, and list the contents of ~.

```
$ ls ~
```

12. List all the files (including hidden files) in your home directory.

```
$ ls -al ~
```

13. List the files in /boot in a human readable format.

```
$ ls -lh /boot
```

14. Create a directory testdir in your home directory.

```
$ mkdir ~/testdir
```

15. Change to the /etc directory, stay here and create a directory newdir in your home directory.

```
$ cd /etc ; mkdir ~/newdir
```

16. Create in one command the directories ~/dir1/dir2/dir3 (dir3 is a subdirectory from dir2, and dir2 is a subdirectory from dir1).

```
$ mkdir -p ~/dir1/dir2/dir3
```

17. Remove the directory testdir.

rmdir testdir

```
$ rmdir testdir
```

LINUX FILES

All files are case sensitive

Big boy is not the same as a small boy. I mean Capslock filename is different from the small letters file name.

Example **File.txt** is different from **file.txt**

```
root@DESKTOP-ICC1VK0:/home$ ls
winter.txt  Winter.txt
root@DESKTOP-ICC1VK0:/home$ cat winter.txt
It is cold.
root@DESKTOP-ICC1VK0:/home$ cat Winter.txt
It is very cold
```

Everything is a file

A directory is a special kind of file, but it is still a (case sensitive!) file. Each terminal window (for example /dev/pts/4), any hard disk or partition (for example /dev/sdb1) and any process are all represented somewhere in the file system as a file. It will become clear throughout this course that everything on Linux is a file.

file

The file utility determines the file type. Linux does not use extensions to determine the file type. The command line does not care whether a file ends in **.txt** or **.pdf**. As a linux system administrator, you should use the file command to determine the file type. Here are some examples of a typical Linux system.

```
root@DESKTOP-ICC1VK0:/home$ file pic.png
pic.png: PNG image data, 380 x 120, 8-bit/color RGBA, non-interlaced
root@DESKTOP-ICC1VK0:/home$ file /etc/passwd
```

```
/etc/passwd: ASCII text
root@DESKTOP-ICC1VK0:/home$ file HelloWorld.c
HelloWorld.c: ASCII C program text
```

It is interesting to point out **file -s** for special files like those in /dev and /proc

```
root@DESKTOP-ICC1VK0:/home$  file /dev/sda
/dev/sda: block special
root@DESKTOP-ICC1VK0:/home$  file -s /dev/sda
/dev/sda: x86 boot sector; partition 1: ID=0x83, active, starthead...
root@DESKTOP-ICC1VK0:/home$  file /proc/cpuinfo
/proc/cpuinfo: empty
root@DESKTOP-ICC1VK0:/home$  file -s /proc/cpuinfo
/proc/cpuinfo: ASCII C++ program text
```

touch

Create an empty file

One easy way to create an empty file is with **touch**. (We will see many other ways for creating files later).

```
root@DESKTOP-ICC1VK0:~$ ls -l
total 0
-rw-r--r-- 1 root root 0 Jun 22 00:08 file.txt
root@DESKTOP-ICC1VK0:~$  touch hey.txt
root@DESKTOP-ICC1VK0:~$  touch hello.txt
root@DESKTOP-ICC1VK0:~$  ls -l
total 0
-rw-r--r-- 1 root root 0 Jun 22 00:08 file.txt
-rw-r--r-- 1 root root 0 Jun 22 00:20 hello.txt
-rw-r--r-- 1 root root 0 Jun 22 00:20 hey.txt
```

touch -t

The touch command can set some properties while creating empty files. Can you determine what is set by looking at the next output? If not, check the manual for touch.

```
root@DESKTOP-ICC1VK0:/home$ touch -t 20050505 hello
root@DESKTOP-ICC1VK0:/home$ touch -t 13020711 BigBattle.txt
root@DESKTOP-ICC1VK0:/home$ ls -l
total 0
-rw-r--r-- 1 paul paul 0 Jul 11  1302 BigBattle.txt
-rw-r--r-- 1 paul paul 0 Oct 15 08:57 file33
-rw-r--r-- 1 paul paul 0 Oct 15 08:56 file42
-rw-r--r-- 1 paul paul 0 May  5  2005 hello
```

rm

remove forever

When you no longer need a file, use **rm** to remove it. Unlike some graphical user interfaces, the command line, in general, does not have a waste bin or trash can to recover files. When you use rm to remove a file, the file is gone. Therefore, be careful when removing files!

```
root@DESKTOP-ICC1VK0:/home$  ls
File1.txt  file2.txt  hello.txt
root@DESKTOP-ICC1VK0:/home$  rm hello.txt
root@DESKTOP-ICC1VK0:/home$  ls
File1.txt  file2.txt
root@DESKTOP-ICC1VK0:/home$  rm file2.txt
root@DESKTOP-ICC1VK0:/home$  ls
File1.txt
```

rm -i

To prevent yourself from accidentally removing a file, you can type rm -i.

```
root@DESKTOP-ICC1VK0:/home$  ls
File1.txt  file2.txt  hello.txt
root@DESKTOP-ICC1VK0:/home$  rm -i hello.txt
rm: remove regular empty file `hello.txt'? yes
root@DESKTOP-ICC1VK0:/home$  ls
File1.txt  file2.txt
```

rm -rf

By default, **rm -r** will not remove non-empty directories. However, rm accepts several options that will allow you to remove any directory. The **rm -rf** statement is famous because it will erase anything (providing that you have the permissions to do so). When you are logged on as root, be very careful with **rm -rf** (the **f means force** and the **r means recursive**) since being root implies that permissions don't apply to you. You can literally erase your entire file system by accident.

```
root@DESKTOP-ICC1VK0:/home$  ls
File1.txt  file2.txt  hello.txt
root@DESKTOP-ICC1VK0:/home$  rm -rf hello.txt
root@DESKTOP-ICC1VK0:/home$  ls hello.txt
ls: cannot access test: No such file or directory
root@DESKTOP-ICC1VK0:/home$  ls
File1.txt  file2.txt
```

cp

Copy one file

To copy a file, use cp with a source and a target argument.

```
root@DESKTOP-ICC1VK0:/home$  ls
File1.txt  hello.txt
root@DESKTOP-ICC1VK0:/home$  cp hello.txt hello.txt copy
root@DESKTOP-ICC1VK0:/home$  ls
File1.txt  hello.txt  hello.txt copy
```

Copy to another directory

If the target is a directory, then the source files are copied to that target directory.

```
root@DESKTOP-ICC1VK0:/home$  mkdir dir2
root@DESKTOP-ICC1VK0:/home$  cp hello.txt dir2
root@DESKTOP-ICC1VK0:/home$  ls dir2/
hello.txt
```

cp -r

To copy complete directories, use **cp -r** (the **-r option forces recursive copying** of all files in all subdirectories).

```
root@DESKTOP-ICC1VK0:/home$  ls
dir42  file42  file42.copy  hey.txt
root@DESKTOP-ICC1VK0:/home$  cp -r dir42/ dir33
root@DESKTOP-ICC1VK0:/home$  ls
dir33  dir42  file42  file42.copy  hey.txt
root@DESKTOP-ICC1VK0:/home$ ls dir33/
hey.txt
```

Copy multiple files to a directory

You can also use **cp** to copy multiple files into a directory. In this case, the last argument (a.k.a. the target) must be a directory.

```
root@DESKTOP-ICC1VK0:/home$ cp file42  file42.copy  hey.txt dir42/
root@DESKTOP-ICC1VK0:/home$  ls dir42/
```

```
file42  file42.copy  hey.txt
```

cp -i

To prevent **cp** from overwriting existing files, use the **-i** (for interactive) option

```
root@DESKTOP-ICC1VK0:~$ ls
file.txt  helli.txt  hello.txt  hey.txt
root@DESKTOP-ICC1VK0:~$ cp -i file.txt helli.txt
cp: overwrite 'helli.txt'? N
root@DESKTOP-ICC1VK0:~$
```

mv

rename files with mv

Use **mv** to rename a file or to move the file to another directory.

```
root@DESKTOP-ICC1VK0:~$ ls
file.txt  helli.txt  hello.txt  hey.txt
root@DESKTOP-ICC1VK0:~$ mv helli.txt hello2.txt
root@DESKTOP-ICC1VK0:~$ ls
file.txt  hello.txt  hello2.txt  hey.txt
```

When you need to rename only one file then **mv** is the preferred command to use.

rename directories with mv

The same mv command can be used to rename directories.

```
root@DESKTOP-ICC1VK0:~$ ls -l
total 0
-rw-r--r-- 1 root root 0 Jun 22 00:08 file.txt
-rw-r--r-- 1 root root 0 Jun 22 11:32 hello.txt
-rw-r--r-- 1 root root 0 Jun 22 11:32 hello2.txt
-rw-r--r-- 1 root root 0 Jun 22 00:20 hey.txt
root@DESKTOP-ICC1VK0:~$ mv dir42 backup
```

```
root@DESKTOP-ICC1VK0:~$ ls -l
total 0
-rw-r--r-- 1 root root 0 Jun 22 00:08 file.txt
-rw-r--r-- 1 root root 0 Jun 22 11:32 hello.txt
-rw-r--r-- 1 root root 0 Jun 22 11:32 hello2.txt
-rw-r--r-- 1 root root 0 Jun 22 00:20 hey.txt
```

mv -i

The mv also has a **-i** switch similar to **cp** and **rm.**

This example shows that **mv -i** will ask permission to overwrite an existing file.

```
root@DESKTOP-ICC1VK0:~$ mv -i file33 hello.txt
mv: overwrite `hello.txt'? no
root@DESKTOP-ICC1VK0:~$
```

Exercises:

1. List the files in the /bin directory

```
ls /bin
```

2. Display the type of file of /bin/cat, /etc/passwd and /usr/bin/passwd.

```
file /bin/cat /etc/passwd /usr/bin/passwd
```

3. Create a directory ~/touched and enter it.

```
mkdir ~/touched ; cd ~/touched
```

4. Create the files today.txt and yesterday.txt are touched.

```
touch today.txt yesterday.txt
```

5. Create a directory called ~/testbackup and copy all files from ~/touched into it.

```
mkdir ~/testbackup ; cp -r ~/touched ~/testbackup/
```

6. Use one command to remove the directory ~/testbackup and all files into it.

```
rm -rf ~/testbackup
```

7. Create a directory ~/etcbackup and copy all *.conf files from /etc into it. Did you include all subdirectories of /etc ?

```
cp -r /etc/*.conf ~/etcbackup
```

Working with File Contents

head

To display the first ten lines of a file

```
root@DESKTOP-ICC1VK0:~$ head /root/hello.txt
It is a long established fact that a reader will be distracted by the
readable content of a page when looking at its layout. The point of
using Lorem Ipsum is that it has a more-or-less normal
distribution of letters, as opposed to using 'Content here,content
here', making it look like readable English. Many desktop publishing
packages and web page editors now use Lorem Ipsum as their default model
text, and a search for 'lorem ipsum' will uncover many websites still in
their infancy. Various versions have evolved over the years, sometimes
by
accident, sometimes on purpose (injected humour and the like).
root@DESKTOP-ICC1VK0:~$
```

The head command can also display the first n lines of a file

```
root@DESKTOP-ICC1VK0:~$  head -4 /etc/passwd
It is a long established fact that a reader will be distracted by the
readable content of a page when looking at its layout. The point of
using Lorem Ipsum is that it has a more-or-less normal
```

```
distribution of letters, as opposed to using 'Content here,content
root@DESKTOP-ICC1VK0:~$
```

And head can also display the first n bytes

```
root@DESKTOP-ICC1VK0:~$   head -c14 /etc/passwd
root:x:0:0:rooroot@DESKTOP-ICC1VK0:~$
```

tail

Similar to **head**, the **tail** command will display the last ten lines of a file.

```
root:x:0:0:rooroot@DESKTOP-ICC1VK0:~$ tail /root/hello.txt
uncover many websites still in their infancy. Various versions have
evolved over the years, sometimes by
accident, sometimes on purpose (injected humour and the like).
Lorem 10 Lorem Ipsum is simply dummy
text of the printing and typesetting industry. Lorem Ipsum has been the
industry's standard
dummy text ever since the 1500s, when an unknown printer took a galley
of type and scrambled it to make a type specimen book. It has survived
not only five centuries, but also the leap into electronic typesetting,
remaining essentially unchanged
```

You can give tail the number of lines you want to see.

```
root@DESKTOP-ICC1VK0:~$ tail -3 /root/hello.txt
dummy text ever since the 1500s, when an unknown printer took a galley
of type and scrambled it to make a type specimen book. It has survived
not only five centuries, but also the leap into electronic typesetting,
remaining essentially unchanged
```

cat

The **cat** command is one of the most universal tools, yet all it does is copy standard

input to standard output. In combination with the shell this can be very powerful and diverse. Some examples will give a glimpse into the possibilities. The first example is simple, you can use cat to display a file on the screen. If the file is longer than the screen, it will scroll to the end.

```
root@DESKTOP-ICC1VK0:~$ cat hello.txt
It is a long established fact that a reader will be distracted by the
readable content of a page when looking at its layout. The point of
using Lorem Ipsum is that it has a more-or-less normal
distribution of letters, as opposed to using 'Content here,content
here', making it look like readable English. Many desktop publishing
packages and web page editors now use Lorem Ipsum as their default model
text, and a search for 'lorem ipsum' will uncover many websites still in
their infancy. Various versions have evolved over the years, sometimes
by
accident, sometimes on purpose (injected humour and the like).
Lorem10 Lorem Ipsum is simply dummy
text of the printing and typesetting industry. Lorem Ipsum has been the
industry's standard
dummy text ever since the 1500s, when an unknown printer took a galley
of type and scrambled it to make a type specimen book. It has survived
not only five centuries, but also the leap into electronic typesetting,
remaining essentially unchanged
```

Concatenate

cat is short for concatenate. One of the basic uses of **cat** is to concatenate files into a bigger (or complete) file.

```
root@DESKTOP-ICC1VK0:~$ echo one >part1
root@DESKTOP-ICC1VK0:~$ echo two >part2
root@DESKTOP-ICC1VK0:~$ echo three >part3
root@DESKTOP-ICC1VK0:~$ cat part1
one
root@DESKTOP-ICC1VK0:~$ cat part2
two
```

```
root@DESKTOP-ICC1VK0:~$ cat part3
three
root@DESKTOP-ICC1VK0:~$ cat part1 part2 part3
one
two
three
root@DESKTOP-ICC1VK0:~$ cat part1 part2 part3 >all
root@DESKTOP-ICC1VK0:~$ cat all
one
two
three
```

Create files

You can use cat to create flat text files. Type the **cat > winter.txt** command as shown in the screenshot below. Then type one or more lines, finishing each line with the enter key. After the last line, type and hold the Control (Ctrl) key and press d.

```
root@DESKTOP-ICC1VK0:~$ cat > winter.txt
It is very cold today!
root@DESKTOP-ICC1VK0:~$ cat winter.txt
It is very cold today!
```

The **Ctrl+d** key combination will send an EOF(End of File) to the running process ending the cat command.

custom end marker

You can choose an end marker for cat with << as is shown in this example. This construction is called a here directive and will end the cat command.

```
root@DESKTOP-ICC1VK0:~$ cat > winter.txt <<stop
> It is very cold today!
> Yes, It is summer.
> stop
```

```
root@DESKTOP-ICC1VK0:~$ cat winter.txt
It is very cold today!
Yes, It is summer.
```

copy files

In the third example you will see that **cat** can be used to copy files.

```
root@DESKTOP-ICC1VK0:~$ cat winter.txt
It is very cold today!
root@DESKTOP-ICC1VK0:~$ cat winter.txt > cold.txt
root@DESKTOP-ICC1VK0:~$ cat cold.txt
It is very cold today!
```

tac

Just one example will show you the purpose of tac (cat backwards).

```
root@DESKTOP-ICC1VK0:~$ cat count
one
two
three
root@DESKTOP-ICC1VK0:~$ tac count
three
two
one
```

strings

With the strings command you can display readable ascii strings found in (binary) files. This example locates the **ls** binary then displays readable strings in the binary file (output is truncated).

```
root@DESKTOP-ICC1VK0:~$ which ls
/bin/ls
root@DESKTOP-ICC1VK0:~$ strings /bin/ls
/lib/ld-linux.so.2
librt.so.1
```

```
__gmon_start__
_Jv_RegisterClasses
clock_gettime libacl.so.1
...
```

Exercises:

1. Display the first 12 lines of /etc/services.

```
head -12 /etc/services
```

2. Display the last line of /etc/passwd.

```
tail -1 /etc/passwd
```

3. Use cat to create a file named count.txt that looks like this:

```
cat > count.txt One Two Three Four Five (followed by Ctrl-d)
```

4. Use cp to make a backup of this file to cnt.txt.

```
cp count.txt cnt.txt
```

5. Use cat to make a backup of this file to catcnt.txt.

```
cat count.txt > catcnt.txt
```

6. Display catcnt.txt, but with all lines in reverse order (the last line first).

```
tac catcnt.txt
```

7. Use more to display /etc/services.

```
cat /etc/services
```

8. Display the readable character strings from the /usr/bin/passwd command.

```
strings /usr/bin/passwd
```

9. Use ls to find the biggest file in /etc.

```
ls -lrS /etc
```

shell history

repeating the last command

To repeat the last command in bash, type **!!** This is pronounced as bang bang.

```
root@DESKTOP-ICC1VK0:~$ echo this will be repeated > file.txt
root@DESKTOP-ICC1VK0:~$ !!
echo this will be repeated > file.txt
```

repeating other commands

You can repeat other commands using one bang followed by one or more characters. The shell will repeat the last command that started with those characters.

```
root@DESKTOP-ICC1VK0:~$ touch file.txt
root@DESKTOP-ICC1VK0:~$ cat file.txt
root@DESKTOP-ICC1VK0:~$ !to
touch file.txt
```

history

To see older commands, use history to display the shell command history (or use history n to see the last n commands).

```
root@DESKTOP-ICC1VK0:~$ history 10
38 mkdir test
39  cd test
40  touch file1
41  echo hello > file2
42  echo It is very cold today > winter.txt
43  ls
44  ls -l
45  cp winter.txt summer.txt
46  ls -l
```

```
47  history 10
```

Note : In most linux distros, the history of commands will be saved in the below file path.

```
cat ~/.bash_history
```

!n

When typing **!** followed by the number preceding the command you want repeated, then the shell will echo the command and execute it.

```
root@DESKTOP-ICC1VK0:~$ !43
ls
file1  file2  summer.txt  winter.tx
```

Ctrl-r

Another option is to use **ctrl-r** to search in the history. In the screenshot below **i** only typed **ctrl-r** followed by four characters apti and it finds the last command containing these four consecutive characters.

```
root@DESKTOP-ICC1VK0:~$
(reverse-i-search)`apti': sudo aptitude install screen
```

$HISTSIZE

The $HISTSIZE variable determines the number of commands that will be remembered in your current environment. Most distributions default this variable to 500 or 1000.

```
root@DESKTOP-ICC1VK0:~$  echo $HISTSIZE
500
```

You can change it to any value you like.

```
root@DESKTOP-ICC1VK0:~$  HISTSIZE=15000
root@DESKTOP-ICC1VK0:~$  echo $HISTSIZE
```

```
15000
```

$HISTFILE

The $HISTFILE variable points to the file that contains your history. The bash shell defaults this value to ~/.bash_history.

```
root@DESKTOP-ICC1VK0:~$  echo $HISTFILE
/home/root/.bash_history
```

$HISTFILESIZE

The number of commands kept in your history file can be set using $HISTFILESIZE.

```
root@DESKTOP-ICC1VK0:~$ echo $HISTFILESIZE
15000
```

prevent recording a command

You can prevent a command from being recorded in history using a **space prefix**.

```
root@DESKTOP-ICC1VK0:~$ echo abc
abc
root@DESKTOP-ICC1VK0:~$  echo def
def
root@DESKTOP-ICC1VK0:~$ echo ghi
ghi
root@DESKTOP-ICC1VK0:~$ history 3
 9501 echo abc
 9502  echo ghi
 9503  history 3
```

Exercise:

Display the last 5 commands you typed

```
root@DESKTOP-ICC1VK0:~$ history 5
   24  ls -l
   25  exit
   26  cal
   27  exit
   28  history 5
root@PRASH-NITRO5:~$
```

file globbing

file globbing is nothing but dynamic filename generation.

* asterisk

The **asterisk *** is interpreted by the shell as a sign to generate filenames, matching the asterisk to any combination of characters (even none). When no path is given, the shell will use filenames in the current directory.

```
root@DESKTOP-ICC1VK0:~/usage$ ls
File4  File55  FileA  FileAB  Fileab  file1  file2  file3  fileab
root@DESKTOP-ICC1VK0:~/usage$ ls File*
File4  File55  FileA  FileAB  Fileab
root@DESKTOP-ICC1VK0:~/usage$ ls file*
file1  file2  file3  fileab
root@DESKTOP-ICC1VK0:~/usage$ ls *ile55
File55
root@DESKTOP-ICC1VK0:~/usage$ ls F*55
File55
```

? question mark

Similar to the asterisk, the question mark **?** is interpreted by the shell as a sign to generate filenames, matching the question mark with exactly one character.

```
root@DESKTOP-ICC1VK0:~/usage$ ls
File4  File55  FileA  FileAB  Fileab  file1  file2  file3  fileab
root@DESKTOP-ICC1VK0:~/usage$ ls File?
File4  FileA
```

```
root@DESKTOP-ICC1VK0:~/usage$ ls Fil?4
File4
root@DESKTOP-ICC1VK0:~/usage$ ls Fil??
File4  FileA
root@DESKTOP-ICC1VK0:~/usage$ ls File??
File55  FileAB  Fileab
```

[ ] square brackets

The square bracket [is interpreted by the shell as a sign to generate filenames, matching any of the characters between [and the first subsequent].

The order in this list between the brackets is not important. Each pair of brackets is replaced by exactly one character.

```
root@DESKTOP-ICC1VK0:~/usage$ ls
File4  File55  FileA  FileAB  Fileab  file1  file2  file3  fileab
root@DESKTOP-ICC1VK0:~/usage$ ls File[5A]
FileA
root@DESKTOP-ICC1VK0:~/usage$ ls File[A5]
FileA
root@DESKTOP-ICC1VK0:~/usage$ ls File[A5][5b]
File55
root@DESKTOP-ICC1VK0:~/usage$ ls File[a5][5b]
File55  Fileab
```

a-z and 0-9 ranges

The bash shell will also understand ranges of characters between brackets.

```
root@DESKTOP-ICC1VK0:~/usage$ ls
File4  File55  FileA  FileAB  Fileab  file1  file2  file3  fileab
root@DESKTOP-ICC1VK0:~/usage$ ls file[a-z]*
fileab
root@DESKTOP-ICC1VK0:~/usage$ ls file[0-9]
file1  file2  file3
```

$LANG and square brackets

But, don't forget the influence of the LANG variable. Some languages include lower case letters in an uppercase range (and vice versa).

```
root@DESKTOP-ICC1VK0:~/usage$ ls [A-Z]ile?
File4  FileA
root@DESKTOP-ICC1VK0:~/usage$ ls [a-z]ile?
file1  file2  file3
root@DESKTOP-ICC1VK0:~/usage$ echo $LANG
C.UTF-8
root@DESKTOP-ICC1VK0:~/usage$ LANG=C
root@DESKTOP-ICC1VK0:~/usage$ echo $LANG
C
root@DESKTOP-ICC1VK0:~/usage$ ls [a-z]ile?
file1  file2  file3
root@DESKTOP-ICC1VK0:~/usage$ ls [A-Z]ile?
File4  FileA
```

If $LC_ALL is set, then this will also need to be reset to prevent file globbing.

preventing file globbing

The example below should be no surprise. The **echo *** will **echo a *** when in an empty directory. And it will echo the names of all files when the directory is not empty.

```
root@DESKTOP-ICC1VK0:~/usage$ mkdir test
root@DESKTOP-ICC1VK0:~/usage$ cd test
root@DESKTOP-ICC1VK0:~/usage/test$ echo *
*
root@DESKTOP-ICC1VK0:~/usage/test$ touch file1 file2
root@DESKTOP-ICC1VK0:~/usage/test$ echo *
file1 file2
```

Globbing can be prevented using quotes or by escaping the special characters, as shown in this example.

```
root@DESKTOP-ICC1VK0:~/usage/test$ echo *
file1 file2
root@DESKTOP-ICC1VK0:~/usage/test$ echo \*
*
root@DESKTOP-ICC1VK0:~/usage/test$ echo '*'
*
root@DESKTOP-ICC1VK0:~/usage/test$ echo "*"
*
```

Exercise:

1. Create a test directory and enter it.

```
mkdir testdir;
cd testdir
```

2. Create the following files :

file1; file10; file11; file2; File2; File3; file33; fileAB; filea; fileA; fileAAA; file(; file 2

(the last one file 2 - has 6 characters including a space)

```
touch file1 file10 file11 file2 File2 File3
touch file33 fileAB filea fileA fileAAA
touch "file("
touch "file 2"
```

3. List (with ls) all files starting with file.

```
ls file*
```

4. List (with ls) all files starting with File

```
ls File*
```

5. List (with ls) all files starting with file and ending in a number.

```
ls file*[0-9]
```

6. List (with ls) all files starting with file and ending with a letter.

```
ls file*[a-z]
```

7. List (with ls) all files starting with File and having a digit as the fifth character.

```
ls File*[0-9]*
```

8. List (with ls) all files starting with File and having a digit as fifth character and nothing else.

```
ls File[0-9]*
```

9. List (with ls) all files starting with a letter and ending in a number.

```
ls [a-z]*[0-9]
```

10. List (with ls) all files that have exactly five characters.

```
ls ?????
```

11. List (with ls) all files that start with f or F and end with 3 or A.

```
ls [fF]*[3A]
```

12. List (with ls) all files that start with f have i or R as second character and end in a number.

```
ls f[iR]*[0-9]
```

13. List all files that do not start with the letter F.

```
ls [!F]*
```

Linux Processes - Processes Management

A process, in simple terms, is an instance of a running program.

When you execute a program on your Unix system, the system creates a special environment for that program. This environment contains everything needed for the system to run the program as if no other program were running on the system.

Whenever you issue a command in Unix, it creates, or starts, a new process. When you tried out the ls command to list the directory contents, you started a process.

The operating system tracks processes through an ID number known as the pid or the process ID. Each process in the system has a unique pid.

Pids eventually repeat because all the possible numbers are used up and the next pid rolls or starts over. At any point of time, no two processes with the same pid exist in the system because it is the pid that Unix uses to track each process.

Starting a Process

When you start a process (run a command), there are two ways you can run it −

- Foreground Processes
- Background Processes

Foreground Processes

By default, every process that you start runs in the foreground. It gets its input from the keyboard and sends its output to the screen.

You can see this happen with the ls command. If you wish to list all the files in your current directory, you can use the following command −

```
$ ls ch*.doc
```

This would display all the files, the names of which start with ch and end with .doc −

```
ch01-1.doc   ch010.doc  ch02.doc    ch03-2.doc
ch04-1.doc   ch040.doc  ch05.doc    ch06-2.doc
ch01-2.doc   ch02-1.doc
```

The process runs in the foreground, the output is directed to screen(in which the above case it is ls command), and if the ls command wants any input (which it does not), it waits for it from the keyboard.

While a program is running in the foreground and is time-consuming, no other commands can be run (start any other processes) because the prompt would not be available until the program finishes processing and comes out.

Background Processes

A background process runs without being connected to your keyboard. If the background process requires any keyboard input, it waits.

The advantage of running a process in the background is that you can run other commands; you do not have to wait until it completes to start another!

The simplest way to start a background process is to add an ampersand (&) at the end of the command.

```
root@PRASH-NITRO5:~$ python -m SimpleHTTPServer 8080 &
[1] 213
root@PRASH-NITRO5:~$ Serving HTTP on 0.0.0.0 port 8080 ...

root@PRASH-NITRO5:~$
```

The above command will start the Http web server on 8080 port number as a background process.

Listing Running Processes

Check processes by running the ps (process status) command as follows −

```
root@PRASH-NITRO5:~$ ps
  PID TTY          TIME CMD
  155 tty1     00:00:00 init
  177 tty1     00:00:00 su
  178 tty1     00:00:00 bash
  213 tty1     00:00:00 python
  216 tty1     00:00:00 ps
root@PRASH-NITRO5:~$
```

One of the most commonly used flags for ps is the -f (f for full) option, which provides more information as shown in the following example −

```
root@PRASH-NITRO5:~$ ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
root       155     1  0 15:11 tty1     00:00:00 /init
root       177   156  0 15:12 tty1     00:00:00 su
root       178   177  0 15:12 tty1     00:00:00 bash
root     213 178 018:32 tty1 00:00:00 python -m SimpleHTTPServer 8080
root       217   178  0 18:37 tty1     00:00:00 ps -f
root@PRASH-NITRO5:~$
```

Stopping Processes

Ending a process can be done in several different ways. Often, from a console-based command, sending a CTRL + C keystroke (the default interrupt character) will exit the command. This works when the process is running in the foreground mode.

If a process is running in the background, you should get its Job ID using the ps command. After that, you can use the kill command to kill the process as follows −

Now lets kill the process that is running in the background.

```
root@PRASH-NITRO5:~$ ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
root       155     1  0 15:11 tty1     00:00:00 /init
root       177   156  0 15:12 tty1     00:00:00 su
root       178   177  0 15:12 tty1     00:00:00 bash
root     213 178 018:32 tty1 00:00:00 python -m SimpleHTTPServer 8080
root       217   178  0 18:37 tty1     00:00:00 ps -f
root@PRASH-NITRO5:~$
```

As you can see, the process id of the python background process is **213**

```
root@PRASH-NITRO5:~$ kill 213
[1]+  Terminated              python -m SimpleHTTPServer 8080
```

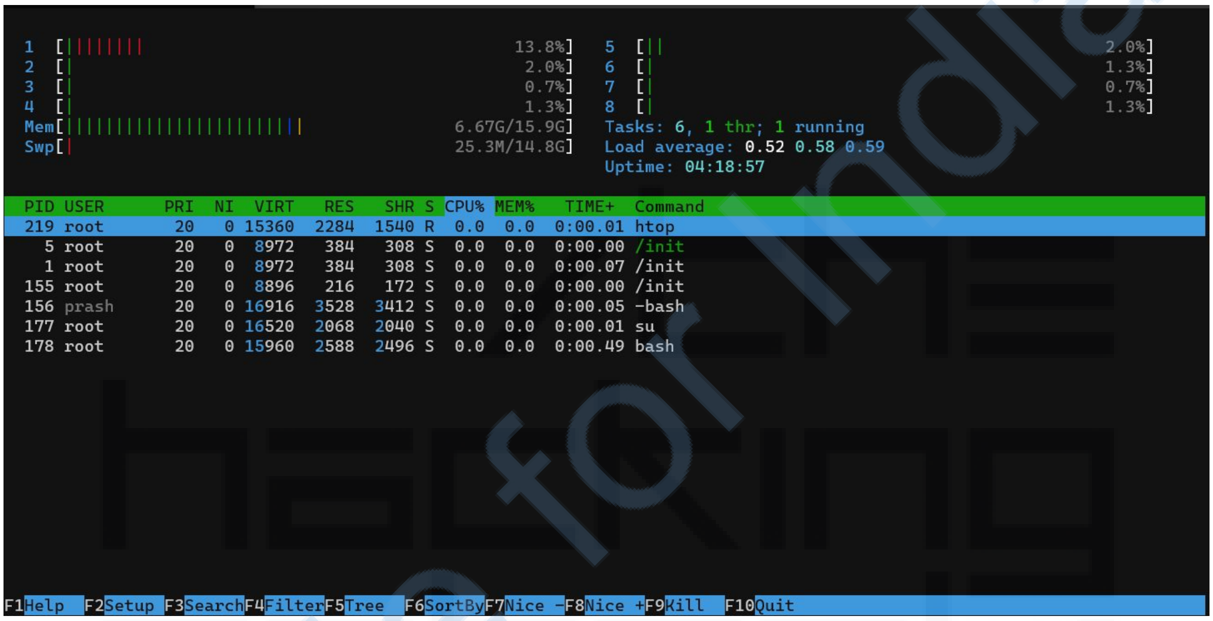
```
root@PRASH-NITRO5:~$
```

Linux Process Manager

htop

htop is an interactive system-monitor process-viewer and process-manager.

```
root@PRASH-NITRO5:~$ htop
```



```
1  [||||||||                                        13.8%]   5  [||                                              2.0%]
2  [|                                                2.0%]   6  [|                                               1.3%]
3  [|                                                0.7%]   7  [|                                               0.7%]
4  [|                                                1.3%]   8  [|                                               1.3%]
Mem[||||||||||||||||||||||||                   6.67G/15.9G]   Tasks: 6, 1 thr; 1 running
Swp[|                                          25.3M/14.8G]   Load average: 0.52 0.58 0.59
                                                              Uptime: 04:18:57

PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
219 root       20   0 15360  2284  1540 R  0.0  0.0  0:00.01 htop
  5 root       20   0  8972   384   308 S  0.0  0.0  0:00.00 /init
  1 root       20   0  8972   384   308 S  0.0  0.0  0:00.07 /init
155 root       20   0  8896   216   172 S  0.0  0.0  0:00.00 /init
156 prash      20   0 16916  3528  3412 S  0.0  0.0  0:00.05 -bash
177 root       20   0 16520  2068  2040 S  0.0  0.0  0:00.01 su
178 root       20   0 15960  2588  2496 S  0.0  0.0  0:00.49 bash

F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

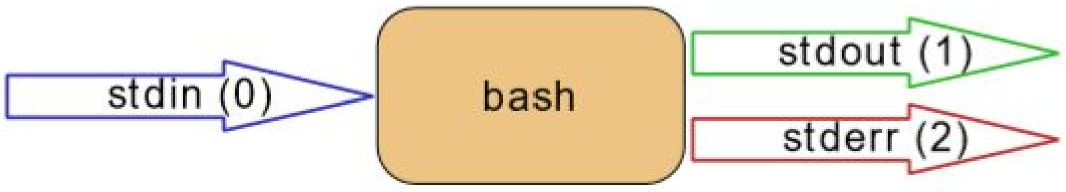
I/O redirection

One of the powers of the Unix command line is the use of input/output redirection and pipes.

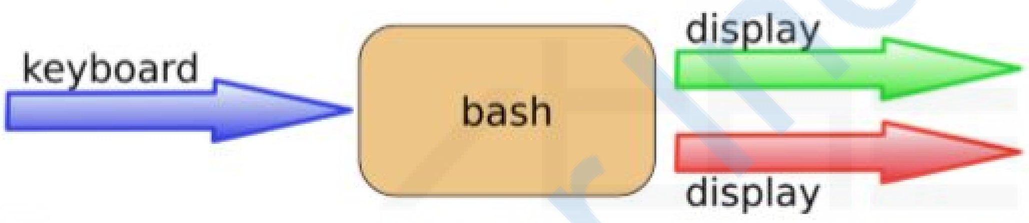
stdin, stdout, and stderr

The bash shell has three basic streams; it takes input from **stdin** (stream 0), it sends output to **stdout** (stream 1) and it sends error messages to **stderr** (stream 2).

The drawing below has a graphical interpretation of these three streams.



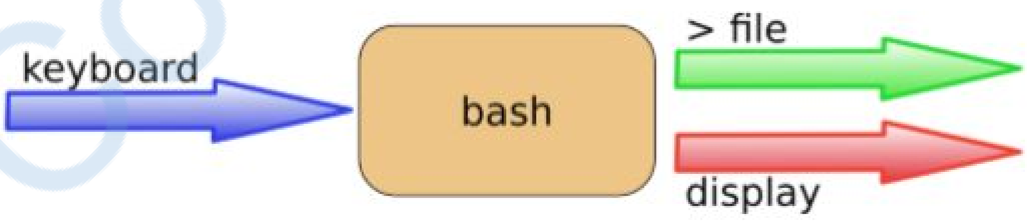
The keyboard often serves as **stdin**, whereas **stdout** and **stderr** both go to the display. This can be confusing to new Linux users because there is no obvious way to recognize **stdout** from **stderr**. Experienced users know that separating output from errors can be very useful.



Output redirection

stdout

stdout can be redirected with a **greater than** sign. While scanning the line, the shell will see the **>** sign and will clear the file.



The > notation is the abbreviation of 1> (stdout being referred to as stream 1).

```
root@DESKTOP-ICC1VK0:~/usage/test$ echo It is cold
It is cold today!
root@DESKTOP-ICC1VK0:~/usage/test$ echo It is cold > some.txt
root@DESKTOP-ICC1VK0:~/usage/test$ cat some.txt
```

```
It is cold
```

Note that the bash shell effectively removes the redirection from the command line before argument 0 is executed. This means that in the case of this command:

```
echo hello > greetings.txt
```

the shell only counts two arguments (echo = argument 0, hello = argument 1). The redirection is removed before the argument counting takes place.

Output file is erased

While scanning the line, the shell will see the > sign and will clear the file! Since this happens before resolving argument 0, this means that even when the command fails, the file will have been cleared!

```
root@DESKTOP-ICC1VK0:~/usage/test$ cat some.txt
It is cold
root@DESKTOP-ICC1VK0:~/usage/test$ zcho It is cold > some.txt

Command 'zcho' not found, did you mean:

  command 'echo' from deb coreutils (8.30-3ubuntu2)

Try: apt install <deb name>
root@DESKTOP-ICC1VK0:~/usage/test$ cat some.txt
root@DESKTOP-ICC1VK0:~/usage/test$
```

noclobber

Erasing a file while using **>** can be prevented by setting the **noclobber** option.

```
root@DESKTOP-ICC1VK0:~/usage/test$ cat some.txt
It is cold
root@DESKTOP-ICC1VK0:~/usage/test$ set -o noclobber
root@DESKTOP-ICC1VK0:~/usage/test$ echo It is cold > some.txt
-bash: some.txt: cannot overwrite existing file
root@DESKTOP-ICC1VK0:~/usage/test$ set +o noclobber
```

```
root@DESKTOP-ICC1VK0:~/usage/test$
```

overruling noclobber

The **noclobber** can be overruled with **>|**.

```
root@DESKTOP-ICC1VK0:~/usage/test$ set -o noclobber
root@DESKTOP-ICC1VK0:~/usage/test$ echo It is cold > some.txt
-bash: some.txt: cannot overwrite existing file
root@DESKTOP-ICC1VK0:~/usage/test$ echo It is cold >| some.txt
root@DESKTOP-ICC1VK0:~/usage/test$ cat some.txt
It is cold
root@DESKTOP-ICC1VK0:~/usage/test$
```

>> append

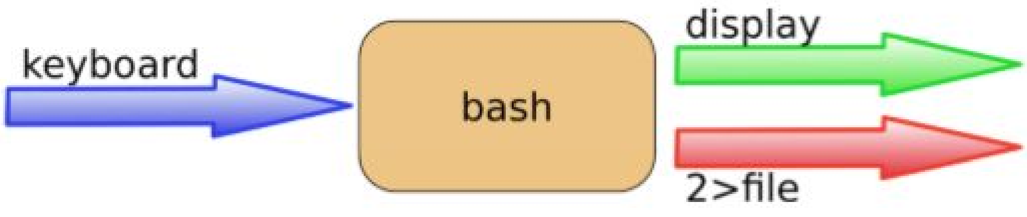
Use **>>** to **append** output to a file.

```
root@DESKTOP-ICC1VK0:~/usage/test$ echo It is cold > some.txt
root@DESKTOP-ICC1VK0:~/usage/test$ cat some.txt
It is cold
root@DESKTOP-ICC1VK0:~/usage/test$ echo where is hot >> some.txt
root@DESKTOP-ICC1VK0:~/usage/test$ cat some.txt
It is cold
where is hot
```

error redirection

> stderr

Redirecting **stderr** is done with **2>**. This can be very useful to prevent error messages from cluttering your screen.



2>&1

To redirect both **stdout** and **stderr** to the same file, use **2>&1**.

For example:

```
root@DESKTOP-ICC1VK0:~$ touch 1.js
root@DESKTOP-ICC1VK0:~$ cat 1.js
cnsole.log(2)
root@DESKTOP-ICC1VK0:~$ node 1.js > error 2>&1
root@DESKTOP-ICC1VK0:~$ cat error
/root/1.js:1
cnsole.log(7)
^
ReferenceError: cnsole is not defined
    at Object.<anonymous> (/root/1.js:1:1)
    at Module._compile (internal/modules/cjs/loader.js:955:30)
atObject.Module._extensions..js (internal/modules/cjs/loader.js:991:10)
    at Module.load (internal/modules/cjs/loader.js:811:32)
  at Function.Module._load (internal/modules/cjs/loader.js:723:14)
atFunction.Module.runMain (internal/modules/cjs/loader.js:1043:10)
    at internal/main/run_main_module.js:17:11
root@PRASH-NITRO5:~$
```

In the above example, a Javascript file with an error is created. And with the help of the command 2>&1 an error file is created that shows the errors of the js file.

joining stdout and stderr

The **&>** construction will put both **stdout** and **stderr** in one stream (to a file).

```
root@DESKTOP-ICC1VK0:~$ rm file42 &> out_and_err
root@DESKTOP-ICC1VK0:~$ cat out_and_err
rm: cannot remove 'file42': No such file or directory
root@DESKTOP-ICC1VK0:~$ echo file42 &> out_and_err
root@DESKTOP-ICC1VK0:~$ cat out_and_err file42
```

input redirection

< stdin

Redirecting stdin is done with < (short for 0<).

```
root@PRASH-NITRO5:~$ cat text.txt
sfsd
root@PRASH-NITRO5:~$ tr [a-z] [A-Z] < text.txt
SFSD
root@PRASH-NITRO5:~$
```

<< here document

The here document (sometimes called here-is-document) is a way to append input until a certain sequence (usually EOF) is encountered. The EOF marker can be typed literally or can be called with **Ctrl-D.**

```
root@PRASH-NITRO5:~$  cat <<EOF > text.txt
> one
> two
> three
> EOF
root@PRASH-NITRO5:~$ cat text.txt
one
two
three

root@PRASH-NITRO5:~$ cat <<pussycat > text.txt
> one
> two
> three
> pussycat
root@PRASH-NITRO5:~$ cat text.txt
one
two
three
root@PRASH-NITRO5:~$
```

<<< here string

The **here string** can be used to directly pass strings to a command. The result is the same as using **echo string | command** (but you have one less process running).

```
root@PRASH-NITRO5:~$ base64 <<< code.in
Y29kZS5pbgo=
root@PRASH-NITRO5:~$ base64 -d <<< Y29kZS5pbgo=
code.in
root@PRASH-NITRO5:~$
```

Exercise:

Start a Python HTTP Server as a background process. Redirect logs to log.txt file.

```
nohup python -m SimpleHTTPServer <PORT> >> log.txt
```

Filters

Commands that are created to be used with a pipe are often called filters.

Filter Commands :

1. cat
2. tee
3. grep
4. cut
5. tr
6. wc
7. sort
8. uniq
9. comm
10. od

11. sed

cat

When between two pipes, the cat command does nothing (except putting stdin and stdout)

```
root@PRASH-NITRO5:~$ tac count.txt | cat | cat | cat | cat
five
four
three
two
one
root@PRASH-NITRO5:~$
```

tee

The tee filter puts stdin on stdout and also into a file. So tee is almost the same as cat, except that it has two identical outputs.

```
root@PRASH-NITRO5:~$ tac count.txt | tee temp.txt | tac
one
two
three
four
five
root@PRASH-NITRO5:~$ cat temp.txt
five
four
three
two
one
root@PRASH-NITRO5:~$
```

grep

The most common use of grep is to filter lines of text containing (or not containing) a certain string.

```
root@PRASH-NITRO5:~$ cat names.txt
Sachin Tendulkar
```

```
MS Dhoni
Sehwag
Yuvaraj
Suresh Raina
Harbhajan Singh
S Ganguly
root@PRASH-NITRO5:~$
```

Get MS Dhoni from names.txt file

```
root@PRASH-NITRO5:~$ grep Dhoni names.txt
MS Dhoni
root@PRASH-NITRO5:~$
```

grep -i which filters in a case insensitive way.

```
root@PRASH-NITRO5:~$ grep dhoni names.txt
root@PRASH-NITRO5:~$ grep -i dhoni names.txt
MS Dhoni
root@PRASH-NITRO5:~$
```

grep -v which outputs lines not matching the string

```
root@PRASH-NITRO5:~$ grep -v Dhoni names.txt
Sachin Tendulkar
Sehwag
Yuvaraj
Suresh Raina
Harbhajan Singh
S Ganguly

root@PRASH-NITRO5:~$ grep -vi dhoni names.txt
Sachin Tendulkar
Sehwag
Yuvaraj
Suresh Raina
Harbhajan Singh
S Ganguly
root@PRASH-NITRO5:~$
```

grep -A1 one line after the result is also displayed.

```
root@PRASH-NITRO5:~$ grep Dhoni names.txt
MS Dhoni
```

```
root@PRASH-NITRO5:~$ grep -A1 Dhoni names.txt
MS Dhoni
Sehwag
root@PRASH-NITRO5:~$
```

grep -B1 one line after the result is also displayed.

```
root@PRASH-NITRO5:~$ grep Dhoni names.txt
MS Dhoni
root@PRASH-NITRO5:~$ grep -B1 Dhoni names.txt
Sachin Tendulkar
MS Dhoni
root@PRASH-NITRO5:~$
```

grep -C1 (context) one line before and one after are also displayed.

```
root@PRASH-NITRO5:~$ grep Dhoni names.txt
MS Dhoni
root@PRASH-NITRO5:~$ grep -C1 Dhoni names.txt
Sachin Tendulkar
MS Dhoni
Sehwag
root@PRASH-NITRO5:~$
```

Note: A2, B4 or C20 can also be used as options according to the requirement.

cut

The cut command in UNIX is a command for cutting out the sections from each line of files and writing the result to standard output. It can be used to cut parts of a line by **byte position, character and field**.

```
root@PRASH-NITRO5:~$ cat names.txt
Sachin Tendulkar
MS Dhoni
Sehwag
Yuvaraj
Suresh Raina
Harbhajan Singh
S Ganguly
root@PRASH-NITRO5:~$
```

```
root@PRASH-NITRO5:~$ cut -b 1,2,3 names.txt
Sac
MS 
Seh
Yuv
Sur
Har
S G
```

1- indicate from 1st byte to end byte of a line

```
root@PRASH-NITRO5:~$ cut -b 1- names.txt
Sachin Tendulkar
MS Dhoni
Sehwag
Yuvaraj
Suresh Raina
Harbhajan Singh
S Ganguly
```

3- indicate from 1st byte to 3rd byte of a line

```
root@PRASH-NITRO5:~$ cut -b -3 names.txt
Sac
MS 
Seh
Yuv
Sur
Har
S G
root@PRASH-NITRO5:~$
```

tr

You can translate characters with **tr**

Example: Translation of all occurrences of S to s.

```
root@PRASH-NITRO5:~$ cat names.txt
Sachin Tendulkar
MS Dhoni
Sehwag
Yuvaraj
Suresh Raina
```

```
Harbhajan Singh
S Ganguly
root@PRASH-NITRO5:~$ cat names.txt | tr 'S' 's'
sachin Tendulkar
Ms Dhoni
sehwag
Yuvaraj
suresh Raina
Harbhajan singh
s Ganguly
root@PRASH-NITRO5:~$
```

Example: Translate all lowercase to uppercase

```
root@PRASH-NITRO5:~$ cat names.txt
Sachin Tendulkar
MS Dhoni
Sehwag
Yuvaraj
Suresh Raina
Harbhajan Singh
S Ganguly
root@PRASH-NITRO5:~$ cat names.txt | tr [a-z] [A-Z]
SACHIN TENDULKAR
MS DHONI
SEHWAG
YUVARAJ
SURESH RAINA
HARBHAJAN SINGH
S GANGULY
root@PRASH-NITRO5:~$
```

The **tr -s** filter can also be used to squeeze multiple occurrences of a character to one.

```
root@PRASH-NITRO5:~$ nano spaces.txt
root@PRASH-NITRO5:~$ cat spaces.txt
one    two       three
     four      five     six
root@PRASH-NITRO5:~$ cat spaces.txt |  tr -s ' '
one two three
 four five six
```

tr to 'encrypt' texts with rot13.

https://en.wikipedia.org/wiki/ROT13

```
root@PRASH-NITRO5:~$ cat names.txt
Sachin Tendulkar
MS Dhoni
Sehwag
Yuvaraj
Suresh Raina
Harbhajan Singh
S Ganguly
root@g:~$ cat names.txt | tr 'a-z' 'nopqrstuvwxyzabcdefghijklm'
Snpuva Traqhyxne
MS Dubav
Srujnt
Yhinenw
Sherfu Rnvan
Hneounwna Svatu
S Gnathyl
root@PRASH-NITRO5:~$ cat names.txt | tr 'a-z' 'n-za-m'
Snpuva Traqhyxne
MS Dubav
Srujnt
Yhinenw
Sherfu Rnvan
Hneounwna Svatu
S Gnathyl
root@PRASH-NITRO5:~$
```

tr -d to delete characters

```
root@PRASH-NITRO5:~$ cat names.txt | tr -d S
achin Tendulkar
M Dhoni
ehwag
Yuvaraj
uresh Raina
Harbhajan ingh
 Ganguly
root@PRASH-NITRO5:~$
```

wc

Counting words, lines and characters is easy with **wc**.

```
root@PRASH-NITRO5:~$ wc names.txt
 7 12 80 names.txt
root@PRASH-NITRO5:~$
```

sort

The **sort** filter will default to an alphabetical sort.

```
root@PRASH-NITRO5:~$ cat names.txt
Sachin Tendulkar
MS Dhoni
Sehwag
Yuvaraj
Suresh Raina
Harbhajan Singh
S Ganguly
root@PRASH-NITRO5:~$ sort names.txt
Harbhajan Singh
MS Dhoni
S Ganguly
Sachin Tendulkar
Sehwag
Suresh Raina
Yuvaraj
root@PRASH-NITRO5:~$
```

Sorting different columns (column 1 or column 2).

```
root@PRASH-NITRO5:~$ sort -k1 names.txt
Harbhajan Singh
MS Dhoni
S Ganguly
Sachin Tendulkar
Sehwag
Suresh Raina
Yuvaraj
root@PRASH-NITRO5:~$ sort -k2 names.txt
Sehwag
```

```
Yuvaraj
MS Dhoni
S Ganguly
Suresh Raina
Harbhajan Singh
Sachin Tendulkar
root@PRASH-NITRO5:~$
```

uniq

uniq command you can remove duplicates from a sorted list.

```
root@PRASH-NITRO5:~$ cat names.txt
Sehwag
Sachin Tendulkar
MS Dhoni
Sehwag
MS Dhoni
Yuvaraj
Suresh Raina
Harbhajan Singh
S Ganguly
Yuvaraj
root@PRASH-NITRO5:~$ sort names.txt | uniq
Harbhajan Singh
MS Dhoni
S Ganguly
Sachin Tendulkar
Sehwag
Suresh Raina
Yuvaraj
root@PRASH-NITRO5:~$
```

uniq can also count occurrences with the **-c** option.

```
root@PRASH-NITRO5:~$ sort names.txt | uniq -c
      1 Harbhajan Singh
      2 MS Dhoni
      1 S Ganguly
      1 Sachin Tendulkar
      2 Sehwag
      1 Suresh Raina
      2 Yuvaraj
root@PRASH-NITRO5:~$
```

comm

Comparing streams (or files) can be done with the comm.

```
root@PRASH-NITRO5:~$ comm names.txt names2.txt
        Rohith Sharma
Sehwag
comm: file 1 is not in sorted order
Sachin Tendulkar
MS Dhoni
Sehwag
MS Dhoni
        Virat Kohli
comm: file 2 is not in sorted order
        MS Dhoni
        Sehwag
        MS Dhoni
                Yuvaraj
                Suresh Raina
                Harbhajan Singh
                S Ganguly
                Yuvaraj
        Andrew Simonds
root@PRASH-NITRO5:~$
```

od

od command is to show the contents of the file in hexadecimal bytes.

```
root@PRASH-NITRO5:~$ cat temp.txt
five
four
three
two
one
root@PRASH-NITRO5:~$ od temp.txt
0000000 064546 062566 063012 072557 005162 064164 062562 005145
0000020 073564 005157 067157 005145
0000030
```

```
root@PRASH-NITRO5:~$
```

The same file can also be displayed in octal bytes.

```
root@PRASH-NITRO5:~$ od -b temp.txt
0000000 146 151 166 145 012 146 157 165 162 012 164 150 162 145 145 012
0000020 164 167 157 012 157 156 145 012
0000030
root@PRASH-NITRO5:~$
```

The file in **ascii** (or backslash) characters

```
root@PRASH-NITRO5:~$ od -c temp.txt
0000000   f   i   v   e  \n   f   o   u   r  \n   t   h   r   e   e  \n
0000020   t   w   o  \n   o   n   e  \n
0000030
root@PRASH-NITRO5:~$
```

sed

The stream editor sed can perform functions on files like, searching, find and replace, insertion or deletion.

Replacing or substituting string :

sed command is mostly used to replace the text in a file. The below simple sed command replaces the word “five” with “fiftyfive” in the file.

```
root@PRASH-NITRO5:~$ cat temp.txt
five
four
three
two
one
root@PRASH-NITRO5:~$ sed 's/five/fiftyfive/' temp.txt
fiftyfive
four
three
two
one
root@PRASH-NITRO5:~$
```

Replacing string on a specific line number :

You can restrict the sed command to replace the string on a specific line number.

```
root@PRASH-NITRO5:~$ cat temp.txt
five
four
three
two
one
root@PRASH-NITRO5:~$ sed '3 s/three/thirtythree/' temp.txt
five
four
thirtythree
two
one
root@PRASH-NITRO5:~$
```

Soft Links and Hard Links

A link in UNIX is a pointer to a file. Like pointers in any programming languages, links in UNIX are pointers pointing to a file or a directory. Creating links is a kind of shortcut to access a file. Links allow more than one file name to refer to the same file.

There are two types of links :

1. Soft Link or Symbolic links
2. Hard Link

Soft Link (Symbolic Link) :

Let's try to understand the soft link with an example.

Create an empty folder called ‘link’

```
root@PRASH-NITRO5:~/$ mkdir link
root@PRASH-NITRO5:~/$ cd link
```

```
root@PRASH-NITRO5:~/link$ touch 1.txt 2.txt
root@PRASH-NITRO5:~/link$ ls
1.txt  2.txt
root@PRASH-NITRO5:~/link$
```

The 2 new files are created.

Now, create a symbolic or soft link using the following command.

```
root@PRASH-NITRO5:~/link$ ln -s 1.txt 3.txt
root@PRASH-NITRO5:~/link$ ls -l
total 0
-rw-r--r-- 1 root root 0 Jun 24 01:10 1.txt
-rw-r--r-- 1 root root 0 Jun 24 01:10 2.txt
lrwxrwxrwx 1 root root 5 Jun 24 01:12 3.txt -> 1.txt
root@PRASH-NITRO5:~/link$
```

Let's find the inode number and permissions of the files.

```
root@PRASH-NITRO5:~/link$ ls -li
total 0
32932572275321685 -rw-r--r-- 1 root root 0 Jun 24 01:10 1.txt
 6192449488346196 -rw-r--r-- 1 root root 0 Jun 24 01:10 2.txt
13792273859111885 lrwxrwxrwx 1 root root 5 Jun 24 01:12 3.txt -> 1.txt
root@PRASH-NITRO5:~/link$
```

Inode Number (ls -i)

An Inode number is a uniquely existing number for all the files in Linux and all Unix type systems. When a file is created on a system, a file name and Inode number is assigned to it.

```
root@PRASH-NITRO5:~/link$ ln -s 1.txt 4.txt
root@PRASH-NITRO5:~/link$ ls -li
total 0
32932572275321685 -rw-r--r-- 1 root root 0 Jun 24 01:10 1.txt
 6192449488346196 -rw-r--r-- 1 root root 0 Jun 24 01:10 2.txt
13792273859111885 lrwxrwxrwx 1 root root 5 Jun 24 01:12 3.txt -> 1.txt
14355223812265420 lrwxrwxrwx 1 root root 5 Jun 24 01:19 4.txt -> 1.txt
root@PRASH-NITRO5:~/link$
```

As you can see the results of the above **ls -li** commands, the inode numbers and file permissions of soft link and original file are different.

What happens if we delete the original file or source file?

```
root@PRASH-NITRO5:~/link$ rm 1.txt
root@PRASH-NITRO5:~/link$ ls -li
total 0
 6192449488346196 -rw-r--r-- 1 root root 0 Jun 24 01:10 2.txt
13792273859111885 lrwxrwxrwx 1 root root 5 Jun 24 01:12 3.txt -> 1.txt
14355223812265420 lrwxrwxrwx 1 root root 5 Jun 24 01:19 4.txt -> 1.txt
root@PRASH-NITRO5:~/link$
```

Now check the contents in 3.txt and 4.txt

```
root@PRASH-NITRO5:~/link$ cat 3.txt
cat: 3.txt: No such file or directory
root@PRASH-NITRO5:~/link$ cat 4.txt
cat: 4.txt: No such file or directory
root@PRASH-NITRO5:~/link$
```

Conclusion with Soft Links:

- Soft Links are invalid once the source or parent file is deleted.
- Soft link is just a link that points to the original file yet with different inode numbers and permissions.
- The softlink is like a shortcut to a file. If you remove the file, the shortcut is useless.
- If we remove the soft link, the original file will still be available.

HardLink:

Let's try to understand the soft link with an example.

Create an empty folder called **'hardlink'**

```
root@PRASH-NITRO5:~$ mkdir hardlink
root@PRASH-NITRO5:~$ cd hardlink/
```

```
root@PRASH-NITRO5:~/hardlink$ touch file1.txt file2.txt
root@PRASH-NITRO5:~/hardlink$ ls -li
total 0
 3096224744326978 -rw-r--r-- 1 root root 0 Jun 24 01:30 file1.txt
10696049115716710 -rw-r--r-- 1 root root 0 Jun 24 01:30 file2.txt
root@PRASH-NITRO5:~/hardlink$
```

The 2 new files are created.

Now, create a hard link using the following command.

```
root@PRASH-NITRO5:~/hardlink$ ln file1.txt new.txt
root@PRASH-NITRO5:~/hardlink$ ls -li
total 0
3096224744326978 -rw-r--r-- 2 root root 0 Jun 24 01:30 file1.txt
10696049115716710 -rw-r--r-- 1 root root 0 Jun 24 01:30 file2.txt
3096224744326978 -rw-r--r-- 2 root root 0 Jun 24 01:30 new.txt
root@PRASH-NITRO5:~/hardlink$
```

Now let's remove the source file.

```
root@PRASH-NITRO5:~/hardlink$ rm file1.txt
root@PRASH-NITRO5:~/hardlink$ cat new.txt
root@PRASH-NITRO5:~/hardlink$ ls -li
total 0
10696049115716710 -rw-r--r-- 1 root root 0 Jun 24 01:30 file2.txt
3096224744326978 -rw-r--r-- 1 root root 0 Jun 24 01:30 new.txt
root@PRASH-NITRO5:~/hardlink$
```

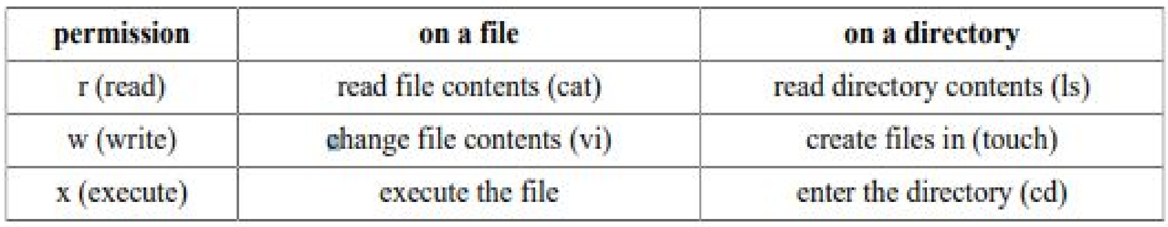
Note: Removing the source file won't affect the hard links.

Conclusion with HardLinks:

- Hard Link(new.txt) and source file (file1.txt) are having the same inode number and permissions.
- If we change the permissions on the source file, the same permission will be applied to the hard link file as well.
- Removing the source file won't affect the hard links because hard links share the same inode number, the permissions and data of the original file.

Linux File Permissions

rwx



permission	on a file	on a directory
r (read)	read file contents (cat)	read directory contents (ls)
w (write)	change file contents (vi)	create files in (touch)
x (execute)	execute the file	enter the directory (cd)

Lets try to check the permissions of the files in a folder.

```
root@PRASH-NITRO5:~/some$ ls -lh
total 0
-rw-r--r-- 1 root root 12 Jun 23 00:26 file.txt
lrwxrwxrwx 1 root root  8 Jun 23 00:28 file2.txt -> file.txt
lrwxrwxrwx 1 root root  8 Jun 23 00:28 file3.txt -> file.txt
lrwxrwxrwx 1 root root  9 Jun 23 00:28 file4.txt -> file3.txt
root@PRASH-NITRO5:~/some$
```

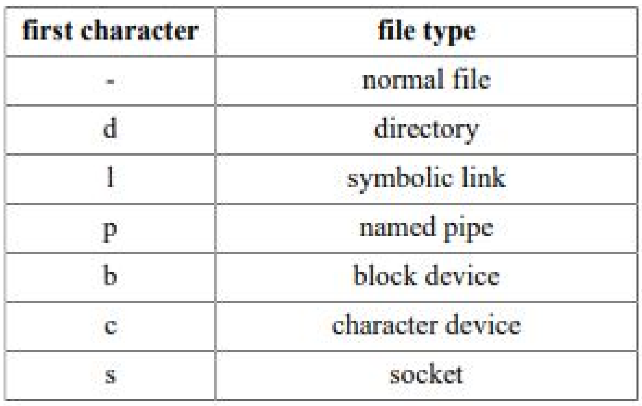
The first column is about file permissions.

Three sets of **rwx**

- The first rwx triplet represents the permissions for the user owner.
- The second triplet corresponds to the group owner; it specifies permissions for all members of that group.
- The third triplet defines permissions for all other users that are not the user owner and are not a member of the group owner

When you use **ls -l**, for each file you can see **ten characters** before the user and group owner.

The first character tells us the type of file.



first character	file type
-	normal file
d	directory
l	symbolic link
p	named pipe
b	block device
c	character device
s	socket

chmod

Setting Permissions of a file is possible with **chmod** command. You need to recall your computer number system (Octal Number System) concepts.

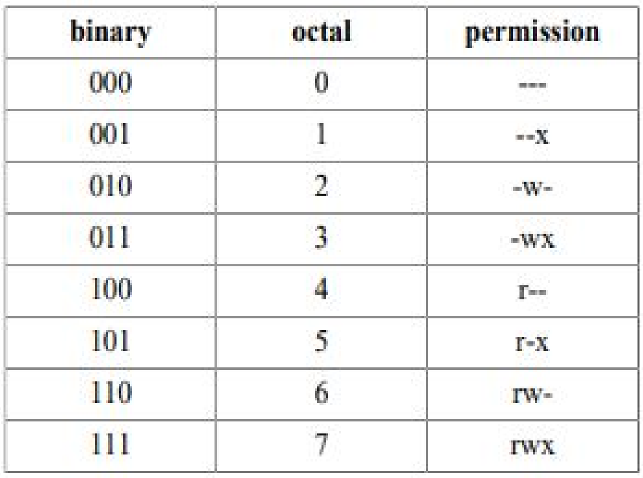
```
root@PRASH-NITRO5:~/some$ ls -lh
total 0
-rw-r--r-- 1 root root 12 Jun 23 00:26 file.txt
lrwxrwxrwx 1 root root  8 Jun 23 00:28 file2.txt -> file.txt
lrwxrwxrwx 1 root root  8 Jun 23 00:28 file3.txt -> file.txt
lrwxrwxrwx 1 root root  9 Jun 23 00:28 file4.txt -> file3.txt
root@PRASH-NITRO5:~/some$ chmod 777 file.txt
root@PRASH-NITRO5:~/some$ ls -lh
total 0
-rwxrwxrwx 1 root root 12 Jun 23 00:26 file.txt
lrwxrwxrwx 1 root root  8 Jun 23 00:28 file2.txt -> file.txt
lrwxrwxrwx 1 root root  8 Jun 23 00:28 file3.txt -> file.txt
lrwxrwxrwx 1 root root  9 Jun 23 00:28 file4.txt -> file3.txt
root@PRASH-NITRO5:~/some$
```

chmod 777 is equal to rwxrwxrwx

7 in binary : 111 => rwx (User)

7 in binary : 111 => rwx (Group)

7 in binary : 111 => rwx (Others)



binary	octal	permission
000	0	---
001	1	--x
010	2	-w-
011	3	-wx
100	4	r--
101	5	r-x
110	6	rw-
111	7	rwx

Exercise 1:

Change the file permission of a file to read-only to user and remove for groups and others.

```
root@PRASH-NITRO5:~/some$ chmod 400 file.txt
root@PRASH-NITRO5:~/some$ ls -lh
total 0
-r-------- 1 root root 12 Jun 23 00:26 file.txt
lrwxrwxrwx 1 root root  8 Jun 23 00:28 file2.txt -> file.txt
lrwxrwxrwx 1 root root  8 Jun 23 00:28 file3.txt -> file.txt
lrwxrwxrwx 1 root root  9 Jun 23 00:28 file4.txt -> file3.txt
root@PRASH-NITRO5:~/some$
```

Exercise 2:

Give the file’s owner read and write permissions and only read permissions to group members and all other users.

```
root@PRASH-NITRO5:~/some$ chmod 644 dirname
```

Exercise 3:

Give the file’s owner read, write and execute permissions, read and execute permissions to group members and no permissions to all other users.

```
root@PRASH-NITRO5:~/some$ chmod 750 dirname
```

Exercise 4:

Give read, write, and execute permissions, and a sticky bit to a given directory.

```
root@PRASH-NITRO5:~/some$ chmod 1777 dirname
```

What is Linux Sticky Bit ?

- You would set the sticky bit primarily on directories in UNIX / Linux.
- If you set the sticky bit to a directory, other users cannot delete or rename the files (or subdirectories) within that directory.
- When the sticky bit is set on a directory, only the owner and the root user can delete / rename the files or directories within that directory.

Exercise 5:

Recursively set read, write, and execute permissions to the file owner and no permissions for all other users on a given directory.

```
root@PRASH-NITRO5:~/some$ chmod -R 700 dirname
```

Command Line Editors

nano

nano is an effective command-line editor. When working on the command line, quite often you will need to create or edit text files. GNU nano is an easy to use command line text editor for Unix and Linux operating systems.

```
root@PRASH-NITRO5:~/some$ nano --version
 GNU nano, version 2.9.3
 (C) 1999-2011, 2013-2018 Free Software Foundation, Inc.
 (C) 2014-2018 the contributors to nano
 Email: nano@nano-editor.org    Web: https://nano-editor.org/
 Compiled options: --disable-libmagic --disable-wrapping-as-root
```

```
--enable-utf8
root@PRASH-NITRO5:~/some$
```

If you do not get the nano version, you need to install nano in your system.

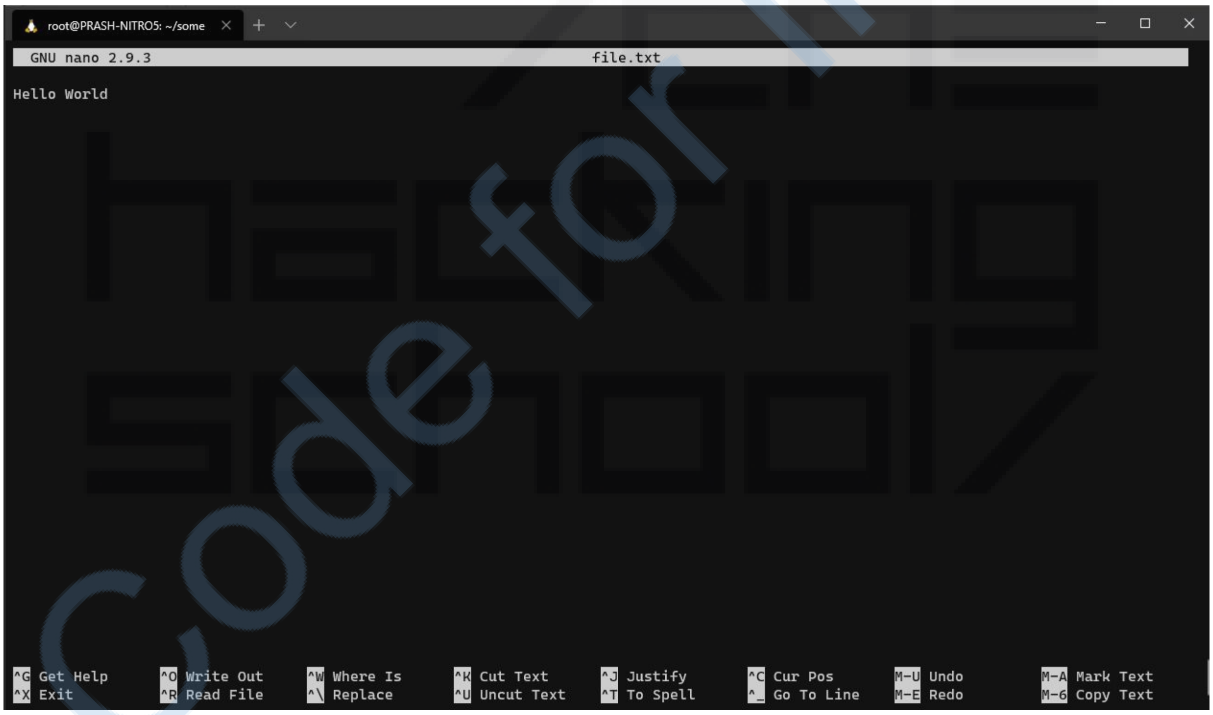
```
root@PRASH-NITRO5:~/$ sudo apt install nano
```

Opening a File/Creating a File

```
root@PRASH-NITRO5:~/$ nano file.txt
```

This opens a new editor window, and you can start editing the file.

At the bottom of the window, there is a list of the most basic command shortcuts to use with the nano editor.



```
  GNU nano 2.9.3                     file.txt

Hello World

^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos    M-U Undo      M-A Mark Text
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text ^T To Spell   ^_ Go To Line M-E Redo      M-6 Copy Text
```

nano command line editor shortcuts :

Copy and Paste

Task	Keystroke
Select a region for a cut or paste operation	**Alt+a** **NOTE:** After setting a mark with Alt+a, move the cursor to define the region, you should see it highlighted as you move the cursor. Also, to cancel the definition of the region just enter Alt+a again.
Copy a highlighted region into the clipboard	**Alt+^**
Cut a highlighted region into the clipboard	**Ctrl+k**
Paste the contents of the clipboard at the current cursor position	**Ctrl+u**
Cut from the current cursor position to the end-of-line (EOL)	**Ctrl+k** **NOTE:** This command doesn't require highlighting of the region.

Search and Replace

Task	Keystroke
Search for a target string	**Ctrl+w** **NOTE:** Once this command has been entered, notice the new menu items at the bottom of the screen, such as toggling the direction of the search (Alt+B) or replacing the search string with a different string (Ctrl+R)
Repeat the last search	**Alt+w**
Toggle direction for next search	**Ctrl+w** followed by **Ctrl+b**
Search and replace	**Alt+r**

Frequent Key Shortcuts

Task	Keystroke
Home	**Ctrl+a**
End	**Ctrl+e**
Page Up	**Ctrl+y**
Page Down	**Ctrl+v**
Arrow Keys	**Ctrl+f** (right), **Ctrl+b** (left), **Ctrl+n** (down), **Ctrl+p** (up)
Tab	**Ctrl+i**
Backspace	**Ctrl+h**
Delete	**Ctrl+d**
Return	**Ctrl+m**
Save	**Ctrl + o**

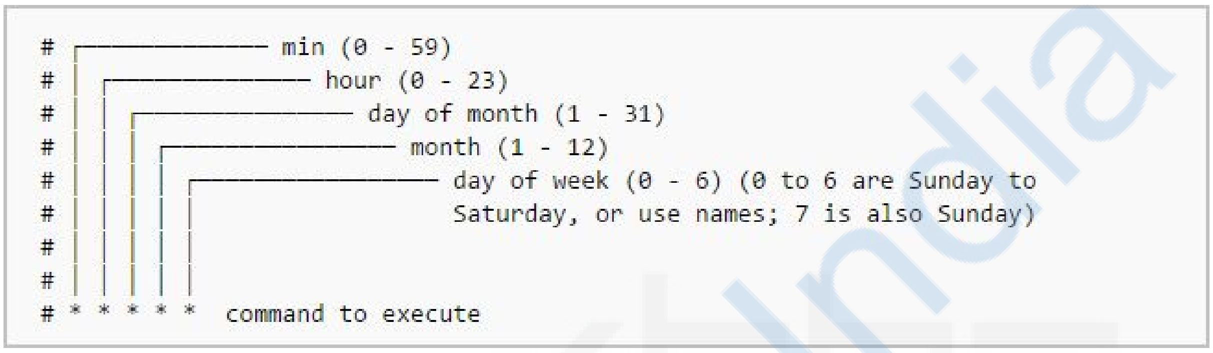
Cron Jobs

The **crontab** software utility is a time-based job scheduler in Unix-like operating systems. Cron allows Linux and Unix users to run commands or scripts at a given time and date. Once can schedule scripts to be executed periodically. It is usually used for system admin jobs such as backups or cleaning/tmp/ directories and more.

Crontab Syntax

Linux crontab has six fields. 1-5 fields define the date and time of execution. The 6'th fields are used for command or script to be executed.The Linux crontab syntax are as following:

- Asterisk (*) – Matches anything
- Define range – You can define range using the hyphen like: 1-10 or 20-30 or sun-fri or feb-apr
- Define multiple range – You can define multiple ranges with command separated like: jan-mar,jul-sep



```
# ┌───────────── min (0 - 59)
# │ ┌────────────── hour (0 - 23)
# │ │ ┌─────────────── day of month (1 - 31)
# │ │ │ ┌──────────────── month (1 - 12)
# │ │ │ │ ┌───────────────── day of week (0 - 6) (0 to 6 are Sunday to
# │ │ │ │ │                  Saturday, or use names; 7 is also Sunday)
# │ │ │ │ │
# │ │ │ │ │
# * * * * *  command to execute
```

Exercise 1 : Schedule a script that writes 'Hello World' every minute to a text file.

Step 1: Create a .sh script file that prints 'Hello World'

```
prash@ubuntu:~$ touch script.sh
prash@ubuntu:~$ nano script.sh
```

script.sh code:

```
echo "Hello World" >> ~/output.txt
```

Test script.sh

```
prash@ubuntu:~$ bash script.sh
prash@ubuntu:~$ cat output.txt
Hello World
```

Now that script.sh is working, let's schedule a cron job that executes the above script every minute.

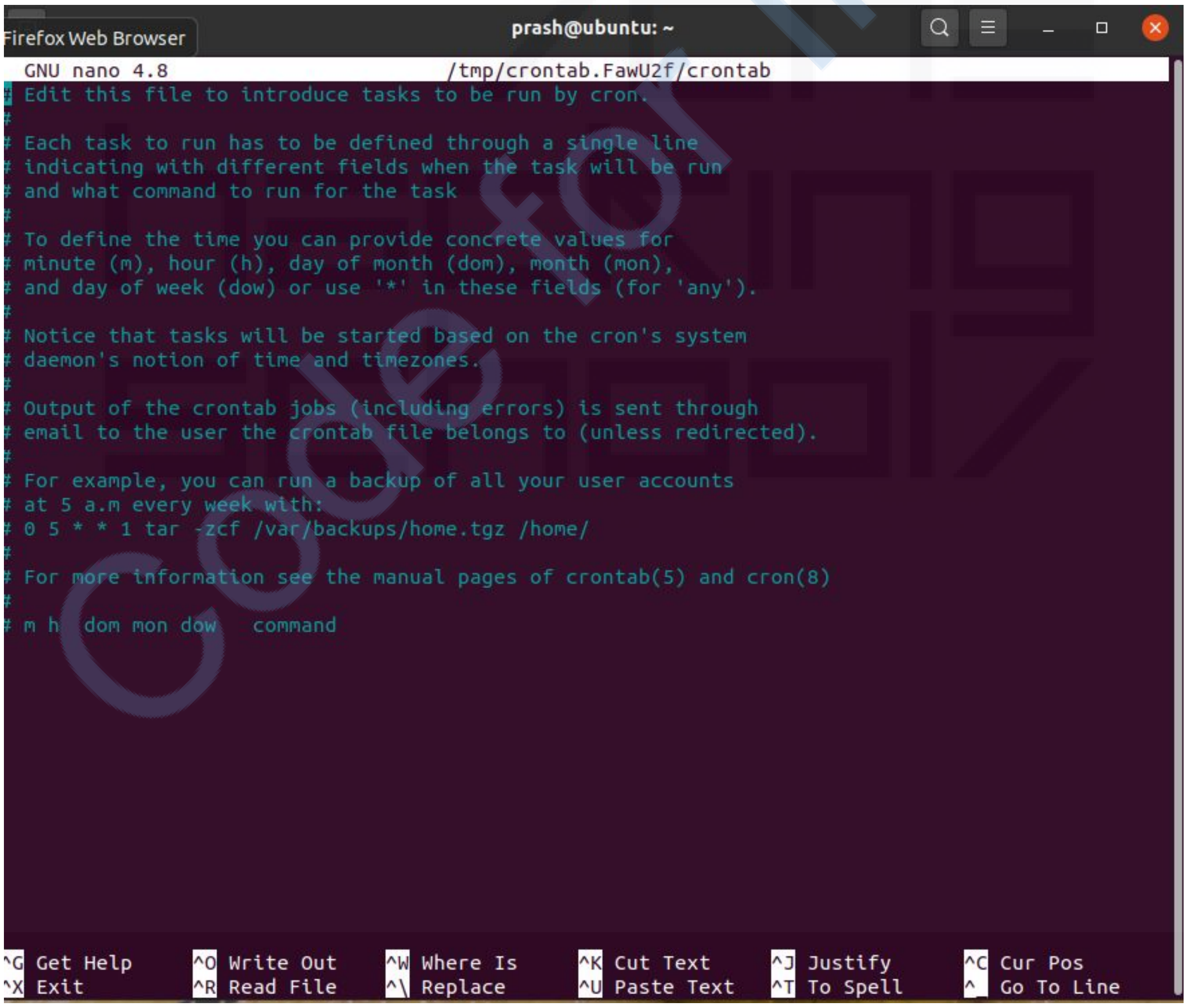
Step 2: Configure Cron Job

crontab

```
prash@ubuntu:~$ crontab -e
no crontab for prash - using an empty one

Select an editor.  To change later, run 'select-editor'.
  1. /bin/nano        <---- easiest
  2. /usr/bin/vim.tiny
  3. /bin/ed

Choose 1-3 [1]:
```



```
  GNU nano 4.8                    /tmp/crontab.FawU2f/crontab
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
```

Crontab syntax

```
* * * * * /home/prash/script.sh
```

The above syntax will run the script.sh every minute that will write ‘Hello World’ to the output.txt file.

Result :

```
prash@ubuntu:~$ ls -l
total 40
drwxr-xr-x 3 prash prash 4096 May  7 01:55 Desktop
drwxr-xr-x 2 prash prash 4096 May  7 01:50 Documents
drwxr-xr-x 2 prash prash 4096 May  7 01:50 Downloads
drwxr-xr-x 2 prash prash 4096 May  7 01:50 Music
-rw-rw-r-- 1 prash prash   24 Jun 24 05:52 output.txt
drwxr-xr-x 2 prash prash 4096 May  7 01:50 Pictures
drwxr-xr-x 2 prash prash 4096 May  7 01:50 Public
-rwxrwxrwx 1 prash prash   35 Jun 24 05:46 script.sh
drwxr-xr-x 2 prash prash 4096 May  7 01:50 Templates
drwxr-xr-x 2 prash prash 4096 May  7 01:50 Videos
prash@ubuntu:~$ cat output.txt
Hello World
Hello World
```

Note:

- crontab works and will be installed by default in Ubuntu distro.
- crontab works in WSl. Follow this additional procedure if your configuration is WSL Linux

https://stackoverflow.com/questions/41281112/crontab-not-working-with-bash-on-ubuntu-on-windows

Few CronTab Examples :

Example 1: Schedule tasks to execute on hourly (@hourly).

```
@hourly /path/to/script.sh
```

Example 2: Schedule a cron to execute on every 30 Seconds.

To schedule a task to execute every 30 seconds is not possible by time parameters, But it can be done by scheduling the same cron twice as below.

```
* * * * * /scripts/script.sh
* * * * *  sleep 30; /path/to/script.sh
```

Example 3: Schedule a cron to execute twice on every Sunday and Monday.

To schedule a task to execute twice on Sunday and Monday only. Use the following settings to do it.

```
0 4,17 * * sun,mon /path/to/script.sh
```

Example 4: Schedule a cron to execute on every Sunday at 5 PM.

```
0 17 * * sun  /path/to/script.sh
```

Exercise 2 : Help Miss Scottie, A system secretary , to schedule a script that auto backups all the files in her work directory in .zip format every 4 hours.

Work Directory path : **~/logs**

logs folder contains : **log1.txt log2.txt log3.txt** files. Assume these 3 files data will be updated by an external program.

Auto Backup every 4 hours all the files in ~/logs folder to ~/backup folder with zip filename as the current timestamp.

Solution :

script.sh

```
zip -r "~/backup/archive-$(date +"%Y-%m-%d %H-%M-%S").zip" ~/logs/*
```

cron job syntax :

```
0 */4 * * * /path/to/script.sh
```