LINUX ESSENTIALS

A BOOK THAT EVERY LINUX BEGINNER SHOULD READ

PALANI KARTHIKEYAN

www.krosum.com

CONTENTS

PREFACE

During my college days, as a young aspiring graduate from computer science department, I used to practice certain programming languages, all on the top of either UNIX or Linux.

At that time my focus was fully on languages and i never wanted to see how my code runs or even who works it out. Later, when I joined an organization all my applications written in fabulous languages stared at me for automation. It triggered me to work on a few command line tasks. Though I am good in C and C++, I found it a bit difficult to write shell scripts especially for searching and filtering kind of task.

Then I realized the fact that Linux Commands are the core part of Shell scripts and that it is impossible to write an optimized automation code without commanding knowledge. That is where I started learning Linux system programming, followed by shell, Perl, python and ruby, all on the top of Linux.

Now with my experience as a corporate trainer *for above 16+ years* I wanted *to share my knowledge on Linux Com*mands in a book. This Book will give you a clear start on Linux and you can keep up with further steps in learning sed, awk and bash scripts to enhance your career.

If you are a beginner or if you want to master the fundamentals of Linux, this book is for YOU.

DEDICATION

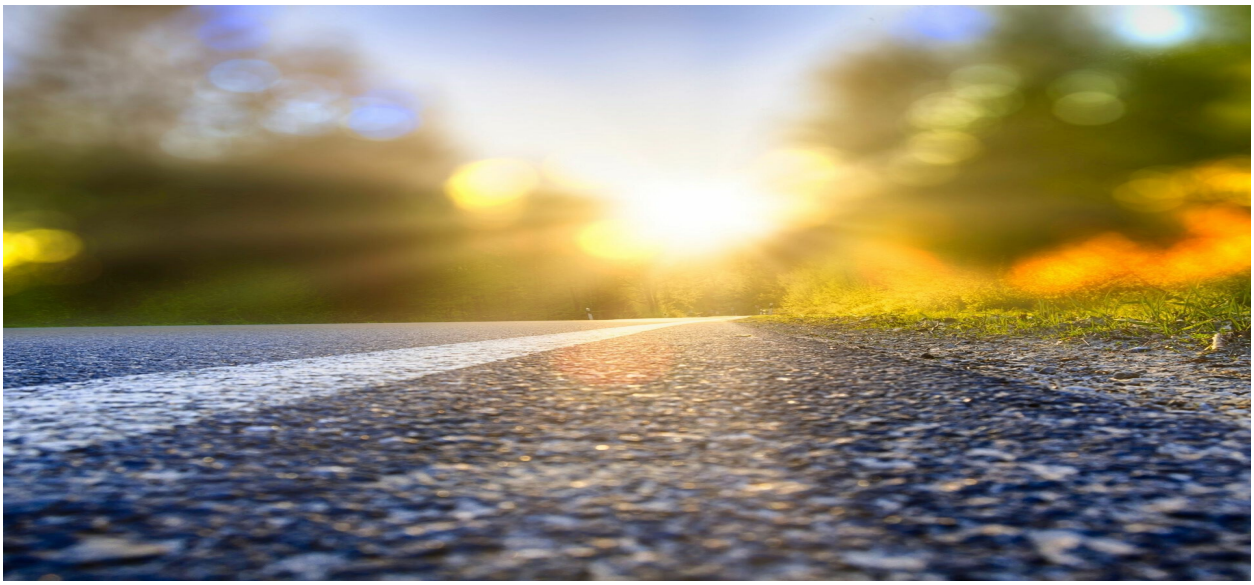
*To my beloved **LORD***

ACKNOWLEDGEMENT

*My heartfelt gratitude to my uncle **Mr.Rajaram** for his dedicated support to complete this book.*

*My sincere thanks to **Mr.Aravind, Mr.JohnPaul, Mr.Kamal, Mr.Yuvraj** and **Mr.Richard** for the able support and acknowledgement given for the perfection of this book.*

*My Love and gratitude to my beloved wife **Theeba Karthikeyan** for her motivation, effort and encouragement for this book.*



CHAPTER 1 - LINUX INTRODUCTION

What is Linux?

Linux is an operating system developed by **Linus Torvalds** in the year 1991.

Generally operating system is defined as a program, *an interface between user applications to the hardware*. This Operating system is responsible to connect system resources to the user. So which part of OS does this task? The ***Kernel,*** the core component of operating system does this task.

Whenever a system starts, once the boot loader stage is done successfully, the Kernel is the first program that is loaded. If boot loader gets failed, then kernel loading is also failed and hence we have to troubleshoot the boot loader issues. Only then the kernel will be loaded.

Once kernel is loaded, it remains in the memory until the Operating

System shut-downs because the Kernel should handle the rest of the thing of the system for the Operating System. We will discuss about boot loader sequence in our following topics.

GNU Project

GNU/Linux – (GNUs Not UNIX)is a term promoted by the Free Software Foundation (FSF) and it is founded by ***Richard Stallman,.***

This project had begun in 1984 to develop a free operating system.

The motto of the project was to develop a free Unix-Compatible operating system. The Linux kernel was added in 1992, achieving the GNU Project's goal of developing a free operating system.

GPL

Linux and GNU software are distributed under the terms of the GNU General Public License (GPL, www.gnu.org/licenses/licenses.html).

The GPL says *we have the right to copy, modify, and redistribute the code covered by the agreement.*

The GPL provides some basic software freedoms

- To use the software for any purpose
- To share the software
- To change the software to suit your needs
- To share the changes that you make.

Why do we use Linux?

Linux has evolved into one of the most promising platform in today's world.

Linux is also *distributed* as an *open source.*

Here we have two terms- ***Open Source*** and ***Distributed***. Let's see what these actually mean here.

Open source follows these *key features.*

- The *freedom to run* the program, for any purpose.
- The *freedom to study* how the program works, and change it in

order to make it do what you wish.

- The *freedom to redistribute* copies so you can help your neighbor.
- The *freedom to distribute copies of your modified versions* to others.

What is distribution?

A ***Linux distribution*** is also referred as ***distro.*** Actually, Linux isn't a complete operating system — it's just a kernel. Linux distributions take the Linux kernel and combine it with other free software to create complete packages like some proprietary software. There are many different Linux distributions available.

If we want to "install Linux," we need to choose a distribution. For example: *Linux mint, Ubuntu, redhat, openSUSE* etc.,

Some popular Linux distributions include:

- Debian
- Fedora
- Mandriva Linux
- OpenSUSE
- Gentoo

- Slackware

Each distribution has a different take on the desktop, the software package management repository and the software package management utility commands.

For example, to install or update a package in RHL (Red-Hat Linux) distribution, we use the following command: *yum install <package name>*

Ex 1: To install Python3 in RHL based distribution,

yum install python3

If we want to install the same package in Debian distribution Linux, we can use apt-*get install <packagename>command*

Ex 2: To install Python3 in Debian based distribution,

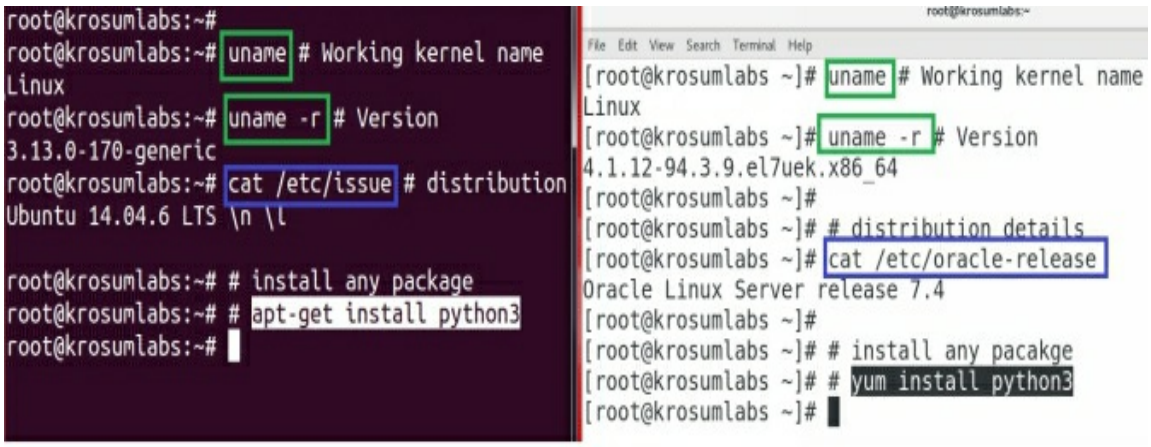
apt-get installpython3

Note that, in both the examples, package name is python3 and working kernel is Linux but distributors are different. The Complete Linux OS means the *combination of Linux kernel and distribution utilities.*

Hope you now understand that the Linux distributions differ by its distribution utilities. Now let's see what this Linux Kernel is.

Linux Kernel

Operating system (OS) is a theoretical term,



Kernel is an implementation structured program; the kernel also has control over everything in the system.

Note that in the above snap, ***uname*** is a command to display our working kernel name. Both the above pictures show the terminal display of Linux but just compare its distribution name and version details- Left side snap shows Ubuntu Linux. The other one is oracle Linux in RHL standard distribution Linux.

In Ubuntu Linux software package management, command line utility is called ***apt-get*** whereas in RHL based Linux package management, command line is **yum** utility. Thus both Linux will install python3 packages in their own way.

My *Ubuntu version* is *14.04.6* (which is the distribution version and not kernel version) whereas my *Oracle Linux version* is *7.4*.

In both the OS, to get working kernel name, type ***uname*** command and to get released the version of kernel, type ***uname–r*** command in command line.

Note kernel is core part of the operating system. Linus Torvalds created only the kernel which is present in all distribution. So in Linux of any distribution, system related commands are same.

So if we want to test our working kernel &version, just type ***uname*** and ***uname –r*** commands in command line. This helps us to know which distribution of Linux we are using. To get distribution details, read configuration file ***/etc/redhat-release***, ***/etc/fedora-release*** and for *Debian based distribution*: linux ***/etc/issue*** file.

So now we are clear with Linux terms such as kernel, distribution etc. We will discuss Linux commands and other details in command topics.

Note: There is single space between command and its option (***uname-r***)

Linux kernel Development model

The heart of the Linux operating system is the kernel.

A lot of developers, representing hundreds of corporates are providing frequent release of the Linux kernel.

The Linux community collaborates through various mailing lists that are set up to handle kernel development.

Features are pushed upstream, through these mail list and Internet Relay Chat (IRC).Upstream is the term used for a community-owned version of a specific project.

This is where the development happens and it always has the most recent changes.

Linus Torvalds leads a team that releases new versions called "vanilla" or "mainline" kernels.

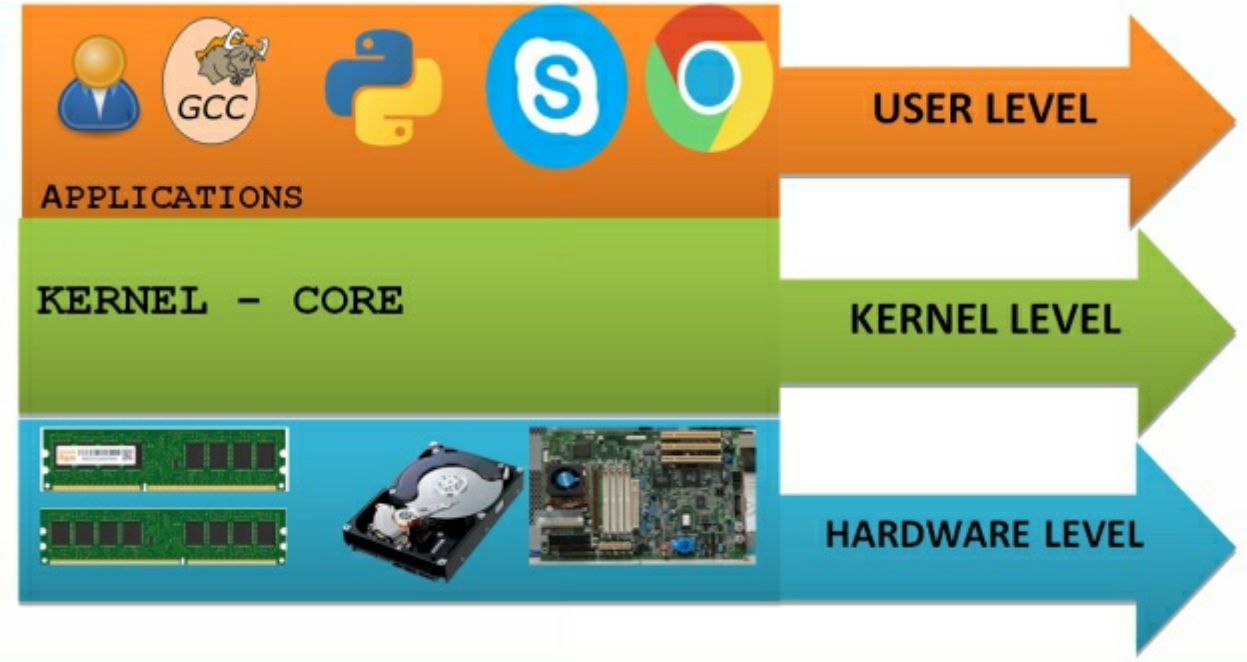
The mainline branch of development incorporates new features, security fixes, and bug fixes. It is not considered a stable branch until it undergoes through testing.

A number of kernel versions are currently being maintained as stable kernels. These kernels have patches that are back ported to them.

Theses patches are primarily driver updates and security fixes.

Kernel branches are available at http://www.kernel.org

Architecture of Linux



In General, System consists of 3 layers**.** See the above diagram. The layer1 is called ***user level*** that holds information about the application created or executed by the user. All application at the user level is called *user process*.

The layer2 is called ***kernel level.*** We know what kernel is. Here the main functionality of kernel is to extend the user requests to hardware which tells about the application that we want to execute on the hardware (CPU) or about the file content that we want to read from storage (Hard disk). To do all these tasks, we need the help of Kernel. Here kernel job is to interface our (user) requests to corresponding hardware resources (CPU, Hard-Disk).

The bottom layer is called *hardware* ***level***. All peripheral devices are connected in this layer.

From this structure we can understand that as a user, we can't talk to hardware directly; so we need a kernel to interface. Now recap our operating

system definition *“OS is a system program, it will interface the user to hardware.”*.

Now we can explore more details about Kernel Architecture, middle layer (kernel level) in the above diagram.

Kernel Exploration

The Linux kernel is a modular designed kernel.

At the architecture level, the Linux kernel interacts with the hardware, controls and schedules the access to resources (CPU, memory, storage, network and so on) on behalf of the applications.

Applications run in what is called the user space and a call is made to a stable set of system libraries to ask for kernel service.

This modular designed kernel allows components of Linux to originate from different developers, each of which had their own specific design goals in mind.

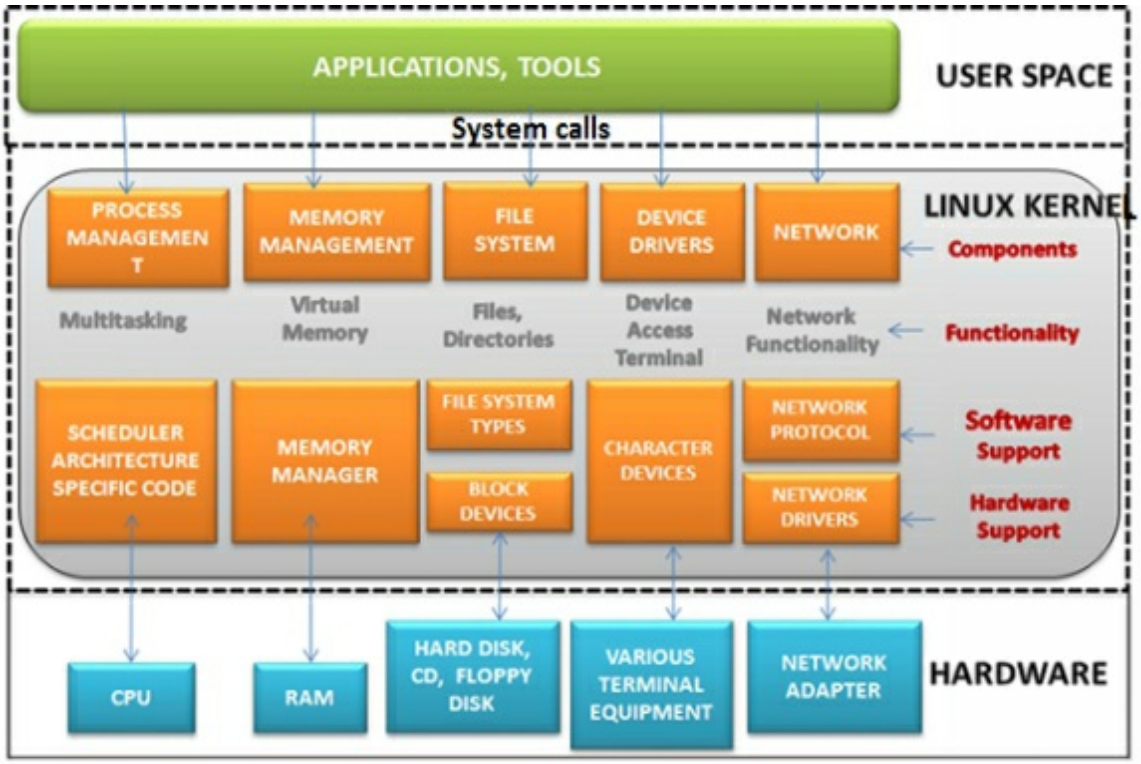
A modular design also means that the Linux kernel is independent of applications and interfaces.

The result is that even when application crashes, all security vulnerabilities in applications tend to remain isolated, rather than affecting the system as a whole.

In Linux, each component is configured separately, typically by using text-based configuration files.

Reading and writing configuration information can be done by scripts or applications by using simple text parsing engines.

No special application programming interface (API) is required to interface with the system configuration data.



LINUX KERNEL STRUCTURE

The above figure shows the detailed architecture of Linux kernel. Unix and Linux are *monolithic* kernel. So, what is a monolithic kernel?

Monolithic kernel is a single large process, running entirely in a single address space. It is a single static object file. All kernel services exist and execute in the kernel address space. The kernel can invoke functions directly. In Monolithic Kernel**,**all the parts of a kernel like **the Scheduler, File System, Memory Management, Networking Stacks, Device Drivers**, etc., are maintained *as a single unit* within the kernel**.**

Components of the System

The main components of Linux operating system are

1. **Command utility**

2. **Shell**
3. **Kernel**

Command utilities

Command utilities are binary files in Linux. As an end user, we will get in to the system with the help of commands. But, how the command gets in to the system? What is the interface name? **The interface name is called shell**.

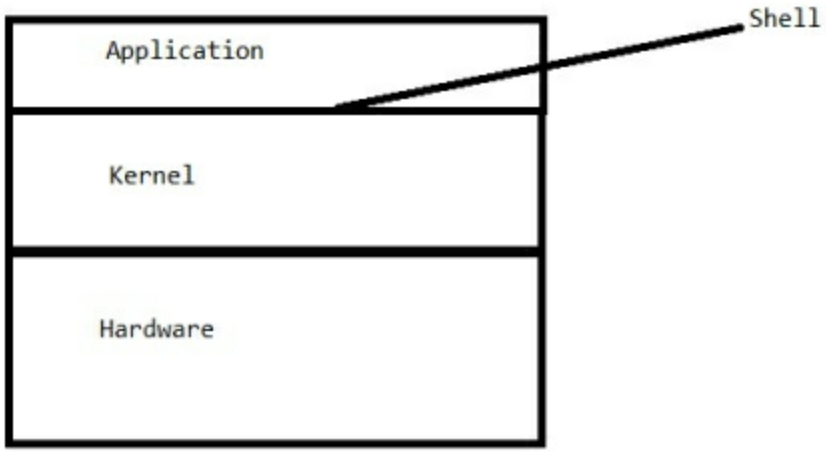
Yes, ***shell*** *is an interface between user and kernel*, the job of shell is interpretation. Each input command will be interpreted (translated) to kernel.

The ***kernel*** is a core important program and the job of the kernel is to interface your instruction (command) to hardware units.

In this entire tutorial book, we will discuss topics such as - *what are Linux command line utilities? how to use Linux command? And how to explore more commands and additional options with real system?*. Note that we won't discuss any programming topics here.

Shell

What is shell? Shell is an interpreter; it interfaces between user space and kernel space.



Theoretically we say that, user interacts with operating system. But how? We are the user and now, how are we going to run our commands on OS?

Let's say, *how to start MySQL database? How to execute (or) run my java program in Linux OS? How to display today date and time?* To do all these tasks, we need an interface and that interface is called shell. So Shell is interface between user and kernel. There are many types of shells in Linux; we will discuss more about shell in next chapter.

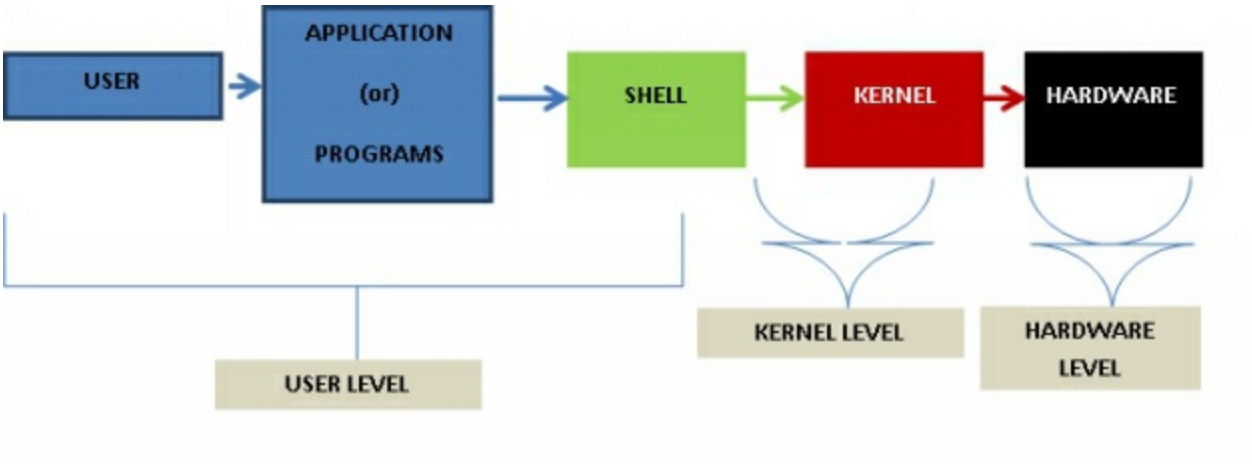
Kernel

Recap the previous chapter; we discussed - *what is an operating system? What is a kernel?*

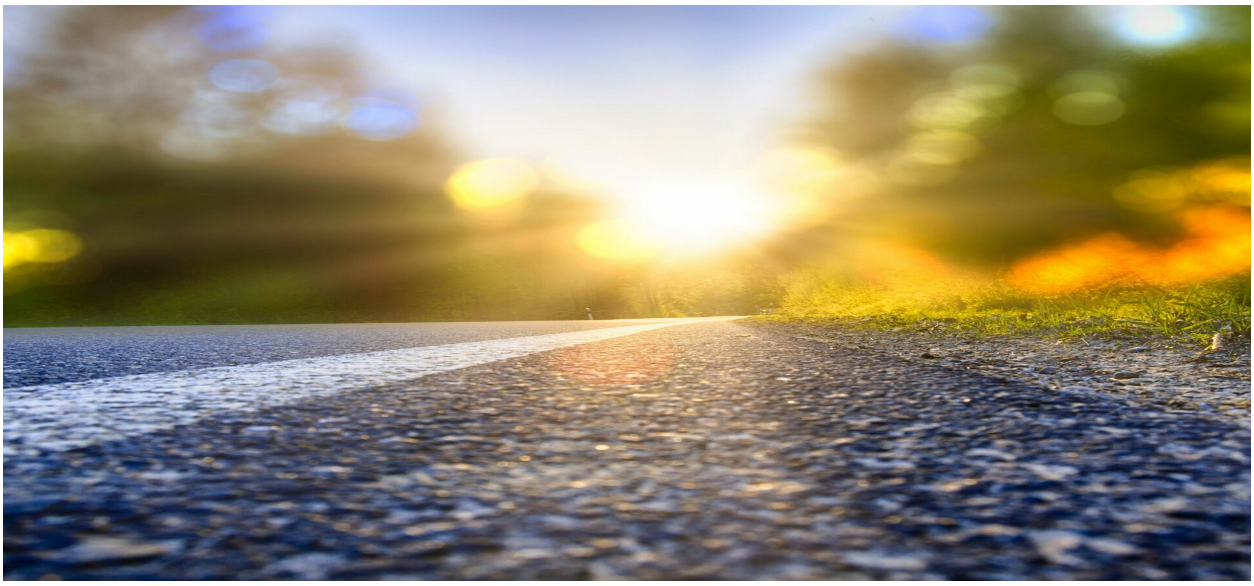
Linux is a kernel but when it first came out, it was called GNU/Linux .

However, the naming convention has its own way. People are more familiar with Linux, and by saying Linux, they refer to the whole operating system. Another unique case is Android. Android uses Linux as its kernel, but people call it Android. Nobody says Android Linux. Linux is a true UNIX kernel. When we say OS services, it is actually kernel services and that is why kernel is called as the heart of an operating system.

Now we understand how our modern operating systemworks.



In next chapter, we will discuss about shell and shell types followed by more exploration on Linux commands.



CHAPTER 2 - LINUX BOOT PROCESS

Step 1- BIOS (Basic Input Output System)

Once the Personal Computer (PC) is switched on, BIOS instruction will start its part. The BIOS is used to perform hardware initialization during the booting process. The main job assigned to BIOS is **POST (Power On Self-Test).** It is hardware self-testing and checking.

The two main errors that occur during POST are:

1. *Fatal error* - This occurs due to hardware problems.
2. *Non-fatal error* - This occurs due to software problems.

Main responsibilities of BIOS during POST are listed below:

1. Verify CPU registers.

2. Verify the integrity of the BIOS code itself.
3. Verify some basic components like DMA, timer, and interrupt controller.
4. Find, size, and verify the system main memory.
5. Initialize BIOS
6. Identify, organize, and select which devices are available for booting.

The *beep sound* after the POST indicates its result. A *single short beep* while restart/start indicates *normal POST* i.e. the system is OK. Two short *beeps* indicate a *POST error* and the error code is shown on screen.

BIOS *act as an intermediary between computer CPU and Input/output devices.* This eliminates the intervention of the operating system and software. The system/server is always aware of the details of hardware and other I/O devices. If any hard disks or I/O devices is changed, the BIOS also needs to be updated.

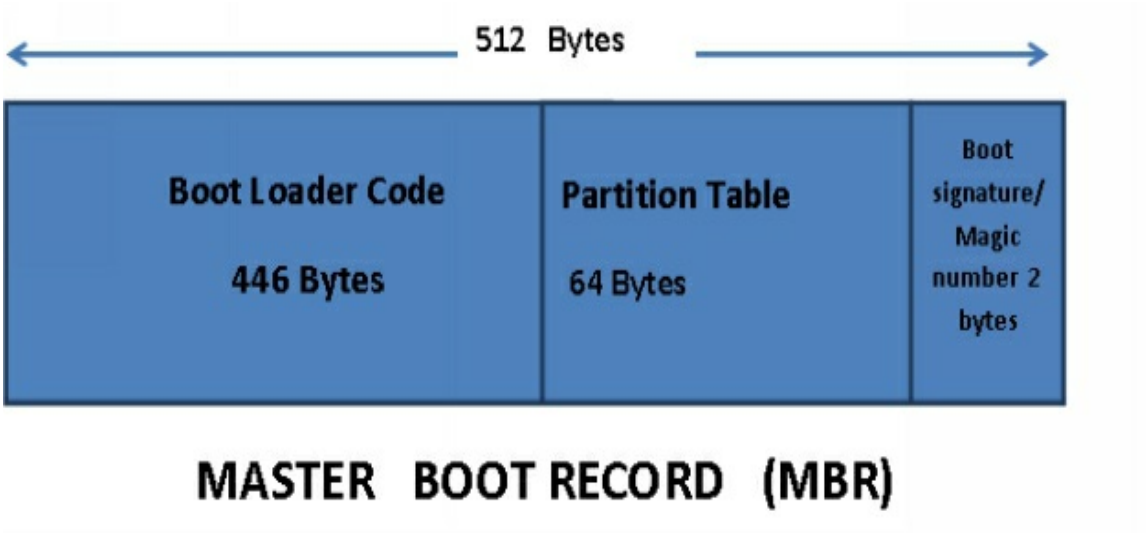
BIOS is stored in **EEPROM** (Electrically Erasable Programmable ROM) / Flash memory. BIOS cannot be stored on a hard disk or other devices because it is the one which manages those devices.

BIOS is written in assembly language. After testing the system hardware and its components, it loads a *Master Boot Record.* It is also called as *Master Boot Loader.*

Step 2 - (MBR) Master Boot Record

What is Master Boot Record (MBR)?

Master Boot Record is the first place where boot loader begins to start. MBR is a 512 byte sector located in the first sector of hard disk. MBR contains both program code and partition table details.



On a computer with x86 architecture, the Master Boot Record (MBR) is the first 512 bytes of the boot drive that is read into memory by the BIOS.

The *first 446 bytes* out of 512 bytes contain *low-level boot code*. For some boot loaders, the code in the MBR points to further boot loader code stored somewhere else on the disk or on another disk.

The *next 64 bytes* contain the *partition table for the disk* where file system utilities like **fdisk, cfdisk, parted**etc., are placed and using these utilities, we can get partition table information.

The *last two bytes* are the *boot signature*, which is used for error detection.

Step 3 - Boot Loader

So far we are aware that when POST is done successfully, the BIOS will execute MBR code. MBR code contains information about boot loader and this MBR executes the boot loader. So let us now see what this boot loader is?

The boot loader software runs when a computer starts. It is responsible for loading and transferring control to the kernel.

It is located in the 1st sector of the bootable disk. (**/dev/sda(or) /dev/had (or) /dev/xvda**)

Boot loader is kept in separate partition under file system; it will mount from /boot partition.

The most common boot loaders for Linux are:

- **LILO**(LInuxLOader)
- **GRUB** (GRand Unified Boot loader).
- **GRUB2** stands for "GRand Unified Boot loader, version 2"

In this chapter we will discuss about **grub2** boot loader.

GRUB2 which stands for "GRand Unified Boot loader, version 2" is a primary boot loader in recent distribution of Linux such as RHL7, Oracle Linux7, ubuntu16 etc.,

GRUB2 is also a program. Once GRUB2 is loaded into RAM, it searches for the location of Kernel. This is the stage where it loads other required drives and kernel modules.

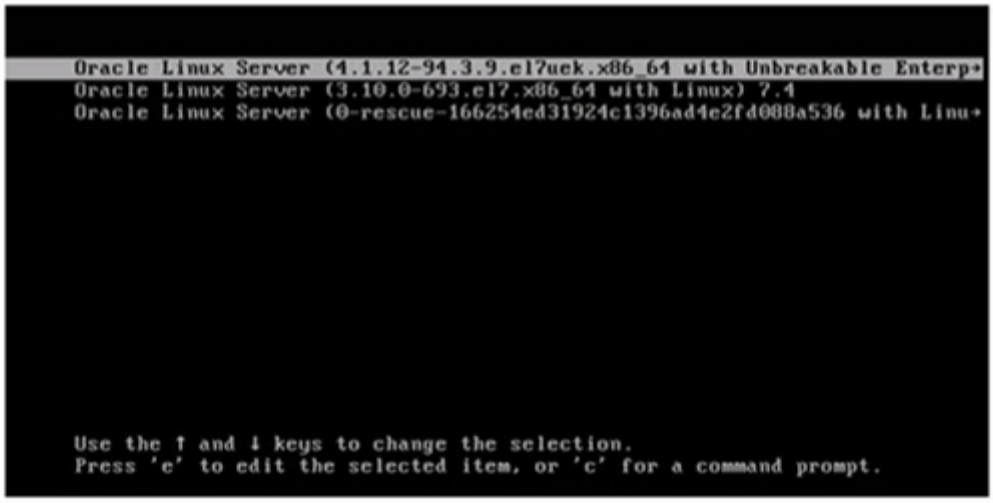
GRUB2 understands file systems and kernel executable formats.

GRUB2 inspects the map file to find the kernel image that is located under /boot.

GRUB2 loads the kernel (vmlinuz-version) from **/boot**partition

If we have multiple kernel images installed on our system, we can choose which one to be executed. We can select our choice based on cursor keys from keyboard.

GRUB 2 Menu



Once we select our desired kernel to load then the kernel stage begins.

Kernel is in compressed format. The selected kernel is now loaded into the memory.

An image file, containing the basic root file systems with all kernel modules, is then loaded into the memory. This image file is located under **/boot** and it is known as initramfs.

Initramfs, abbreviated from “initial RAM file system”, is the successor of initrd “initial ramdisk”.

This image file contains the initial file system. The GRUB starts the kernel and tells the memory address of this image file.

The kernel then mounts this image file as a starter memory based root file system.

The kernel then starts to detect the hardware of the system.

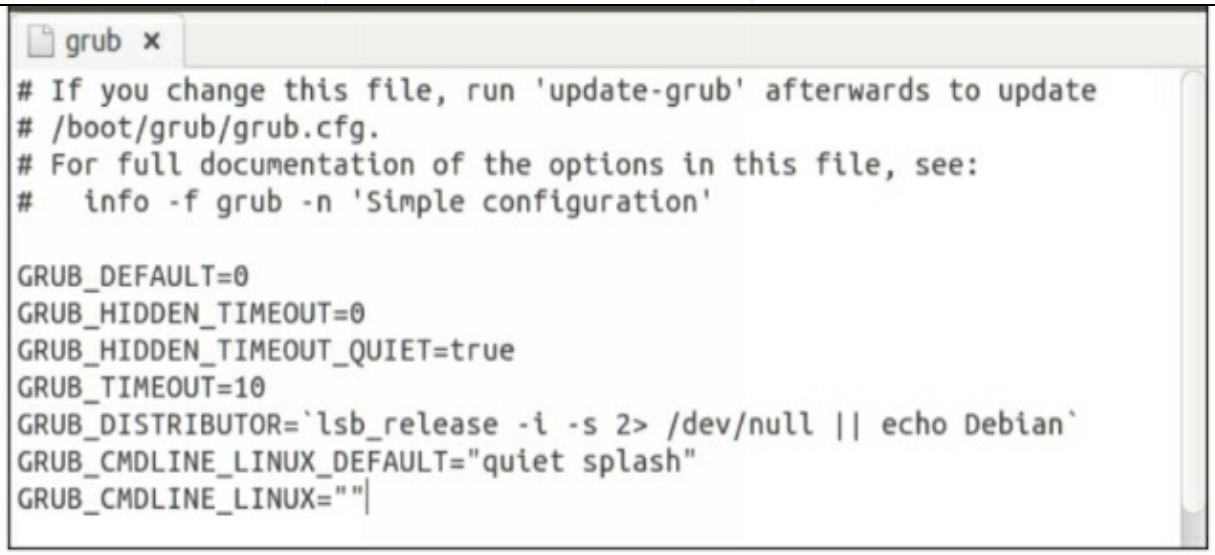
The boot process then starts **init (or) systemd** and other software daemons. Init or systemd (systemdaemon manager) is the parent of all the process. Process ID (PID) of **init (or) system** is 1 depending upon the working Linux distribution.

Let us *display the process name.*

ps–e|grepinit (or) ps–e|grepsystemd

The Differences between grub and grub2 boot loader are as follows.

SPECIFICATION	GRUB	GRUB2
FULL FORM	GRand Unified Bootloader	GRand Unified Bootloader, version 2
USED IN	RHL5,DEB14,Oracle Linux 5,6	RHL7,DEB16,Oracle Linux 7
CONFIGURATION FILE	**/boot/grub/grub.conf**	**/boot/grub2/grub.cfg**
CONFIGURATION FILE CREATED BY	anaconda installer program.	**grub2-mkconfig** template
Is it editable?	Grub configuration file is editable by root user	Grub2 configuration file is **NOT** editable.
Is it a script file?	Grub configuration file is **NOT** shell script	Grub2 configuration file is a shell scri pt file.



```
root@krosumlabs:~# cat /boot/grub/grub.cfg
#
# DO NOT EDIT THIS FILE
#
# It is automatically generated by grub-mkconfig using templates
# from /etc/grub.d and settings from /etc/default/grub
#

### BEGIN /etc/grub.d/00_header ###
if [ -s $prefix/grubenv ]; then
  set have_grubenv=true
  load_env
fi
if [ "${next_entry}" ] ; then
   set default="${next_entry}"
   set next_entry=
   save_env next_entry
   set boot_once=true
else
   set default="0"
fi

if [ x"${feature_menuentry_id}" = xy ]; then
  menuentry_id_option="--id"
else
  menuentry_id_option=""
fi

export menuentry_id_option

if [ "${prev_saved_entry}" ]; then
  set saved_entry="${prev_saved_entry}"
```

File : /boot/grub/grub.conf

File: /boot/grub2/grub.cfg

init process

Once boot process is done successfully, the next step is kernel initialization.

So what is Kernel initialization?

The kernel will initialize CPU components,(Example: MMU,process scheduler) and mount the root file system in read/write mode and finally starts the init(initialization) process (/sbin/init) .

init is a *parent of all the process*; init*process id is 1* (PID value is 1).

init is the first process which loads all the daemons and mounts all the

partitions which are listed under /etc/fstab

init identifies the *default initlevelfrom /etc/inittab*and uses that to load all the appropriate program.

Let's have a look at the /etc/inittabfile to decide the Linux run level.

Following are the available run levels

0 – halt

1 – Single user mode

2 – Multiuser, without NFS

3 – Full multiuser mode

4 – unused

5 – X11

6 – reboot

The kernel starts the /sbin/initprocess with a process ID of 1 (PID 1) and if we want to start/ stop any daemon services, we can use an administrative command called ***service*** command.

systemD introduction

On RHL7/ Ubuntu 16,

The boot loader loads the **vmlinuz**kernel image file into memory and extracts the contents of the **initramfs** image file into a temporary, memory-based file system (tmpfs).

The initial RAM disk (initramfs) is an *initial root file system* that is mounted before the real root file system.

After the newly loaded kernel gets far enough in its initialization sequence (disks probed, memory mapped, and so on), it then switches

over, using the *real root file system* as specified by the root directive *in the GRUB 2 configuration.* This contains, among other things, the **/etc/fstab**fileidentifying the rest of the file systems to be mounted.

The kernel starts the systemd process with a process ID of 1 (PID 1).

systemd is the system and service manager in the working Linux. It is backward compatible with **SysVinit scripts** used by previous versions of Linux. It replaces Upstart as the default initialization system.

systemd is the *first process* that starts after the system boots, and it is the *final process* that is running when the system shuts down. It controls the final stages of booting and prepares the system for use. It also speeds up booting by loading services concurrently.

systemd allows you to manage various types of units on a system, including services (name.service) and targets (name.target), devices (name.device), file system mount points (name.mount), and sockets (name.socket).

systemd units are defined by **unit configuration files**located in the following directories:

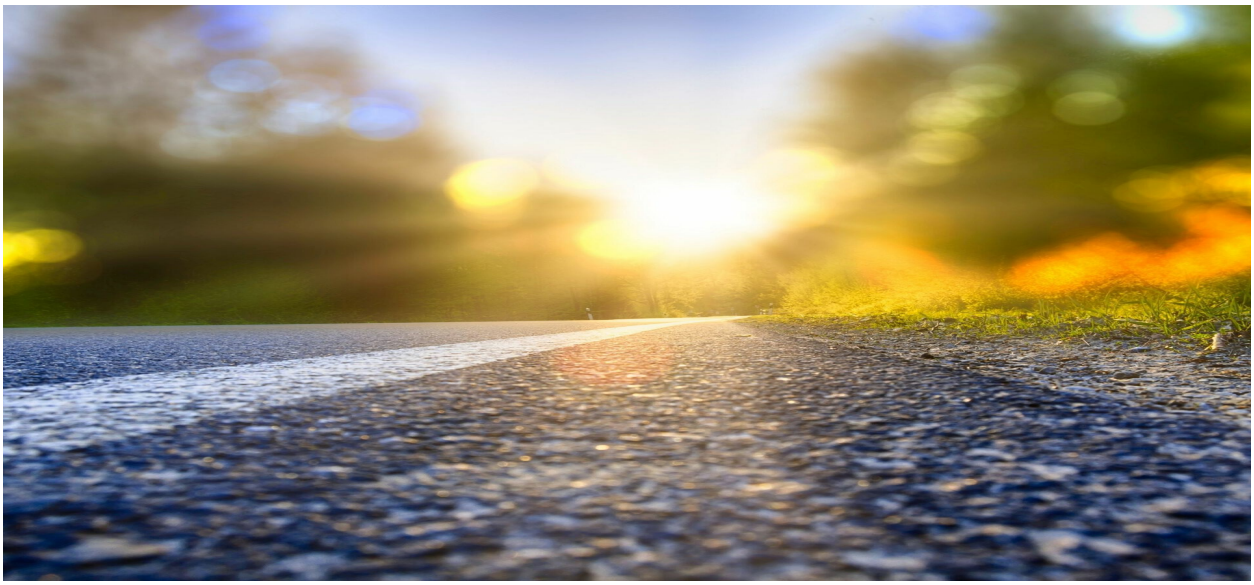
- **/usr/lib/systemd/system/**: systemd units included with installed RPM packages.
- **/run/systemd/system/**: systemd units created by a running program. These take precedence over units in the **/usr/lib/systemd/system** directory.
- **/etc/systemd/system/**: systemd units created and managed by the system administrator. These take precedence over all other units.

The systemd (system and service manager) provides the following features:

- Systemd will start all the daemons (services) in parallel so it provides fast boot when compared to init system.
- Processes are tracked by using Control Groups (cgroups).
    - A cgroup is a collection of processes that are bound together so that you can control their access to the system resources.
- System services can be started on-demand when a client attempts to communicate, when a piece of hardware becomes available, and when a file or directory state changes.

- Snapshotting of the system state and restoration of the system state from a snapshot is supported.

- Mount and auto mount points can be monitored and managed by systemd.
- For administration task we will use **systemctl**(system control) command.



CHAPTER 3 - ABOUT SHELL

What is shell?

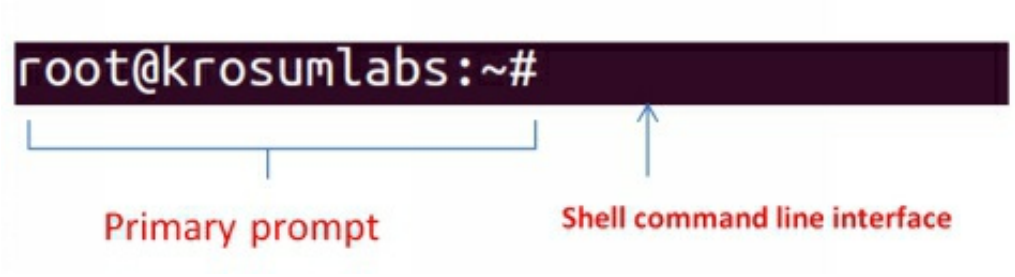
Shell is a *command line interpreter*.

So what is an interpreter? Interpreter is a translator which translates our user instruction into an intermediate format.

So here shell is ***translating our user input (command) to kernel.***

This means that all the Linux commands which we are typing in command line will be interpreted (translated) by shell into kernel understanding format. Then kernel will interact with hardware unit. So *shell has no connection with the hardware*.

Hence we say *Shell is an interface between user and kernel.*



All the user input commands are typed on command line interface and this interface is called **shell primary prompt (or) user interface prompt**.

Linux abstractions are **FILE** and **PROCESS.**The *execution of command* (or) *interpretation of command* is called a ***process*** and this process is created by user; so we can say that this is *user process*.

In Linux every process creates another process, where the newly created process is called ***child process*** and the creator process is called ***Parent process*.** This is termed as ***Linux process model.***

Here shell will interpret our user input command; so shell becomes the parent process and the newly created (execution of command result) process is called child process.

Shell is a parent process of our running commands.

The above snapshot shows the **parent –child** relationship. See the picture where the active (Running) shell primary prompt is marked in red, over which the command **uname** is present.

Note the place where I am typing the command. On the top of the shell interface, the **uname** command is interpreted by the working shell. So here ***shell*** *is called the parent of* ***uname*** *command.* Execution of a command becomes a process and here that process is called a child process to shell. Whenever *child* process is in *active* (running) state, *parent* process will be in *waiting* state. Once child execution is done, child process will reach to exit state and the parent process will resume from waiting state to running state.

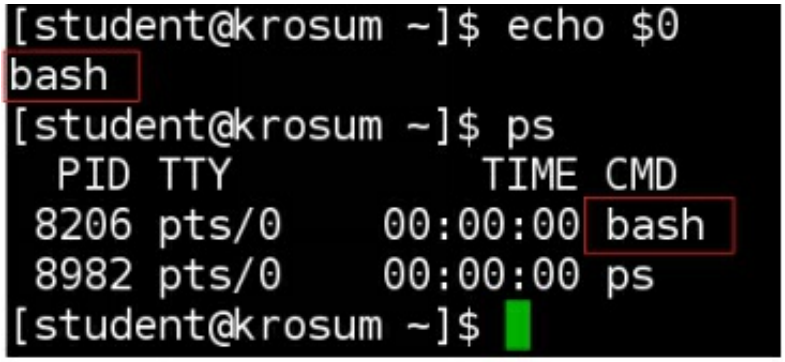
Types of shell

1. Bourne Shell (SH)
2. KornShell(KSH)
3. BourneAgainShell(BASH)
4. C shell(CSH)
5. TurboCshell(TCSH) - Tcsh is enhanced C shell
6. ZSH

How to find current working shell?

1. Type the following special variable in command line

echo **$0** will display current working shell name.

2. 

Another way is to use **ps** command.

We will discuss about the **ps** command and command attributes in next chapter.

Do you know how shell is initialized?

Recap our system boot process. Once *POST* is done successfully, *BIOS* will execute *MBR(Master Boot Record)*. Then MBR will *execute boot loader and* the *boot loader* executes the *kernel*. Then kernel will pass the control to process *(init or systemD)*. Now see the process execution where *init or*

systemD will be responsible to start all our Linux services (system process) and these services are called *daemon process*, which are not created by user. It is all created by *systemD or init.*Type **pstree** command in Linux command line to see the process creation hierarchy.

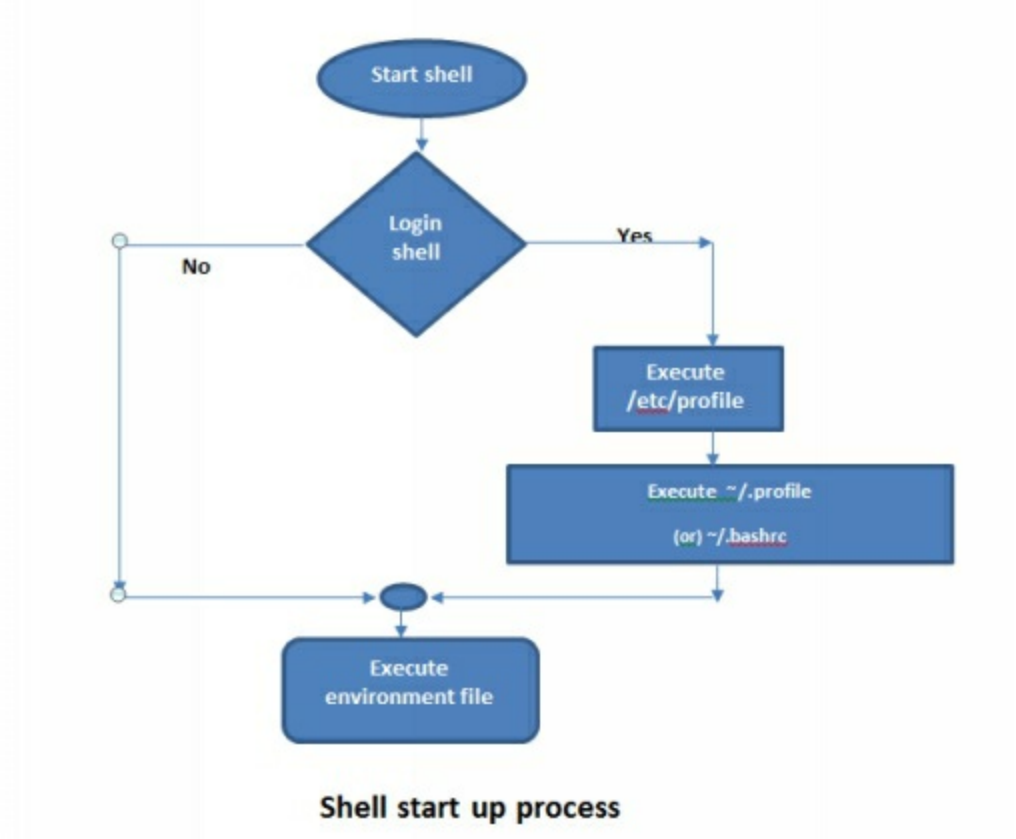
One of the daemons, called *getty(genome teletype terminal)* is our login screen interface. Once login is done successfully, shell will be started.

So ***shell is created by getty***. If login credentials are invalid,*getty* won't execute shell.

Now in user perspective, let's have a look in our graphical desktop system. Once we switch on the PC, both boot process and kernel execution all are done automatically and then we get our login window.

Once login (user name and password) is success, desktop will open. Desktop is a *graphical user interface (GUI)* and on the top of the desktop, all our applications (user process) will start. Similarly once Linux command line environment login is done successfully, shell which is a *command line interface (CLI)* will get initialized. So on the top of the shell; we will execute our Linux *commands* (user process). Hope now you have got answers for the following questions - what is shell? How shell is started? Who is the parent of shell? Who is the child of shell?.

Once Login is done successfully shell will get started but internally shell will read and execute **/etc/profile**file. This is shell script file which is located under **/etc**configuration directory. The **/etc/profile**file, referred as *system profile file* contains all global setting parameters like *permission, umask ,user ID* etc.,



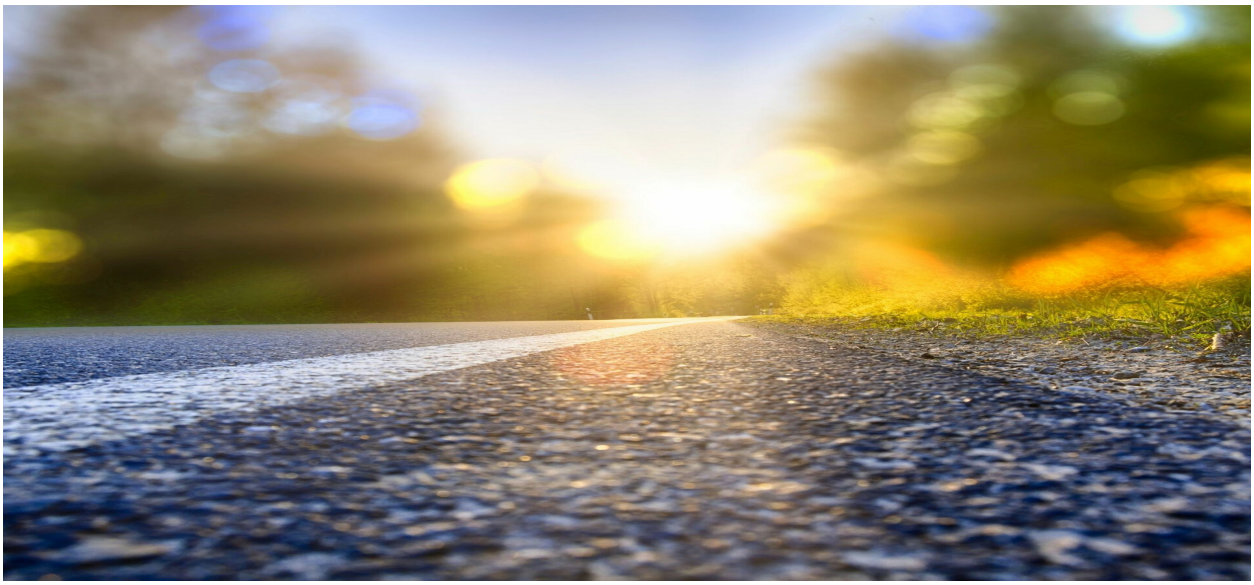
Shell start up process

This system file **(/etc/profile**) provides *personal profile file*(**~/.profile, ~/.bashrc**) that contains commands that are used to customize the startup shell. It is an optional file that runs immediately after the system profile file. This **~/.profile**and **~/.bashrc**files are called as *user file*. If you want to make any changes in personal login environment use personal profile file.

The shell also provides environment for user to execute commands that can be customized using *initialization files* (or) *personal profile files.*

These files contain settings for user environment characteristics, such as:

- *Create alias* of commands
- *Search paths* for finding commands.
- *Default permissions* on new files.
- Exporting *Values for variables* that other programs use.



CHAPTER 4 - FILE STRUCTURE

File structure

Linux File structure is *tree structure* (hierarchical structure). All the files will map from *root directory* (/). There is a chance to get confused between *login root* and *root directory*. Both ***login root (/root)*** and the ***root directory (/)*** *are not same*.

Linux / *symbol* is called *root directory* this is like MyComputer in other operating system.

/ is the entry point to map any files in Linux.

In Linux, however, the root of the file system doesn't correspond with any physical device. It is a *logical location* simply denoted as "/".

In Linux everything is *a file/process.* It means that all peripherals

devices attached with system are considered as a file. Example, consider the hard drive. Hard disk is a storage device. Once it is mounted to the kernel, the kernel will treat them as a file. Such a file is called *device file* represented as **/dev/sda**(device file).

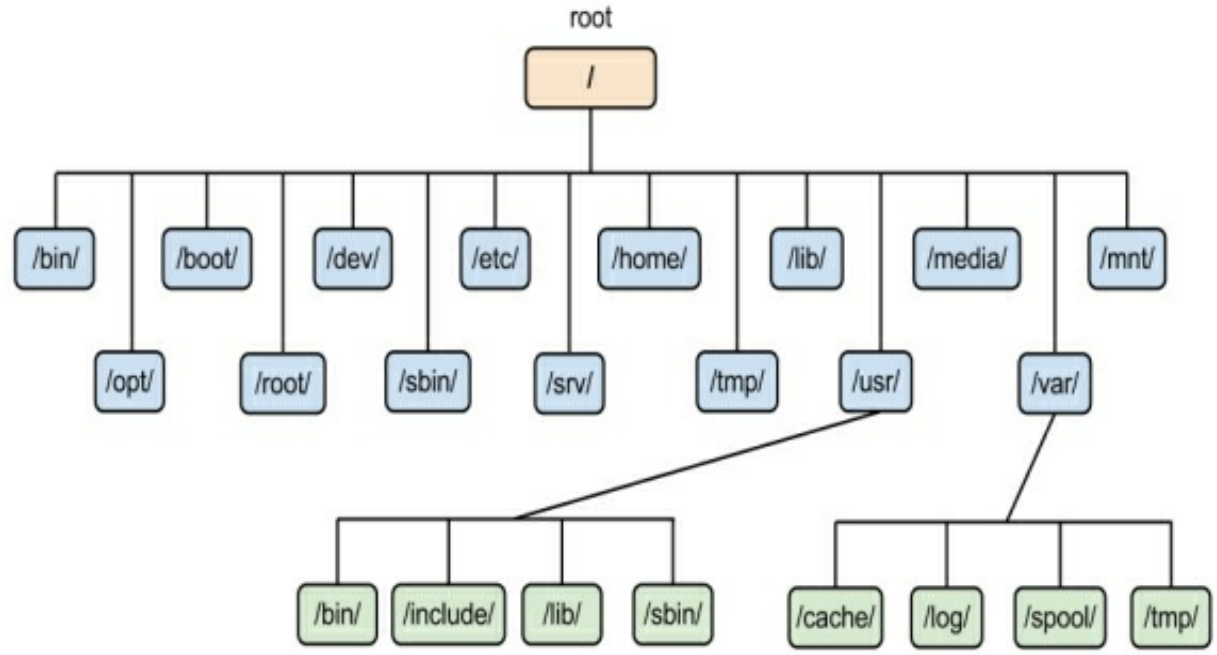
Linux uses a hierarchical file system structure, much like an upside-down tree, with *root (/) at the top of the file system* and all other directories spreading from the root(/) . The *root(/) is the parent directory* and all other directories becomes its sub directories.

Once logged in, the current working directory can be found through **pwd** command.

root@krosumlabs:~# ls/

bindevinitrd.img media procsbin sys var

bootetc lib mnt root selinuxtmpvmlinuz



drom home opt run srvusr

All files and directories appear under the root directory /**.**

DIRECTORY	DESCRIPTION
/	• This is known as “root directory”, the logical beginning of the Linux file system structure. Every single file path in Linux begins from root in one way or another. • / contains the entirety of the operating system.
/bin	• This is where most of the binary files are stored, typically for the Linux terminal commands and core utilities, such as cd (change directory), pwd (print working directory), mv (move), and so on.
/boot	• This is where all the needed files for Linux to boot are kept. • A key thing to note is that even when /boot is stored on different partition, it is still *logically* located at /boot as far as Linux is concerned.
/dev	• This is where the physical devices are mounted, such as the hard drives, USB drives, optical drives, and so on. • The system hard drive is mounted under /dev/sda, whereas the USB thumb drive might be mounted under /dev/sde. • Different partitions on the disk, /dev/sda1, /dev/sda2, and so on are available in Linux.
/etc	• This is where configuration files are stored. Configurations stored in /etc will typically

	affect all users on the system
/home	• This is where all of the personal files are kept. The Desktop, Documents, Downloads, Photos, and Videos folders are all stored under the /home/username directory. • Whereas users can also store configuration files under their own /home folders, which will only affect that particular user.
/lib	• This is where libraries are kept. • When installing Linux software packages, additional libraries are also automatically downloaded, and they almost always start with lib-something. • These files are needed for the programs on Linux to work.
/media	• Another place where external devices such as optical drives and USB drives can be mounted. This varies between different Linux distros.
/mnt	• This is basically a placeholder used for mounting other folders or drives. • Typically this is used for Network locations .
/opt	• Optional software for the system
/var	• This is where variable data is kept, usually system logs but can also include other types of data as well.

/usr	• Contains files and utilities that are shared between users
/proc	• The “processes” directory includes a lot of system information which is represented as files (remember, everything is a file). • It basically provides a way for the Linux kernel (the core of the operating system) to send and receive information from various processes running in the Linux environment. • Using /proc directory we can customize all kernel parameters at runtime.
/root	• This is the login directory path for root user where the root user is called the super user or administrator.
/sbin	• This is similar to /bin, except that it’s dedicated to certain commands that can only be run by the root user, or the super user.
/tmp	• This is where temporary files are stored, and they are usually deleted on shutdown.

Linux File Types

Linux File types are classified as follows.

1. Regular (or) Ordinary file
2. directory file
3. Link file (or) symbolic file (or) soft link file (or) symlink file
4. Device files
5. named pipe or fifo file
6. Socket file

1. **Regular** (or) **Ordinary file**

The Regular file or Ordinary file will be classified as following two types

- **ASCII** or T**EXT**
- **Binary** (or) **ELF** (or) **Object**

What is ASCII or TEXT type file?

- The format in which the file contents can be read or understandable by the user (plain text) is referred as ASCII or TEXT type file.
- Linux file type can be determined by the file command.

How to use file command?

Syntax : filefilename # determine the file type.

root@hostname~]#cat ab.c

#include<stdio.h>

int main(){

printf("Hello\n");

return 0;

}

The above file contents are C program content. Even if we are not aware of C programming language, we can read the content. So this type of file (i.e.,

Understandable format) is called ASCII or TEXT file.

root@hostname~]# file ab.c

ASCII or TEXT

root@hostname~]# file/etc/passwd

ASCII or TEXT

root@hostname~]#file/var/log/boot.log

ASCII or TEXT

So in Linux, *file types **are not determined** by their file extension*.

Binary file (or) object file (or) ELF File

A *binary file* is any *file* that contains at least some data that consists of *sequences of bits* that **do not represent *plain text*.**

Binary files are used to represent images, sound, *executable* (i.e., runnable) programs and compressed data (including documents created by most word processing programs).

They are usually the most compact means of storing data. This is because of the data compression techniques, and the fact is that the programs stored in binary form can be executed faster.

Linux binary files are called as ***ELF***(Executable and Linkable Format) file.

What is an ELF file?

ELF is the abbreviation for **Executable and Linkable Format** and defines the structure for binaries, libraries, and core files.
The formal specification allows the operating system to interpret its underlying machine instructions correctly. **ELF** files are typically the output of a compiler or linker and are a binary format. With the right tools, such file can be analyzed and better understood.

A common misconception is that ELF files are just for binaries or executable.

They *can be used for partial* pieces (object code).

Take our sample Cprogram(**ab.c**) file. When you compile, source (ab.c) will get compiled Executable file (**a.out).**Using that executable file, we can run the C program.That***a.out****file is* ***ELF*** *type file*.

An example is ***shared libraries*** or even ***core dumps*** (those core or **a.out**files). The ELF specification is also used on Linux for the kernel itself and Linux kernel modules.

root@hostname~]# file/lib/libproc-3.2.8.so

ELF

In Linux, all the commands are ELF type files.

root@hostname~]# file/bin/date

ELF

Here***Date*** file is binary file not in a readable format.

All the system commands are ELF type file.

2. Directory File

Directory file *contains a collection of other files and directories.* This resembles like a folder.

3. Link File (Or) Symbolic File (Or) Soft Link File (Or) Symlink File

A symbolic link, also known as a symlink or a soft link, is a special kind of file (entry) that points to the actual file or directory on a disk (like a shortcut in Windows).

Symbolic links are mostly used in administration task.

4. Device files

Recap in Linux, everything is a FILE. So, all the peripheral devices that are attached with system are treated as **device file**.All the device files are mounted under **/dev**directory.

In Linux there are *two categories* of device files: **character** and **block.**

Character-type device files include devices such as keyboard, mouse, and serial ports. In general, operations with these devices (read, write) are performed sequentially byte by byte.

The **Block -type device files** includes devices where *data volume is large* and is organized on *blocks*. Examples: *Hard disks, USB, CDROM* are block-type device files.

5. Named Pipe Or FIFO File

- A FIFO special file (a named pipe) is *similar to a pipe.*
- It can be *opened by multiple processes* for reading or writing.
- When processes are exchanging data via the FIFO, the kernel passes all data internally without writing it to the file system.
- Thus, the ***FIFO special file has no contents on the file system***; the

file system entry merely serves as a reference point so that processes can access the pipe using a name in the file system.

- Using **mkfifo** command we can create FIFO file.

6. Socket file

A socket is a special file used for inter-process communication, which enables communication between two processes.

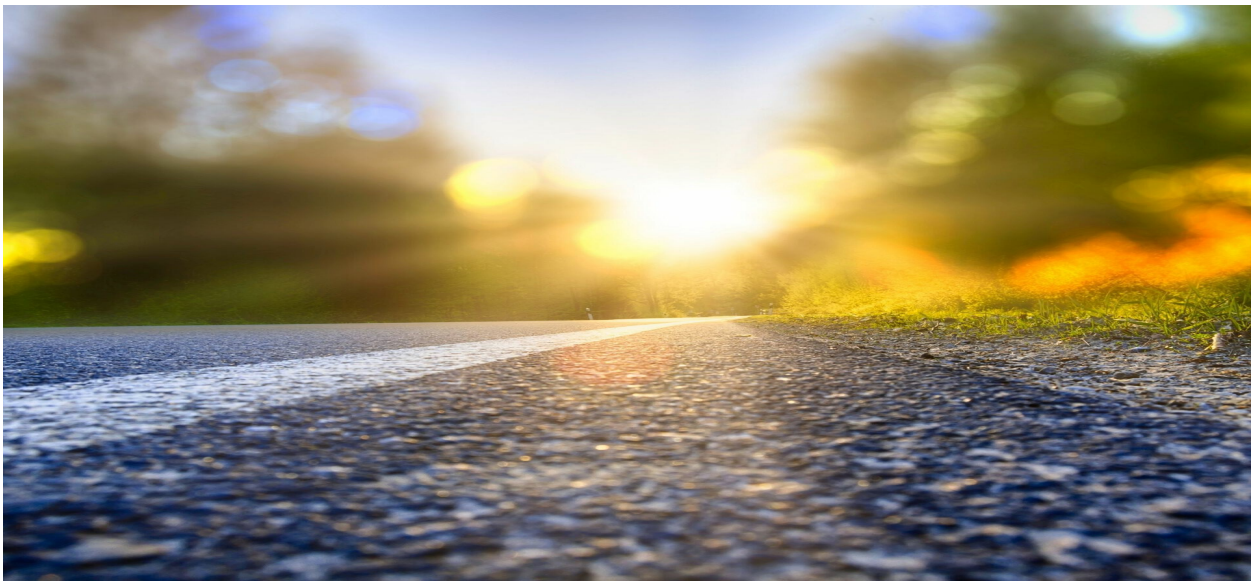
Unlike named pipes which allow only unidirectional data flow, sockets are fully duplex-capable.

There are many way to determine Linux file types

1. Using **file**command (**file filename**)
2. Using**ls–l**command (**ls-l filename**)

Just type**ls–l**followed by the filename. From the resulting output, observe the leftmost first character which depicts the type of file.

Character	File Type
-	regular file
d	directory file
l	link file
c	character type device file
b	block type device file
p	named pipe file
S (capital ‘s’)	socket file



CHAPTER 5 - COMMON LINUX COMMANDS

Linux will treat everything as file and process.

What Is File?

File is nothing but ***data under storage location.***

What is process?

Process is nothing but ***Data under the processor.***

So the job of processor is to fetch data from memory and store it in the register from where execution takes place.

The execution instance is referred *as process.*

So every command in Linux is a binary file and that binary file will become a process once it is executed.

Eg) ***date*** *command* is a *file* and it is also a *process.*

date binary file stored under */bin* is an example for file.

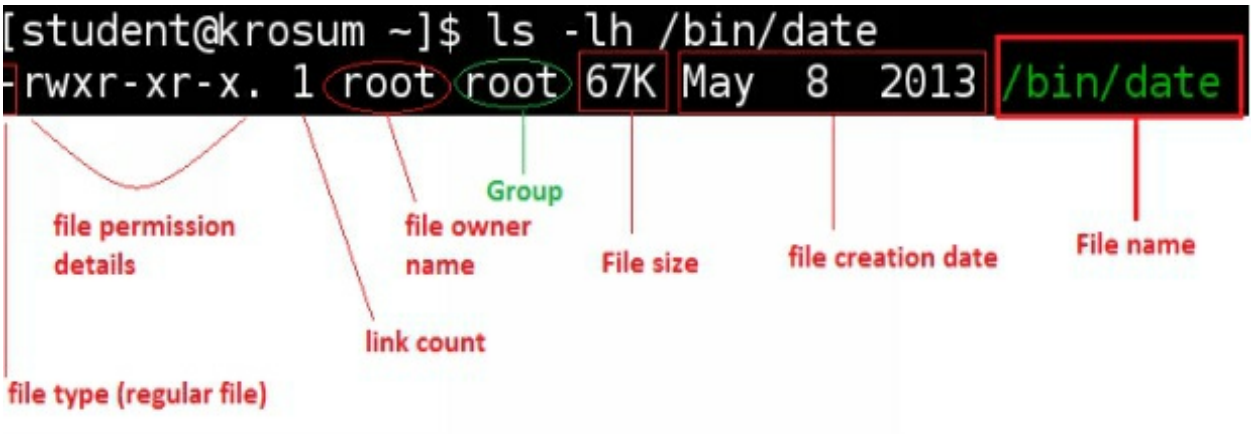


Fig. date file

*date file*when executed by processor will create a process. So now date is a process.

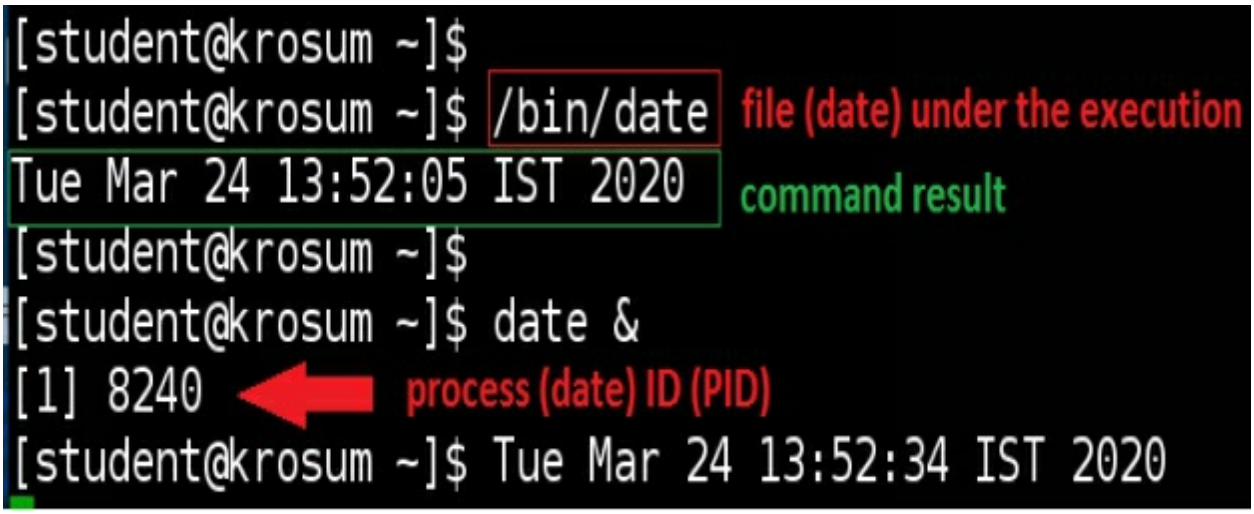


Fig. date process

See the above example, **date** is a command in user view. In Kernel point of view, date is both a file and a process. Like this all the Linux command when entered to execution state will become a process.

From the above snap note the following **symbols:-&**(ampersand) symbol and **[1].** We will discuss them in Linux Process control system.

Linux Command

Linux is command based operating system. The basis of all Linux interaction is the command.

Linux commands are binary files, it's placed on **/bin**(or) **/usr/bin/**directory. *Recap file structures* discussed in previous topics.

- All the Linux Commands are single line entered at a console.
- A Linux command is an action request given to the shell for execution.
- Linux commands are entered at the command line prompt. Command line prompt is known as shell prompt.

Recap shell definition that *shell is an interpreter*; all the user input commands are interpreted by shell.

root@krosum~]# **this is shell command line prompt**

Every command execution displays some result to monitor. This result can be either STDOUT or STDERR.

STDOUT - when a program needs to print output, it normally prints to "standard out".

STDERR - when a program needs to print error information, it normally prints to "standard error".

STDOUT, STDERR -Both are associated with monitor.

Although we do not need to worry about how the command does its job, we must understand exactly what it does. We must know where to find the input and where we want the output to be placed.

In general input comes from keyboard (STDIN) and the output is

usually shown on the monitor (STDOUT/STDERR).

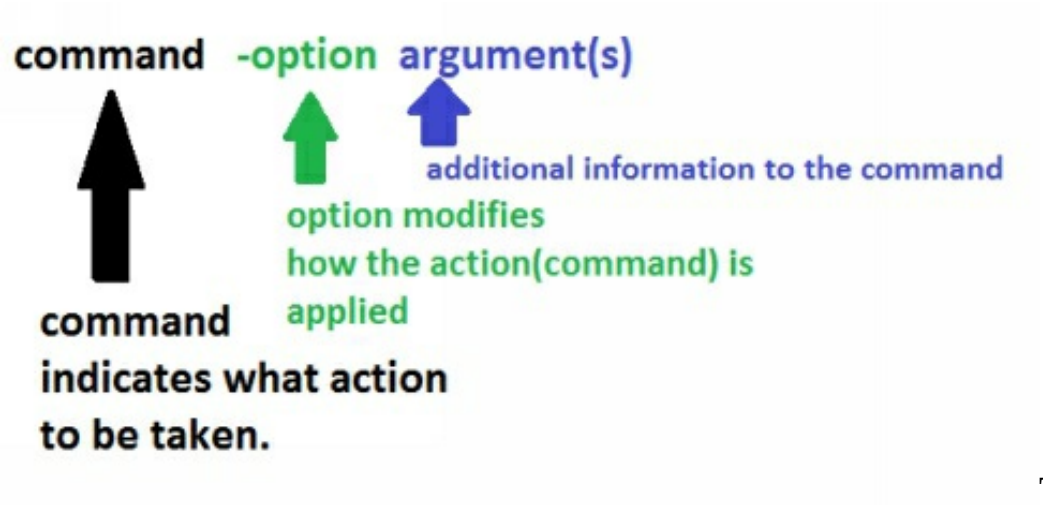
There are many sources to pass input to system; one is keyboard (STDIN) and another source is files on a disk.

Command Syntax

All the Linux commands are entered at *the command line prompt.*

Linux commands are *case sensitive*. Generally all Linux commands are *lowercase characters.*

Every command has *options* and *arguments.*



The command format is:

command-option argument(s)

Linux options are identified as **–(hypen)**followed by single character.

The option characters are case sensitive; all options usage varies by its command.

Examples :

- **ls-r#** Here option **–r**indicates the list out files in reverse order.
- **uname-r#** Here option **–r**indicates released version of Linux kernel.
- **ls-R#** Here option **–R**indicates recursive view of files .

See the above example. With ls command, –r is different from –R. We will discuss more details about this in following topics.

System commands

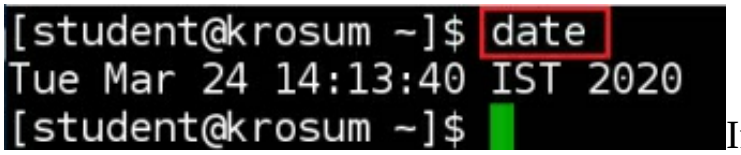
1. **Date**
    - The date command displays system date and time.

- Each date response indicates what time zone is beingused.

Syntax:-date-option argument

The input for date is the system itself. The date is actually maintained in the computer as a part of the operating system.

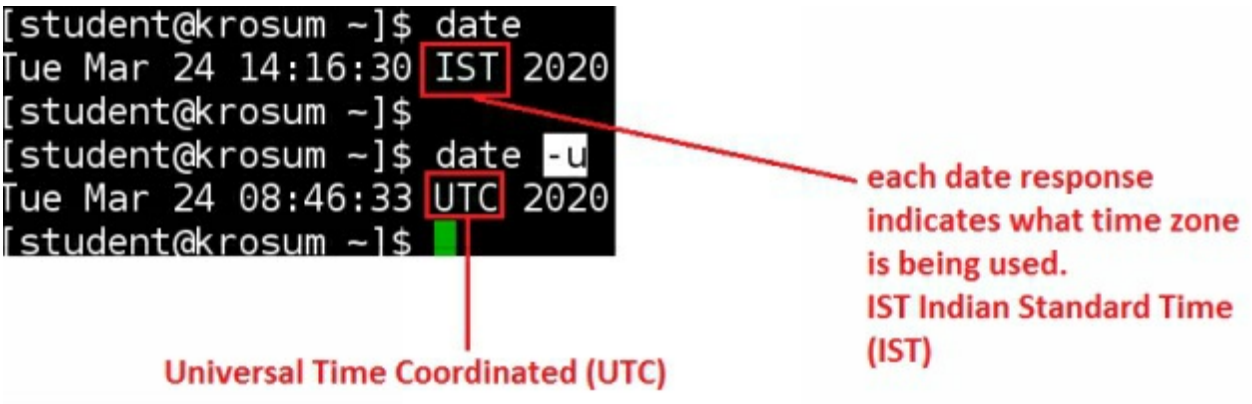
Most modern hardware also has a hardware date and time clock that is often updated automatically to ensure that it is accurate.



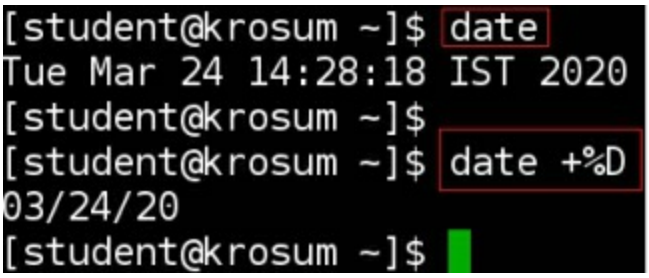
If you enter date command *without any option*, it displays *the current date* and *time* as in the following example.

If *no options are specified* in date command, the *local time* is displayed. Once command execution is done successfully, next primary prompt will appear in the screen automatically.

If a ***-u***option is used, the time is UTC. The following example displays time in UTC (*universal time*) which is equivalent to GMT.



The date commandargument allows you to customize the format of the date. The *format of date* command includes a *plus sign (+) followed by text and series of format codes,all enclosed in double quotes mark (””).* Each code is *preceded by a percentage sign (%)* that identifies it as a code.


[student@krosum ~]$ date
Tue Mar 24 14:28:18 IST 2020
[student@krosum ~]$
[student@krosum ~]$ date +%D
03/24/20
[student@krosum ~]$

FORMAT controls the output. It can be the combination of any one of the following:

%FORMAT String	Description
%%	a literal %
%a	locale’s abbreviated weekday name (e.g., Sun)
%A	locale’s full weekday name (e.g., Sunday)
%b	locale’s abbreviated month name (e.g., Jan)
%B	locale’s full month name (e.g., January)
%c	locale’s date and time (e.g., Thu Mar 3 23:05:25 2005)
%C	century; like %Y, except omit last two digits (e.g., 21)
%d	day of month (e.g, 01)
%D	date; same as %m/%d/%y
%e	day of month, space padded; same as %_d
%F	full date; same as %Y-%m-%d
%g	last two digits of year of ISO week number (see %G)
%G	year of ISO week number (see %V); normally useful only with %V
%h	same as %b
%H	hour (00..23)
%I	hour (01..12)

%j	day of year (001..366)
%k	hour (0..23)
%l	hour (1..12)
%m	month (01..12)
%M	minute (00..59)
%n	a newline
%N	nanoseconds (000000000..999999999)
%p	locale’s equivalent of either AM or PM; blank if not known
%P	like %p, but lower case
%r	locale’s 12-hour clock time (e.g., 11:11:04 PM)
%R	24-hour hour and minute; same as %H:%M
%s	seconds since 1970-01-01 00:00:00 UTC
%S	second (00..60)
%t	a tab
%T	time; same as %H:%M:%S
%u	day of week (1..7); 1 is Monday
%U	week number of year, with Sunday as first day of week (00..53)
%V	ISO week number, with Monday as first day of week (01..53)

%w	day of week (0..6); 0 is Sunday
%W	week number of year, with Monday as first day of week (00..53)
%x	locale’s date representation (e.g., 12/31/99)
%X	locale’s time representation (e.g., 23:13:48)
%y	last two digits of year (00..99)
%Y	Year
%z	+hhmm numeric timezone (e.g., -0400)
%:z	+hh:mm numeric timezone (e.g., -04:00)
%::z	+hh:mm:ss numeric time zone (e.g., -04:00:00)
%:::z	numeric time zone with : to necessary precision (e.g., -04, +05:30)
%Z	alphabetic time zone abbreviation (e.g., IST)

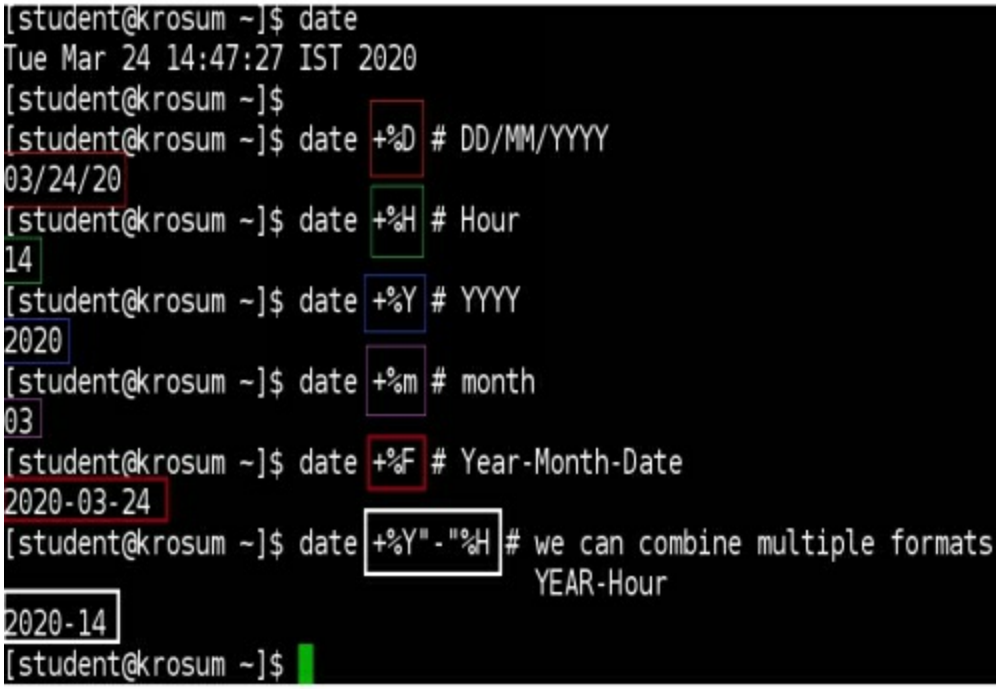
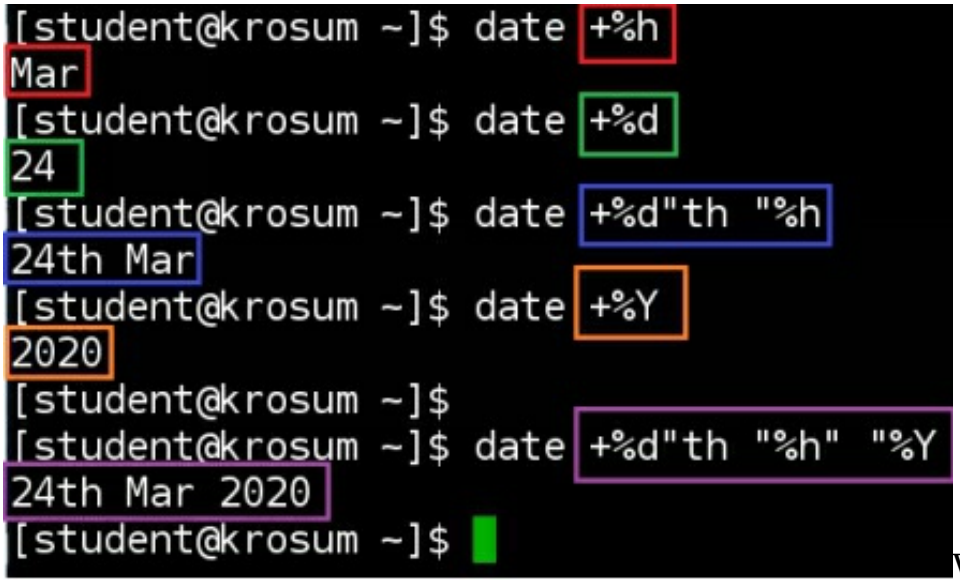


Fig. Date Command with format string



We can combine multiple date formats as per our required format style. See the below examples.

Fig. Date Command with multiple format string

The date command can also be used to set the date and the time but can be done only by a system administrator.

For Example use the following syntax to change the current date.

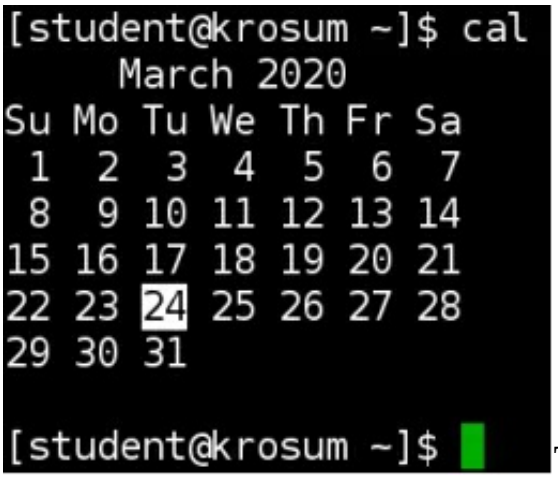
Replace YYYY with a four-digit year, MM with a two digit month, and DD with a two digit day of the month.

Syntax: **date +%D –s <YYYY-MM-DD>**

date +%D -s 2020-05-11

05/11/20

2. *Calendar (cal) command*



The calendar command (cal), displays the calendar for a specified month or a year.

Syntax:- ***cal-options [[month]year]***

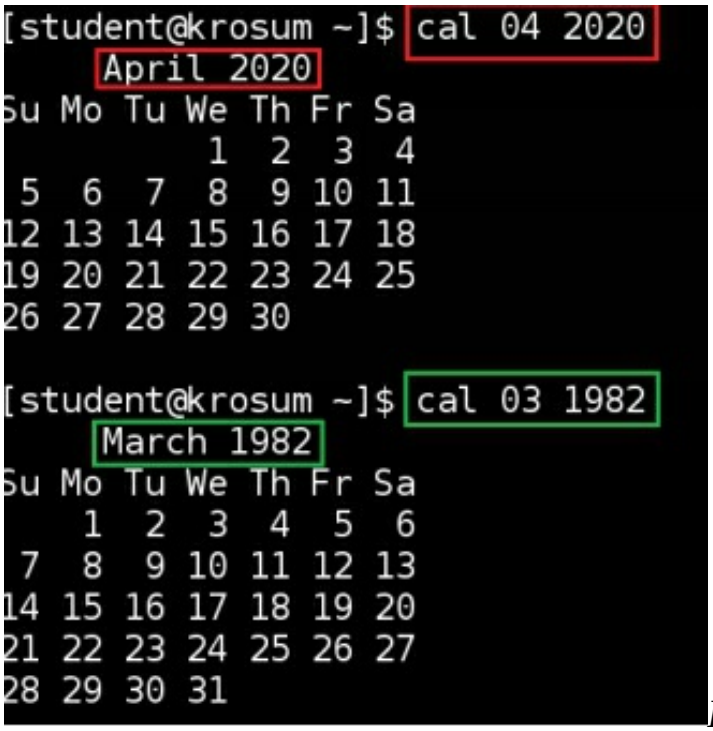
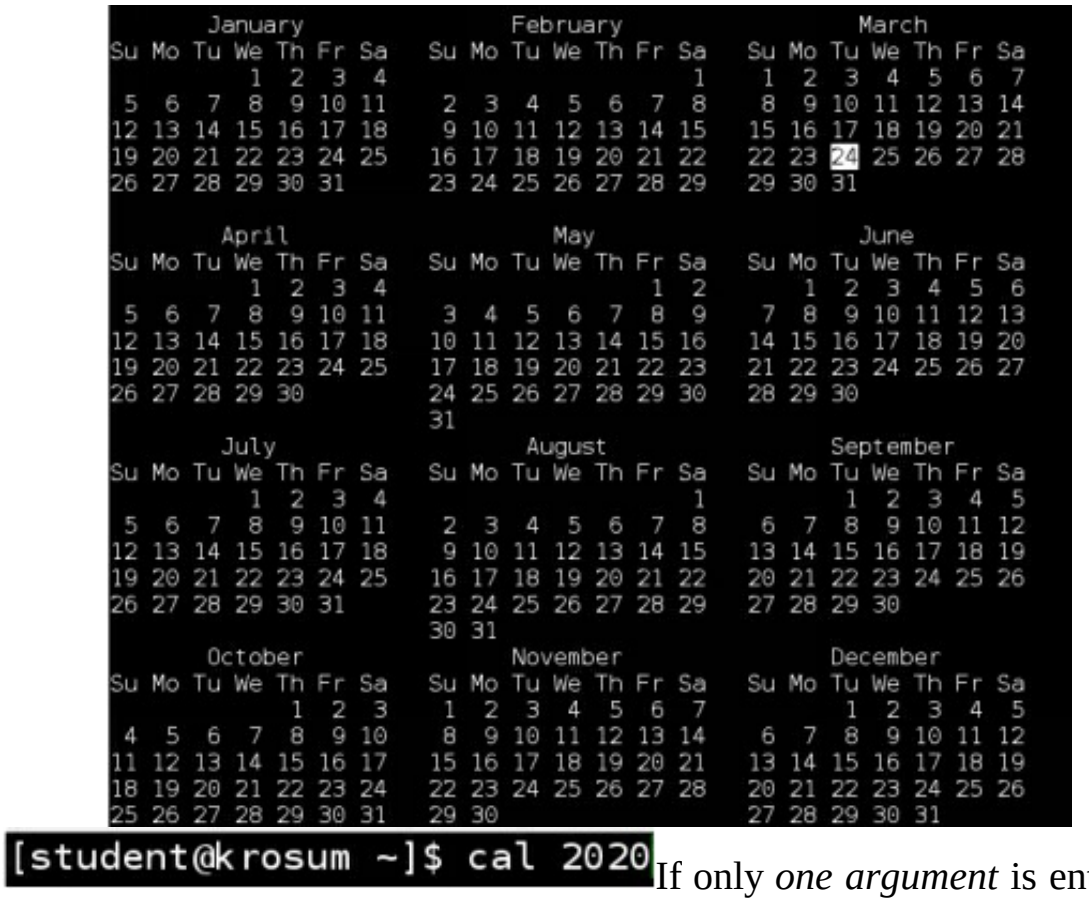


Fig. cal command without option and arguments

Fig.cal command with 2 argument no option

From the above example, there are *2 arguments without any options*.

Arguments (i.e. month, year) *are still optional*. If there is no argument, the calendar for current month is displayed.



If only *one argument* is entered, it is *assumed to be a year* and *not month*. Hence the calendar for the year is displayed.

Fig.cal command with 1 argument assumed as year

If both month and year are entered, then just one month of that particular year is displayed.**Note: -** The 2 arguments are separated with a whitespace.

3. *Who command*

The who command displays all the users currently logged into the system.

Syntax:- ***Who -options***

```
[student@krosum ~]$ who
student  tty1         2020-03-19 20:22 (:0)
student  pts/0        2020-03-24 13:45 (:0.0)
student  pts/1        2020-03-24 14:47 (:0.0)
[student@krosum ~]$
```

The who command returns the user's name (ID), corresponding terminal and time he/she logged in.

with option -u

```
[student@krosum ~]$ who -u
student  tty1         2020-03-19 20:22  old         882 (:0)
student  pts/0        2020-03-24 13:45   .         8202 (:0.0)
student  pts/1        2020-03-24 14:47 00:24       8202 (:0.0)
[student@krosum ~]$
```

To display the duration how long the system was idle (idle time), -u option is used.

with option -H

This option is used to get header information.

```
[student@krosum ~]$ who -H
NAME     LINE         TIME             COMMENT
student  tty1         2020-03-19 20:22 (:0)
student  pts/0        2020-03-24 13:45 (:0.0)
student  pts/1        2020-03-24 14:47 (:0.0)
```

Who am i

```
[student@krosum ~]$ who am i
student  pts/0        2020-03-24 13:45 (:0.0)
```

If you want to display only your *individual system information*, enter **who am i**.(note the white space entered between each word).

Whoami

- The whoami command *returns the user ID*(Login name).

```
[student@krosum ~]$ whoami
student
[student@krosum ~]$
```

```
[root@krosum ~]# whoami
root
[root@krosum ~]#
```

- If you logged in as root user, whoami displays root.
- If you logged in as Student user, whoami displays Student.

This is equivalent to the result of the following command:-

```
[student@krosum ~]$ whoami
student
[student@krosum ~]$ id -un
student
[student@krosum ~]$ id -u
1000
[student@krosum ~]$
```

```
[root@krosum ~]# whoami
root
[root@krosum ~]# id -un
root
[root@krosum ~]# id -u
0
[root@krosum ~]#
```

id-un

id -u displays only the login ID(Integer). This is always zero(0) for root user.

4. System name (uname) command

Theunameutility is most commonly used to determine *the processor architecture*, the *system hostname* and the *version of the kernel* running on the system.

Syntax:- **uname–option**

Option	Description
-r(kernel release)	Prints the kernel release
-v	Prints the kernel version
-m	Prints the name of the machine's hardware name
-p	Prints the architecture of the processor
-i	Prints the hardware platform
-o	Print the name of the operating system. On Linux systems that is "GNU/Linux"
-a(all)	When the -a option is used, uname behaves the same as if the -snrvmo options have been given.

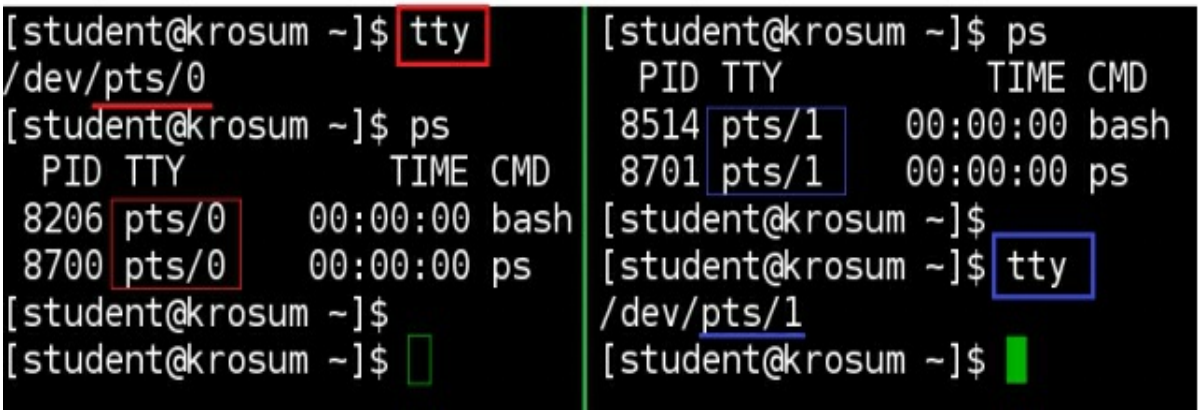
```
[student@krosum ~]$ uname
Linux
[student@krosum ~]$ uname -r
3.9.5-301.fc19.i686
[student@krosum ~]$ uname -s
Linux
[student@krosum ~]$ uname -i
i386
[student@krosum ~]$ uname -p
i686
[student@krosum ~]$ uname -m
i686
[student@krosum ~]$ uname -o
GNU/Linux
```

```
[student@krosum ~]$ uname -a
Linux krosum.com 3.9.5-301.fc19.i686 #1 SMP Tue Jun 11 20:01:50 UTC 2013 i686 i686 i386 GNU/Linu
```

Fig. uname command with options

5. Terminal (tty) command

The ***tty***utility is used to show the *name of the terminal* we are using.

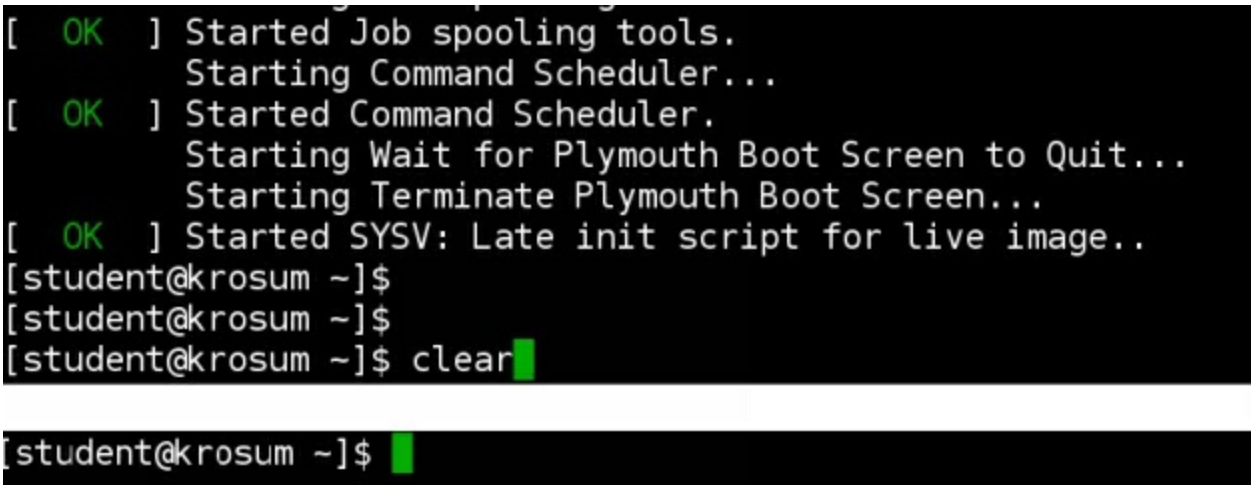


Already we know that Linux treats everything as File and Process. *So**tty**is also a file*stored under ***/dev***directory.

When we open a new terminal, port number will be incremented automatically as pts/0, pts/1 and so on.

6. Clear screen (clear) command

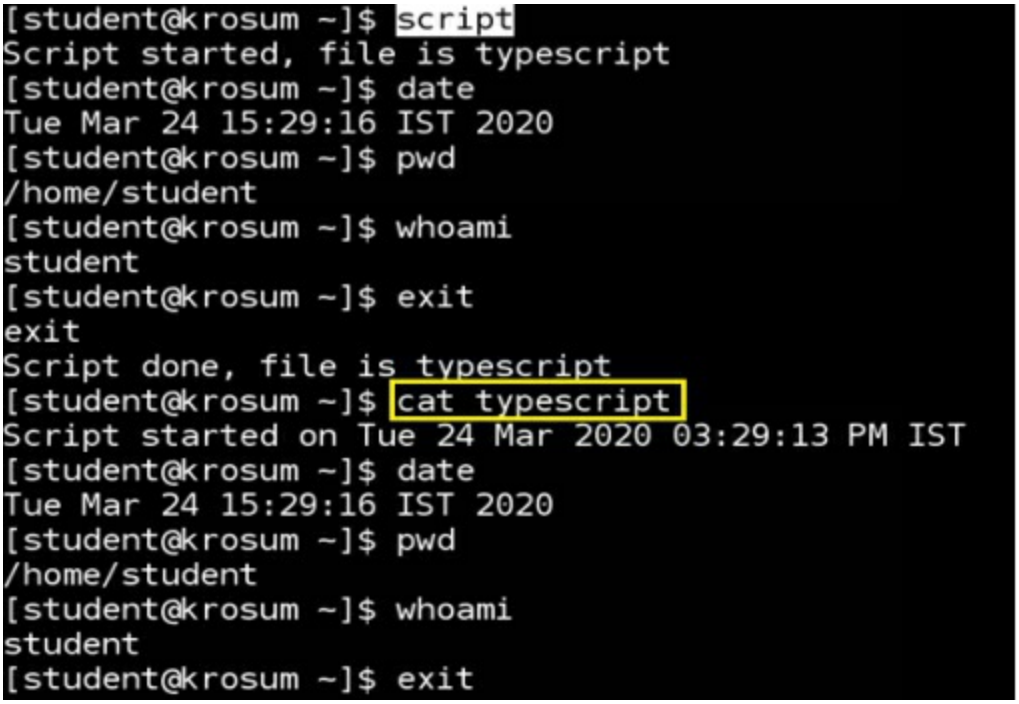
The clear command clears the screen and places the cursor at the top.



7. *Script command*

- This records an interactive session.
- When you want to record the full session, use this as your first command on your session so that everything you do will be recorded until you log out from the session.
- To stop the recording, type ***exit***
- If you do not specify your output log file name, then by default the session log is saved as a file named *typescript.*

Refer the below example for more details.



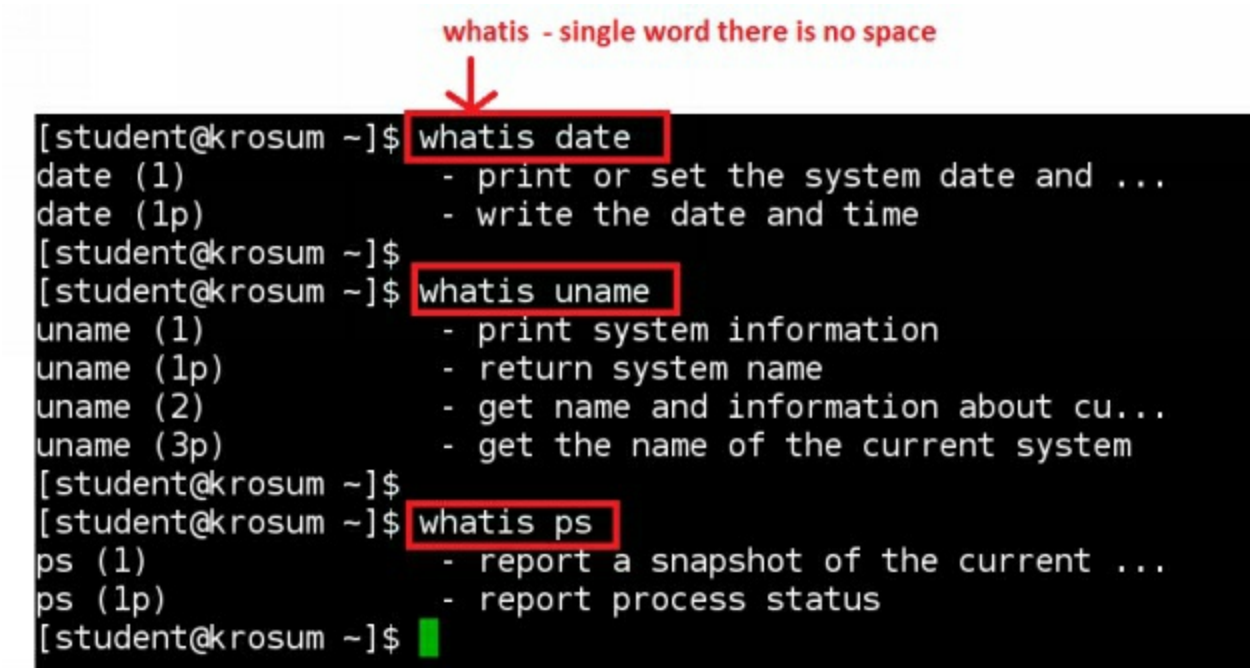
Help documentation

- **whatiscommand**
- **man command**

1. **whatis command**

It is a help command that displays the*definition of the command.*In order to get more detailed (syntax, options) information about command, refer man command.

Syntax:- **whatis<command>**



2. **mancommand**

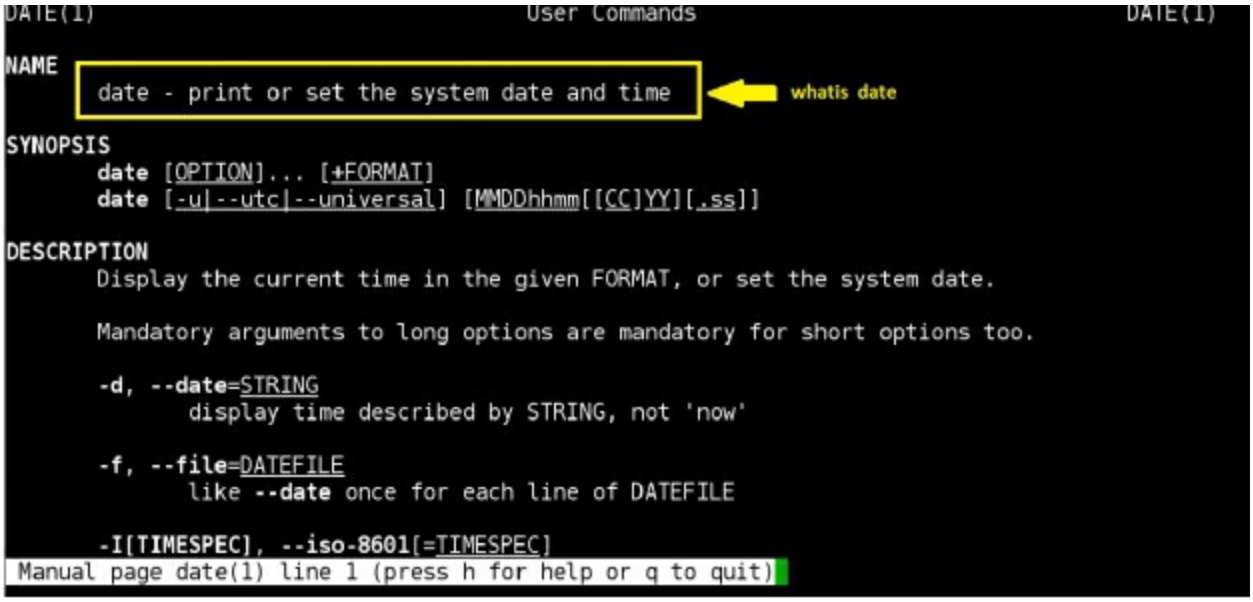
Syntax:- ***man***<command>

man is online reference manual(man)page. It provides detailed descriptions and usage of the commands.

You can use the man command to display the man page entry that explains a given command.

For example, display the man pages for the date command

```
[student@krosum ~]$ man date
```



Using the man command

Name– definition about command

Synopsis- Command with options – Syntax about command (or) how to apply command in command line.

Scrolling Through the Man Pages

The following table lists the keyboard commands for scrolling through the man pages.

Key board command	Action
Space bar	Displays the next screen of a man page
Return	Displays the next line of a man page
B	Moves back one full screen
/pattern	Searches forward for a pattern
N	Finds the next occurrence of a pattern after you have used **/pattern**
H	Provides a description of all scrolling capabilities
Q	Quits the man command and returns to the shell Prompt

Searching the Man Pages

There are two ways to search for information in the man pages:

- *Searching by section*
- *Searching by keyword*

1. ***Searching the Man Pages: By Section***

The online man page entries are organized into sections based on the type or usage of the command or file.

For example, Section 1 contains user commands, and Section 5 contains information about various file formats.

To look up a specific section of the man page, use the man command with the -s option, followed by the section number, and the command or file name.

man-s number command

(or)

man-s number filename

The *bottom portion* of a man page, titled SEE ALSO, lists *other commands or files related to the man page.*

The *number in parentheses* reflects the *section* where the man page is located. We can use the man command with the **-l**option to list the man pages that relate to the same command or file name.

For example, *to view the online man page for the passwd file*, use the following commands:

man-l passwd

passwd (1) -M /usr/man

passwd (4) -M /usr/man

man-s 4 passwd

Reformatting page. Please Wait... done

File Formats passwd(4)

NAME

passwd - password file

SYNOPSIS

/etc/passwd

DESCRIPTION

The file /etc/passwd is a local source of information about users' accounts. The password file can... (Output truncated).

2. Searching the Man Pages: By Keyword

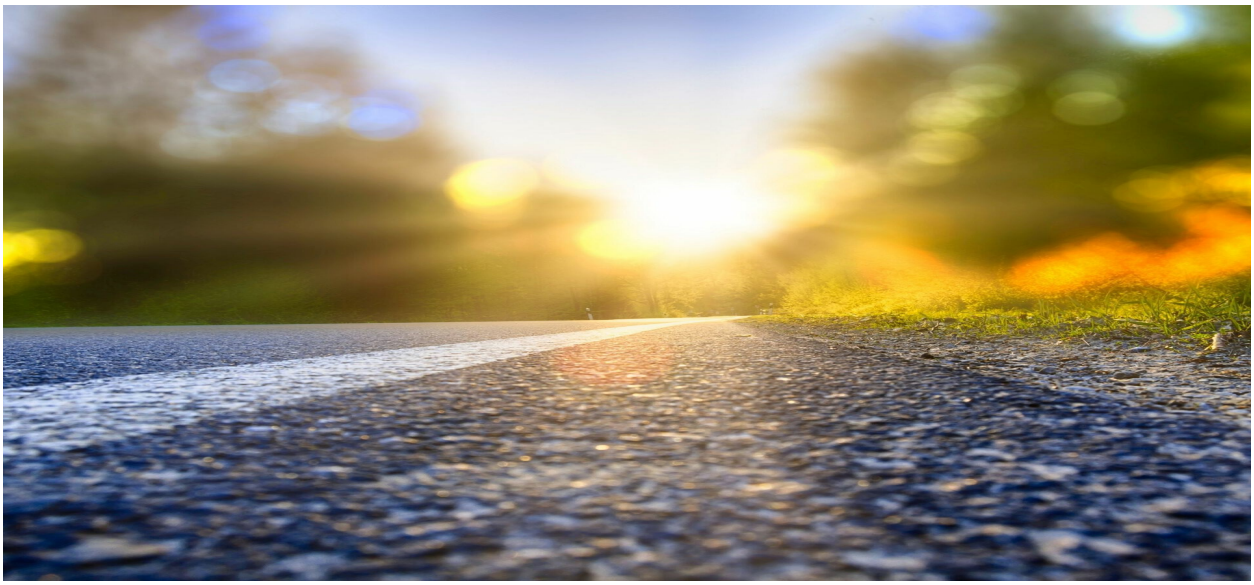
When we are unsure of the name of a command, we can use the man command with the -k option and a keyword to search for matching man page entries

man-k keyword

The man command output provides a list of commands and descriptions that contain the specified keyword.

For example, using the man command, view commands containing the calendar keyword.

$ man-k calendar



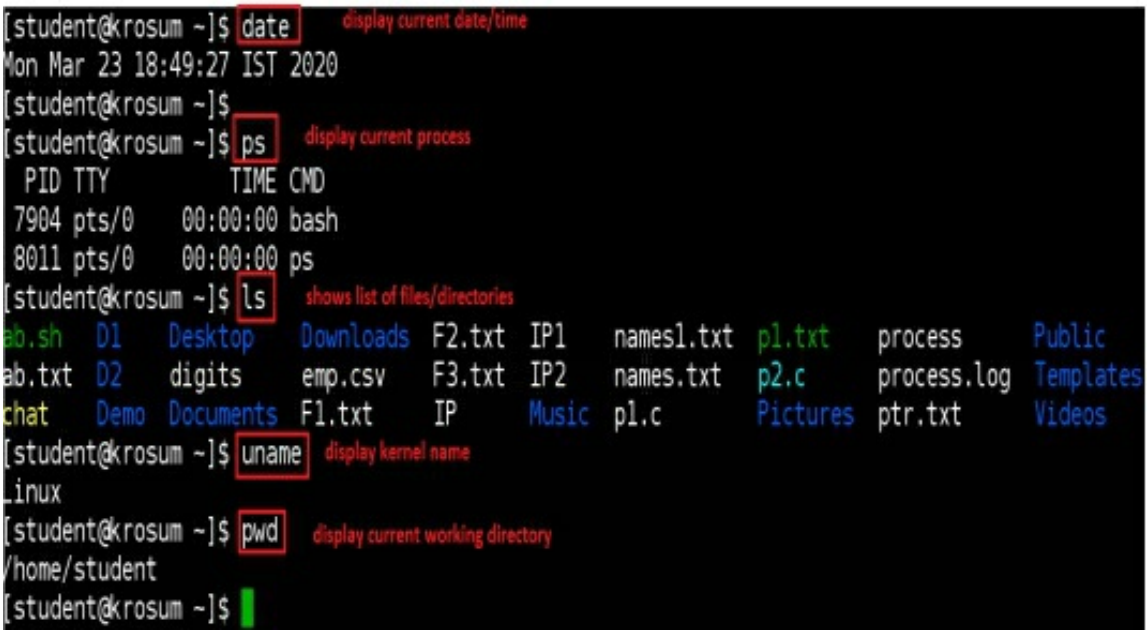
CHAPTER 6 - LINUX COMMAND LINE STRUCTURE

Linux Command line structure consists of following format.

- Command only - there is no option and arguments
- Command with arguments
- Command with `command` (or) Command $(Command)

1. **Command only**

    - There is no option and arguments.
    - Command line prompt allows only a single command.

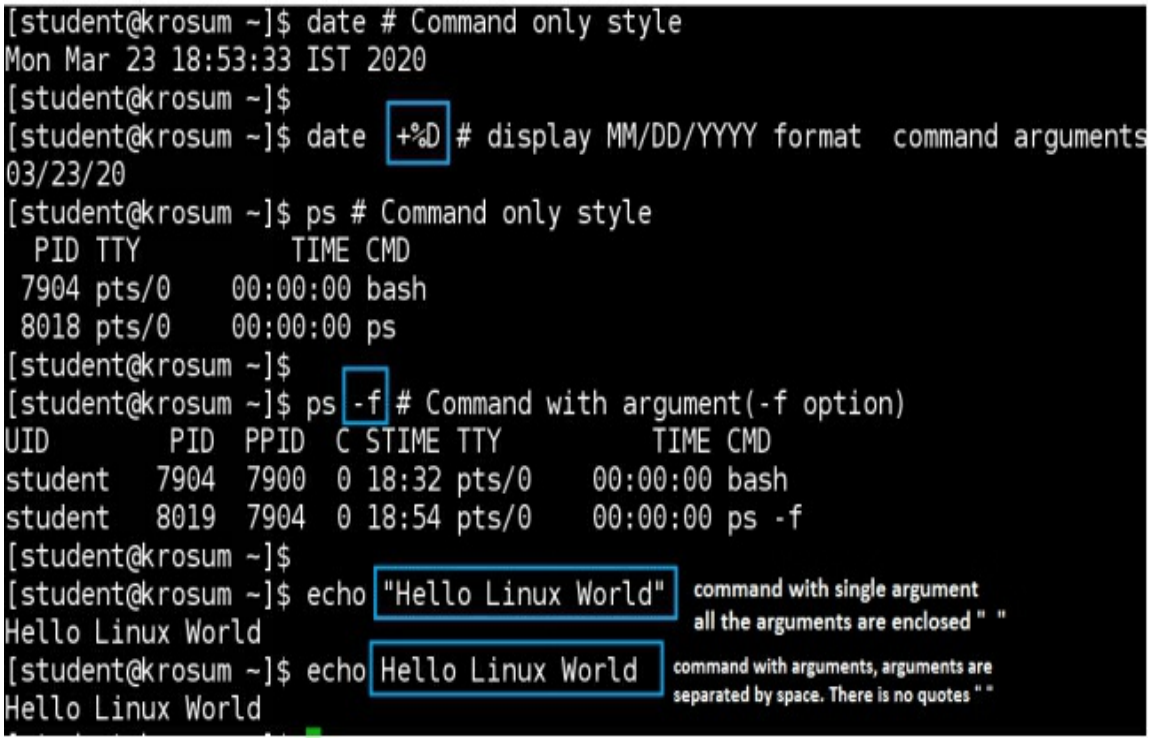


See the below example snap

Command followed by option or arguments style.

Command<space>**-option**
Command <space>**argument**
Command <space> **-option**<space>**argument**

- There is a single space in between command and option, again followed by a space prefixing the arguments.



Note the difference in usage of echo command with quotes and without quotes.

In the upcoming topics, we will discuss more about the command options. Just understand the command line structure.

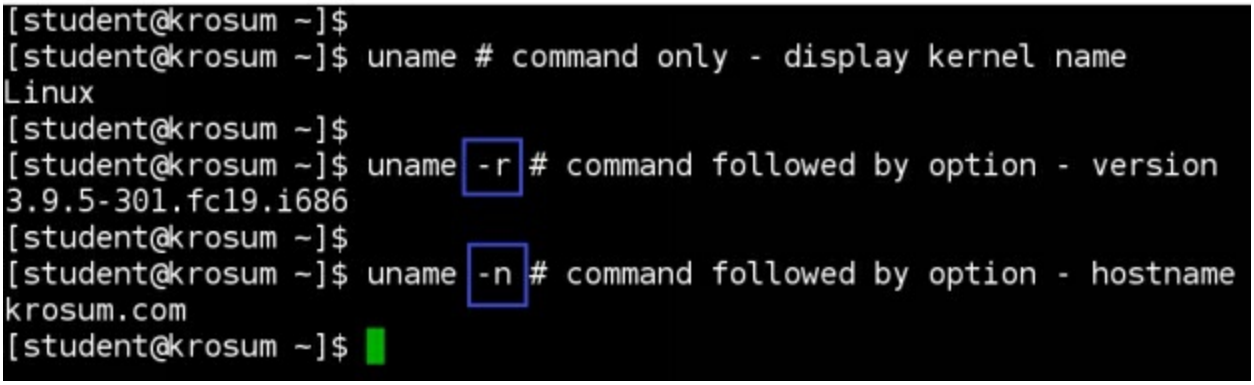
Difference between **ps** vs **ps-f**and**date** vs. **date+%D**format.

In Linux, command options are useful to get specific information about command. In learning point of view, first let's focus on the command and then will explore more on options and arguments.

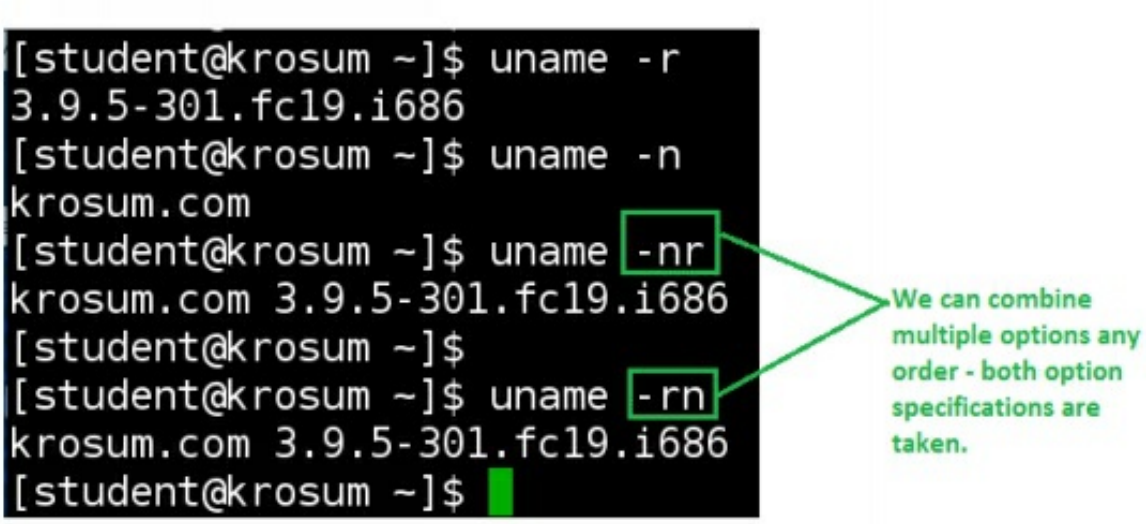
command only style :- ***uname***displays kernel name.

command –option structure :- ***uname-r***displays kernel version and

uname*-*ndisplays hostname



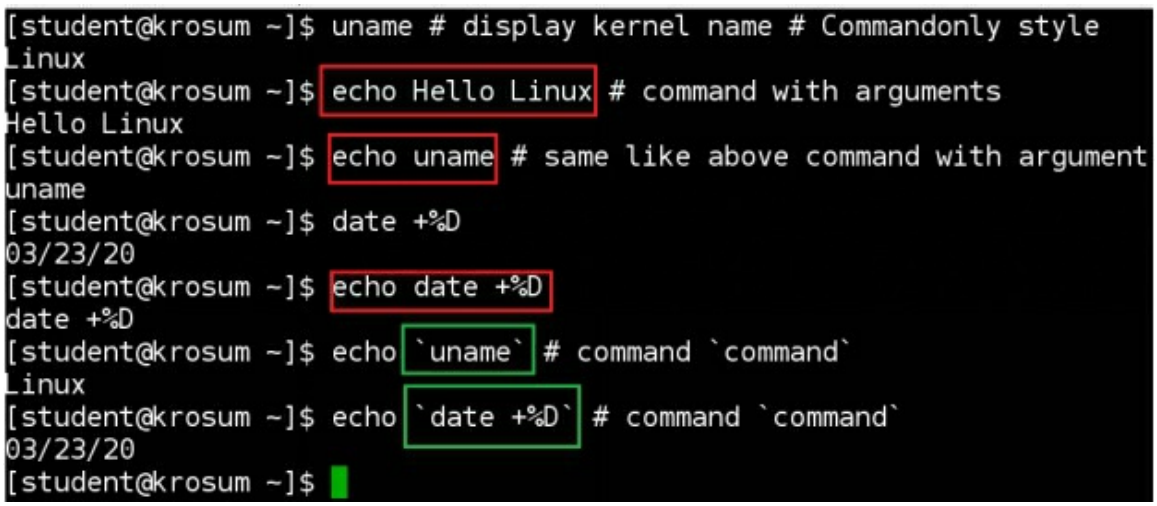
See the above 3 command line difference, when we put option –r (or) –n the command display specific result. This is command –option structure.



We can combine multiple options together in any order.

1. Command followed by Command style

Whenever we pass command without a back quote as an argument to another command, shell will treat it as any ordinary word and won't consider it as a command.



Note the difference between**echo uname**vs**echo `uname`**

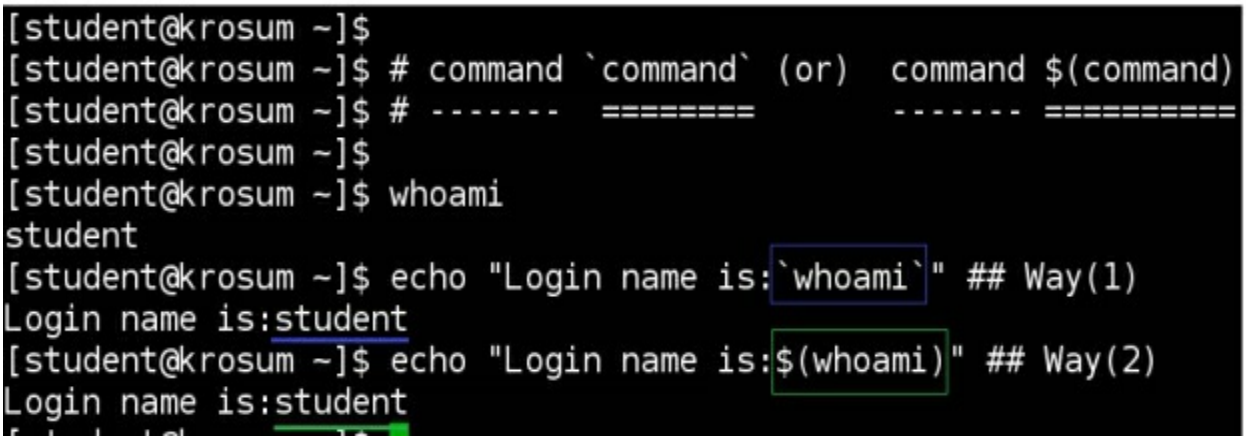
- **uname**is a Linux command and it displays working kernel name (see the 1st line)
- **echo** command is used to display message to console (see the 2nd line in above snap)

When we combine both **uname** and **echo** command, shell will interpret them as *command with argument*(**echouname**). Here **uname**is considered as *ordinary word* and *not a command*. That's the reason **echo uname** displays message as **uname**to monitor.

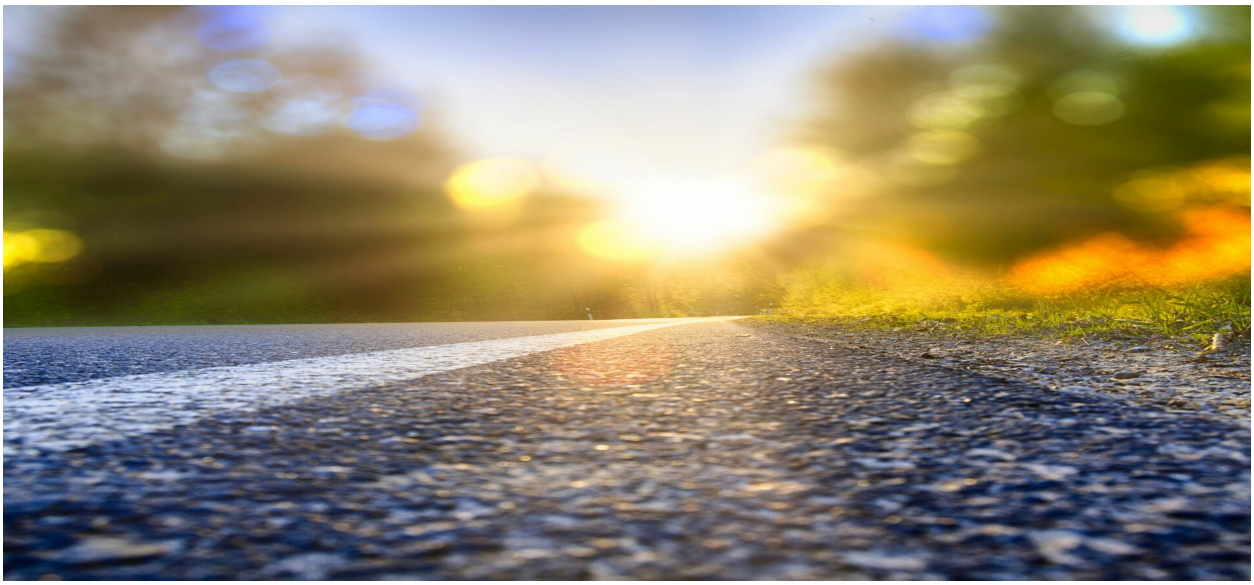
When we enclose command with back quote ` ` symbol, shell interprets them as command with `command`.

See the difference between command line (A) and command line (B)

When we pass command *as an argument to another Linux command*, the command being passed is *enclosed with back quote*.i.e. ` *command* ` (back quote notation).



Note the difference between *a back quote* and *single quote*. Both *are different.*



CHAPTER 7 - THE VI EDITOR

- The ***vi*** editor is a *command-line, interactive editor* that we can use to *create* and *modify the text files*.
- The ***vi***editor is also *the only text editor* that we can *use to edit* certain *system files without changing the permissions* of the files.
- ***vi*** improved *(**Vim**)* is the *default editor.*
- The ***Vim*** editor is an enhanced version of the ***vi*** editor.

Accessing the vi Editor

- To *create*, *edit,* and *view files* in the vi editor, use the ***vi*** command.
- The ***vi*** command includes the following three syntaxes:

vi

vifilename

vioptionsfilename

- The initial display of the editor in a terminal window is a *blank windowfilled with tildes* and a *blinking cursor* in the *top left corner*.

- If the *system crashes while you are editing a file*, you can use the ***-r***option ***to recover the file***.

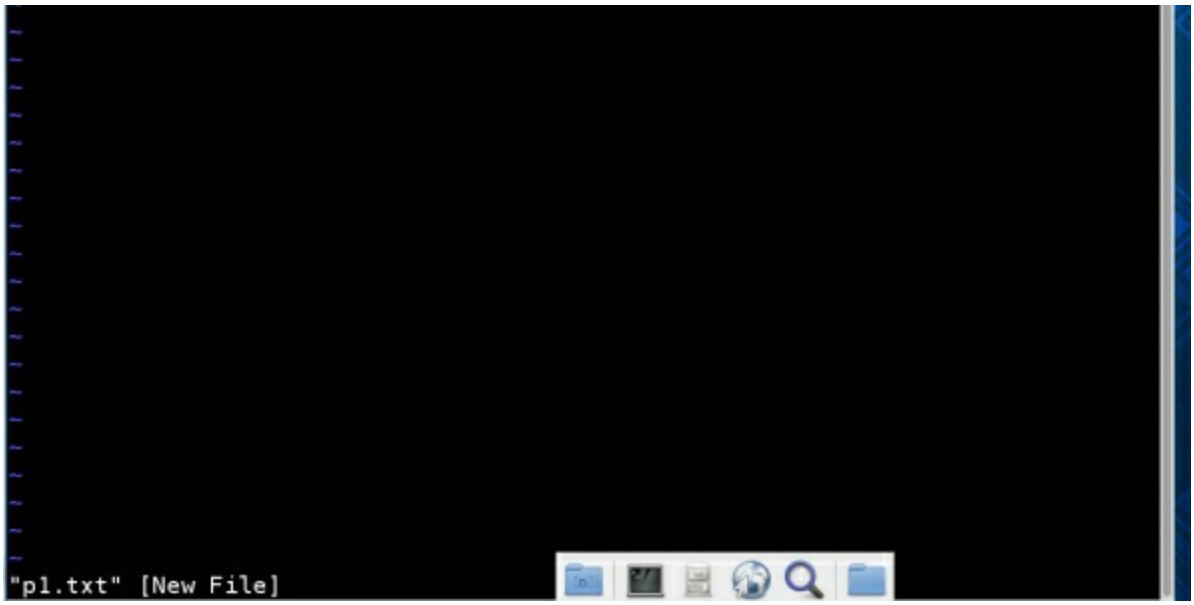
vi-rfilename

The file opens so that you can edit it. You can then save the file and exit the ***vi***editor, by using the following command:

vi-Rfilename

The file opens in read-only mode to prevent accidental overwriting of the contents of the file.

student@krosum ~]$vi p1.txt {Enter}



The vi Editor Modes

The vi editor provides *three modes of operation*:

- **Command**
- **Input**
- **Last line**

1. **Command mode**

   - The command mode is the *default mode* for the ***vi***editor.
   - In this mode, you can run commands *to delete*, *change*, *copy*, and *move text*.

- You can also *position the cursor, search for text strings,* and *exit the* ***vi****editor.*

2. Input mode

- You can *insert text* into a file *in the input mode*.
- The ***vi*** editor *interprets everything* you type in the input mode *as text.*
- To *invoke input mode,* press any one of the following ***lowercase keys:***

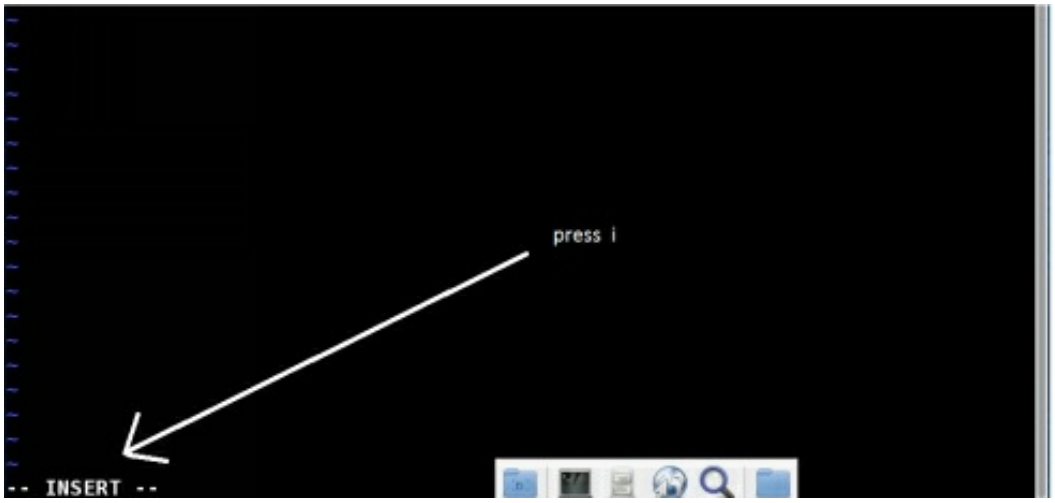
KEY	DESCRIPTION
i	Inserts *text before the cursor*
o	Opens a *new blank linebelow the cursor*
a	Appends text *after the cursor*

- You can also invoke the input mode *to insert text into a file* by pressing one of the following ***uppercase keys:***

KEY	DESCRIPTION
I	Inserts *text at thebeginning of the line*
O	Opens a *new blank line above the cursor*
A	Appends text at the *end of the line*

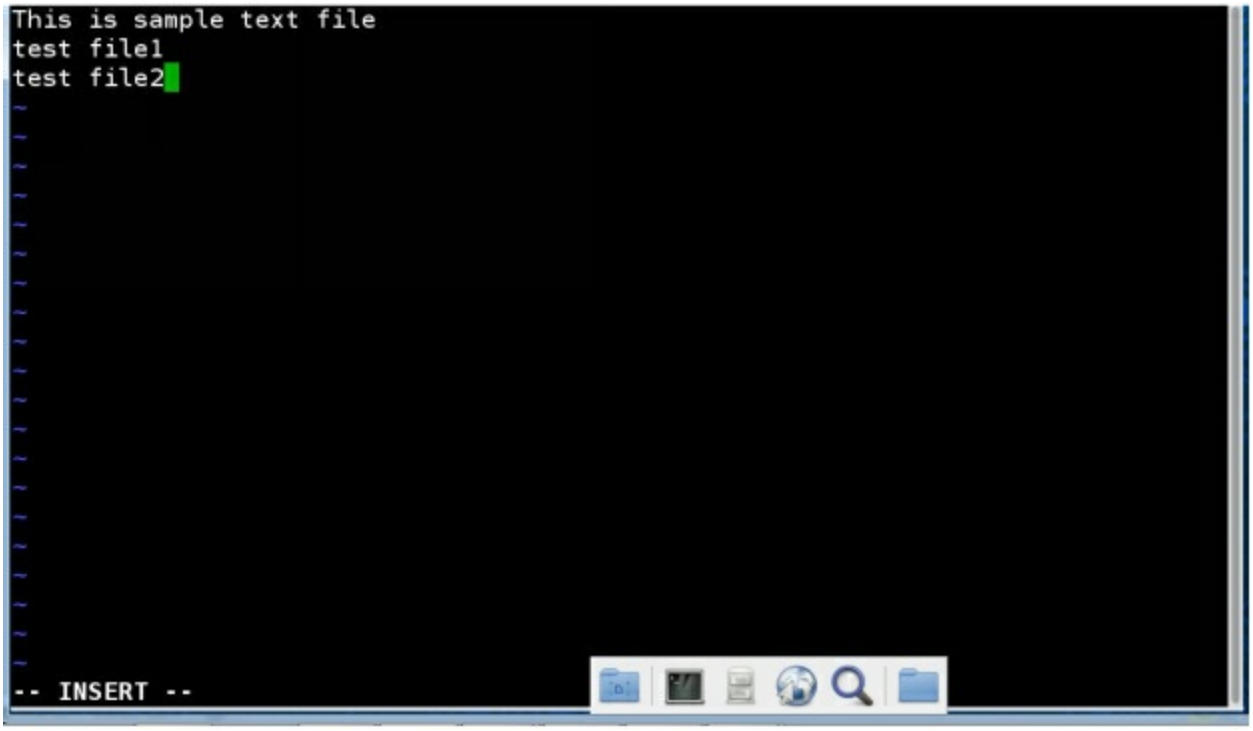
3. Last line mode

- We can use *advanced editing commands* in the last line mode.
- *To access* the last line mode, *enter a colon (:)* when *you are still in the command mode*.
- Entering the colon (**:**) character *places the cursor at the bottom line* of the screen.

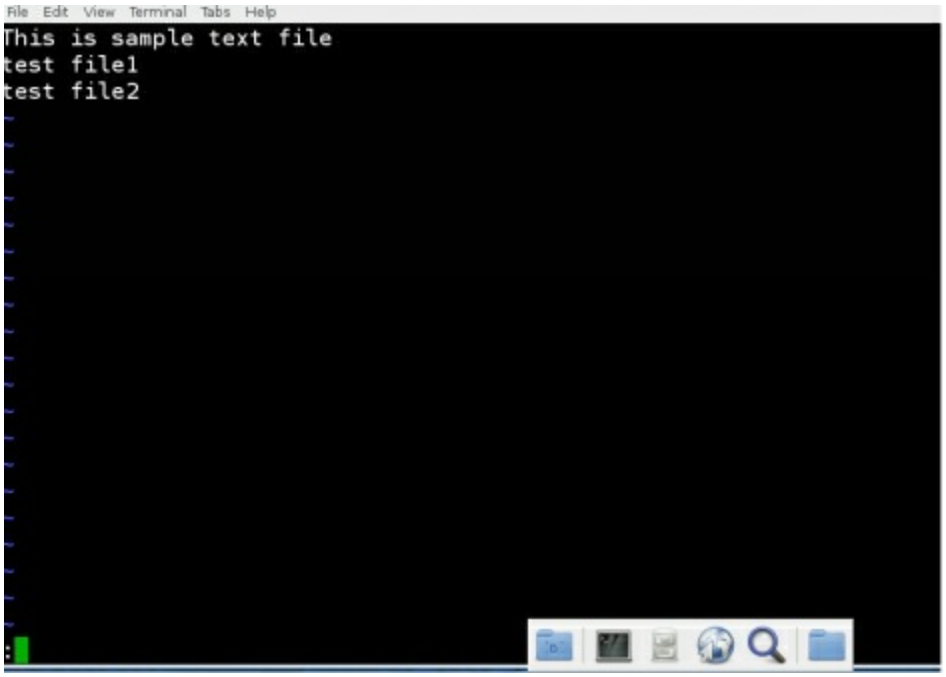


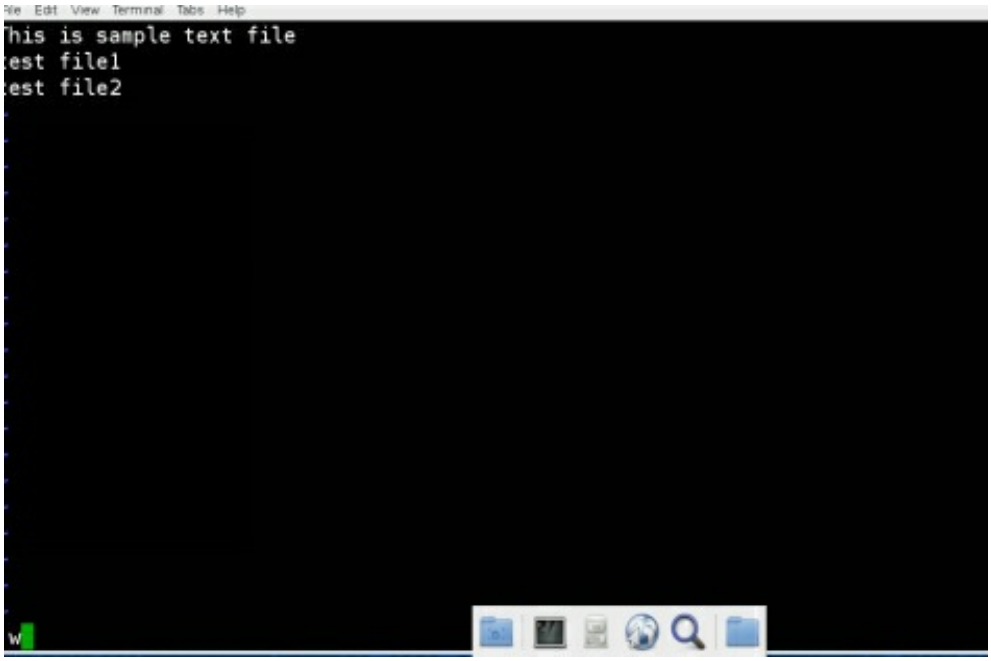
student@krosum~]$ vip1.txt {Press-Enter}

- Once we *press **i*** (insert), *the command mode* is switched *to input mode*.
- Now we can type our input text to editor. See the below snap.

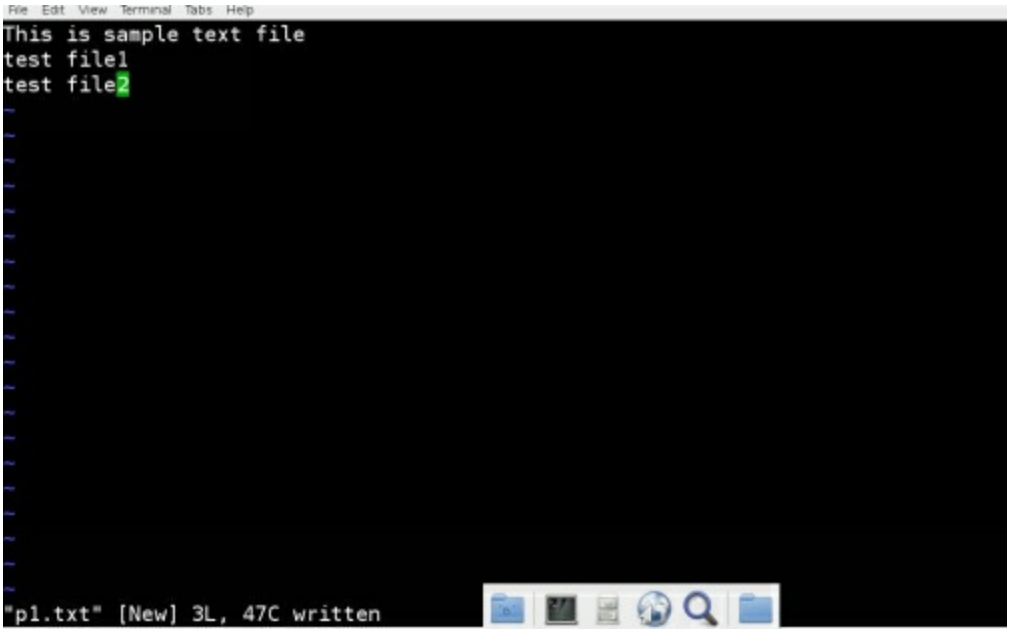


- Now we want to write our input data to storage and switch to command mode.
- Now *how to switch from input mode to command mode?* Press ESC key type shift with colon (***ESC shift :***)

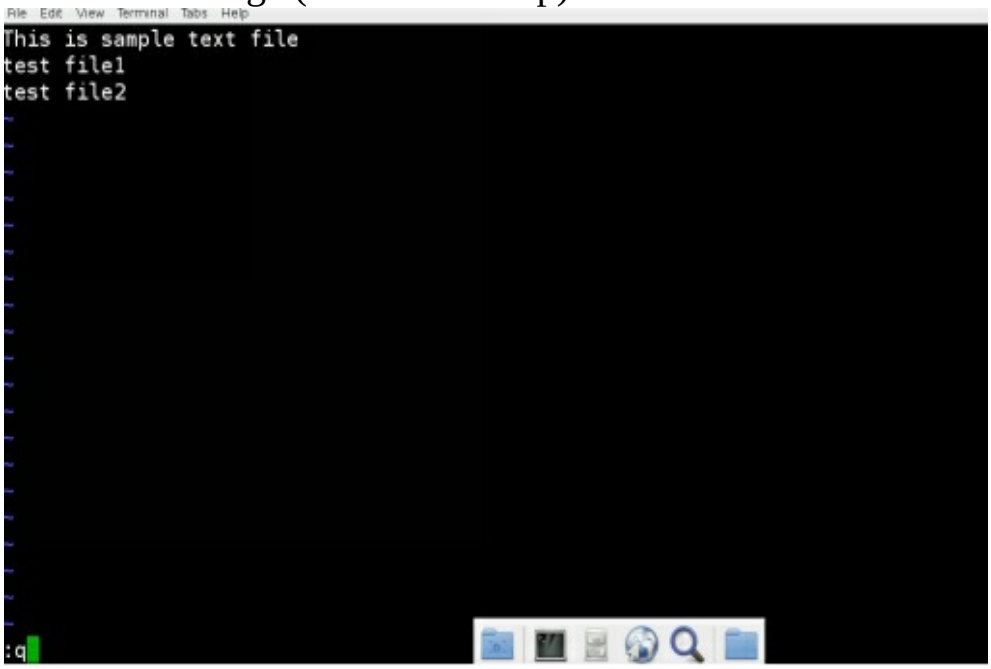


- 

  Now type lower case character ***w***(write) then press ***Enter***key



- See the above snap ***p1.txt*** file. *3 lines 47 characters* are written to storage (last line in snap).

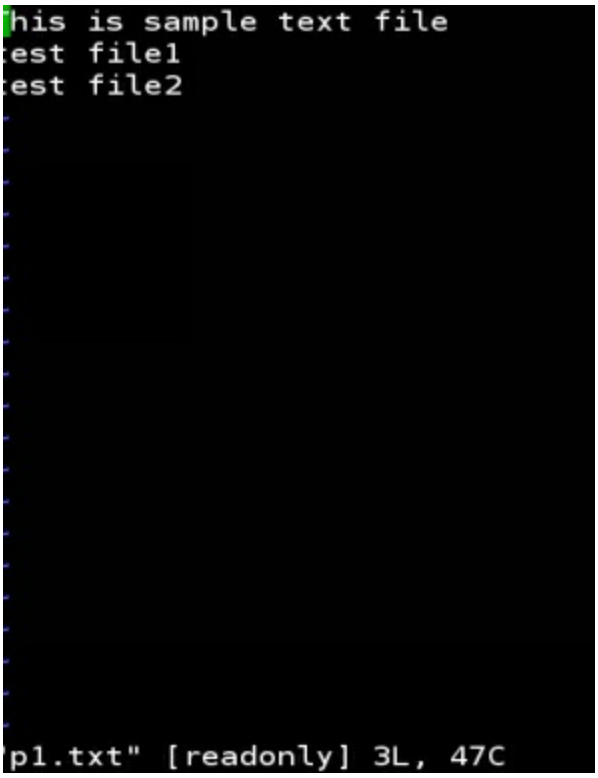


- In order to come out of the editor (***vi***), press ***q***(quit) then enter, it will switch to *exit mode (or)last line mode.*

Note: The last line mode is actually the ***ex****mode*. The ***vi*** editor is essentially a *visual extension to the* ***ex****editor,*which in turn is *an extended version of the* ***ed****editor.*

Switching Between Modes

- The *default mode* for the ***vi***editor is *the command mode.*
- *To switch to the input mode*, press ***i***, ***o***, or ***a***.
- *To return to the command mode,* press the **Esc key.**
- In the command mode, you can *save the file* and *quit* the ***vi***editor, and *return* to the shell prompt.



Viewing Files in the Read-Only Mode

- The view command enables you to view files in the read-only mode.

Syntax:-

viewfilename

- The view command invokes the ***vi*** editor in the read-only option, which means you cannot save changes to the file.
- For example, *to view the* ***p1.txt****file in the read-only mode*, enter the following command:

viewp1.txt

- The filename appears. Enter the***: q***command to *exit* the file, exit the ***vi*** editor, and *return to the shell* prompt.

Moving the Cursor within the vi Editor

The tables shows the *key sequences that move the cursor*.

KEY	SEQUENCE CURSOR MOVEMENT
h	left arrow, or Backspace -shifts *one character left*
J	down arrow - Down one line
K	up arrow - Up one line
l	right arrow, or space bar - Right (forward) one character
w	Forward one word
b	Back one word
e	To The *end of the current word*
$	To the *end of the line*
0 (zero)	To the *beginning of the line*
^	To the first non-white space character on the line

KEY	SEQUENCE CURSOR MOVEMENT
1G	Goes to the *first line of the file*
***:*n**	Goes to *Line n*
n*G*	Goes to *Line n*
Control + F	*Pages moves forward* one screen
Control +D	Scrolls *down one –half* screen
Control + B	*Pages moves backward* one screen
Control + U	Scrolls *up one-half* screen
Control + L	*Refreshes* the screen
Control +G	Displays *current buffer* information

Inserting and Appending Text

KEY	SEQUENCE CURSOR MOVEMENT
a	new or existing file by using the vi editor
A	Appends text after the cursor
i	Inserts text before the cursor
I	Inserts text at the beginning of the line
o	Opens a new line below the cursor
O	Opens a new line above the cursor
:r	File name inserts text from another file into the current file

Text-Deletion Commands

KEY	SEQUENCE CURSOR MOVEMENT
R	*Overwrites or replaces character* on the line andthe cursor position is moved pointing to next character. To terminate this operation, press Escape.
C	*Changes or overwrites sequence of characters* from the cursor to the end of the line.
s	*Substitutes a string* for a character at the cursor position.
x	*Deletes a character* at the cursor position.
dw	*Deletes a word or part* of the word to *the right of the cursor* position.
dd	*Deletes the line* containing *the cursor position*.
D	*Deletes the line* from the *cursor to the right end* of the line.
:n,md	*Delete from n^{th} line to m^{th} line* (For example, :5,10d deletes lines 5–10.)

We can use numerous commands to edit files by using the***vi***editor.

The following sections will describe the basic operations for *deleting*, *changing*, *replacing*, *copying*, and *pasting*. Remember that the***vi***editor is case-sensitive.

Note:-Output from the ***delete***command writes to a buffer from which text can be retrieved.

Edit Commands

The table describes the commands to change text, undo a change, and repeat an edit function in the ***vi***editor.

COMMAND	FUNCTION
cw	Changes or overwrites sequence of characters from the cursor position to the end of that word.
R	Replaces the character at the cursor position with another character.
J	Joins the current line and the line below
Xp	Transposes the character at the cursor position and the character to the right of the cursor position.
~	Changes the case of character placed at the cursor position from uppercase to lowercase,
u	Undo the previous command
U	Undo all changes to the current line
.	Repeats the previous command

Note:- Many of these commands change the ***vi*** editor into the *input mode*. To return to the *command mode*, press the *Esc key.*

Search and Replace Commands

The table shows the commands that search for and replace text in the ***vi*** editor.

COMMAND	FUNCTION
/string	Searches forward for the string
?string	Searches backward for the string
n	Searches for the next occurrence of the string. Use this command after searching for a string.
N	Searches for the previous occurrence of the string.
:%s/*old*/*new*/g	Searches for the old string and replaces it with the new string globally.

Copy and Paste Commands

The table shows the commands that cut, copy, and paste text in the ***vi*** editor.

COMMAND	FUNCTION
Yy	Yanks a copy of a line
P	Puts yanked or deleted text under the line containing the cursor
P	Puts yanked or deleted text before the line containing the cursor
:x,yco n	Copies lines from x to y and puts them after line n (Forexample, :1,3 co 5 copies lines 1–3 and puts themafter line 5.)
:x,y m n	Moves lines in range between x and y to line after n. For example, :4,6 m 8

Save and Quit Commands

The table describes the commands that save the text file, quit the vi editor, and return to the shell prompt.

COMMAND	FUNCTION
:w	Saves the file with changes by writing to the disk
:w **new_filename**	Writes the contents of the buffer to new_filename.
:wq	Saves the file with changes and quits the vi editor
:x	Saves the file with changes and quits the vi editor
ZZ	Saves the file with changes and quits the vi editor
:q!	Quits without saving changes
ZQ	Quits without saving changes

Session Customization

To create an *automatic customization* for all your vi sessions, perform as following:

1. *Create a file* named ***.exrc***in your home directory.
2. *Enter* any of the *set variables* into the ***.exrc***file.
3. Enter each set variable *without the preceding colon*.
4. Enter each command *on one line*.

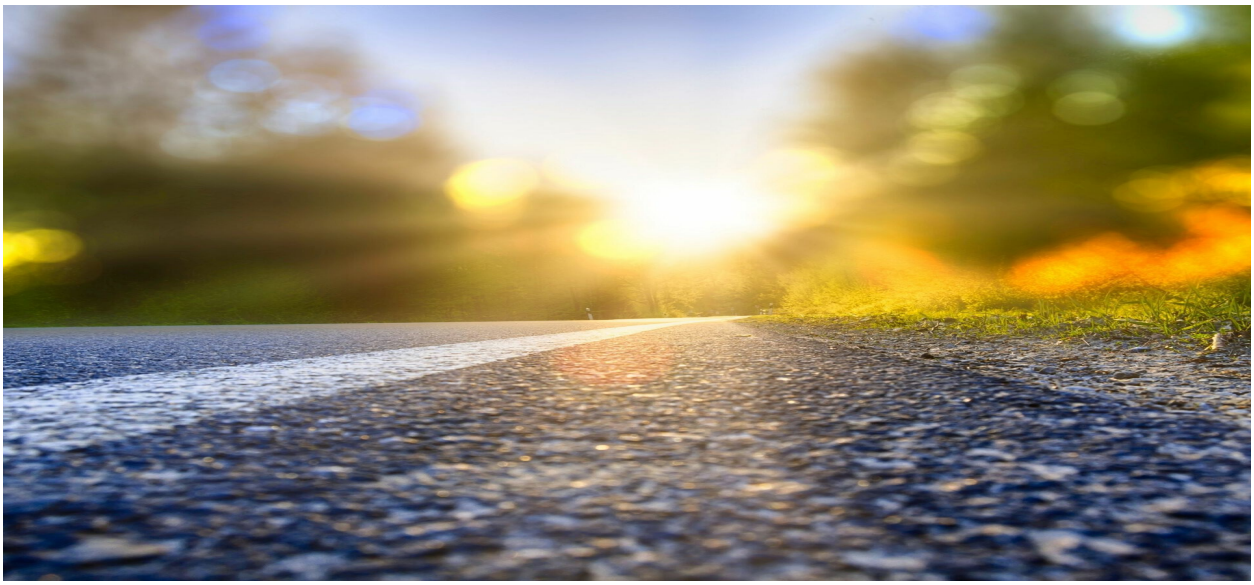
The ***vi*** editor reads the ***.exrc***file located in your home directory each time you open a ***vi*** session, regardless of your current working directory.

Note: The same steps apply for *customizing a session* in the *Vim editor*. Except that, instead of creating an ***.exrc***file, you need to create a ***.vimrc***file.

Session Customization Commands

COMMAND	FUNCTION
:set nu	Shows line numbers
:set nonu	Hides line numbers
:set ic	Instructs searches to ignore case
:set noic	Instructs searches to be case-sensitive
:set list	Displays invisible characters, such as ^I for a Tab and $ for end-of-line characters
:set nolist	Turns off the display of invisible characters
:set showmode	Displays the current mode of operation
:set noshowmode	Turns off the mode of operation display
:set	Displays all the vi variables that are set
:set all	Displays all vi variables and their current values

The table in the slide describes some of the variables of the set command.



CHAPTER 8 - DISPLAYING THE DIRECTORY CONTENT

ls –options

The **ls** command displays the content of a directory.

Syntax:- ***ls–options filename***

```
[student@krosum ~]$ pwd
/home/student
[student@krosum ~]$ ls
Desktop    Downloads  Parent1   Public     Videos
Documents  Music      Pictures  Templates
[student@krosum ~]$
```

To list the files and directories in the current directory, type ***ls*** command

without arguments.

To display the content of a **specific directory** *within the current working directory*, type ***ls*** command followed by the directory name in both ways (absolute way or relative way).

```
[student@krosum ~]$ ls
Desktop    Downloads  Parent1   Public     Videos
Documents  Music      Pictures  Templates
[student@krosum ~]$ ls Parent1
Child1  Child2  Child3
[student@krosum ~]$
[student@krosum ~]$ ls ./Parent1 # Relative Path
Child1  Child2  Child3
[student@krosum ~]$ ls /home/student/Parent1 # absolute path
Child1  Child2  Child3
[student@krosum ~]$
```

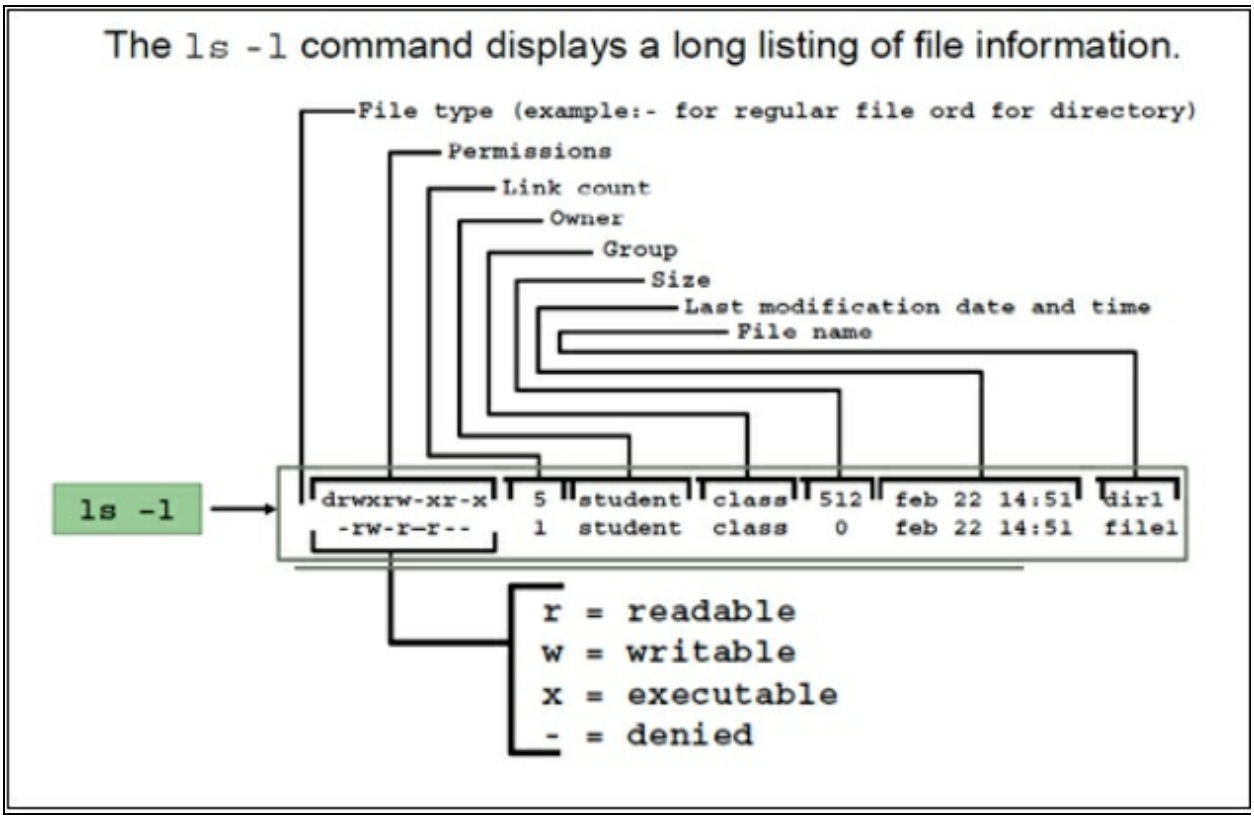
```
[student@krosum ~]$ whoami
student
[student@krosum ~]$ # student(non-root) user can list out files
[student@krosum ~]$ # ---------------- under /tmp directory
[student@krosum ~]$ ls /tmp
D1       ProjectA               systemd-private-fArlO7
ModuleB  ssh-ntAsN1DQ32lF       systemd-private-NIcOfQ
Parent2  systemd-private-blYHk6  vmware-root
[student@krosum ~]$
[student@krosum ~]$ ls /var/log/audit/
ls: cannot open directory /var/log/audit/: Permission denied
[student@krosum ~]$
[student@krosum ~]$
```

If we *don't have permission* to list out files from a specific directory, then *an error message* will be received as *permission denied.*

Note: If input directory *is not present* in current working directory location, enter the ***ls*** command with the *complete path of input directory.*

1. ls –l

The ***ls-l***command displays a long listing of file information.



The following is a brief explanation of the parts of the long list displayed in the snap:

- The *first character* is the ***file type.(***refer*file permission for more details*)

- The *second ninecharacters* indicate the file permissions: ***r*** means ***readable,w*** means***writable***, ***x*** means ***executable***, and the **–** means denied.
- The *third section* (as single number) is the ***link count.***
  - The *fourth section* is the***owner***(student).
- The *fifth* section is the ***group***(class).
- The *sixth* section is the ***file size.***
- The *seventh* section is the ***date.***
- The *eighth* section is the ***file name.***

2. ls –a

- The ***ls-a***command lists *all files* in a directory, *including hidden files*.

```
[student@krosum Parent1]$ ls
Child1  Child2  Child3  p1.c  process.log
[student@krosum Parent1]$ ls -a
.  ..  Child1  Child2  Child3  p1.c  process.log
[student@krosum Parent1]$
[student@krosum Parent1]$ mkdir .Demo # creating hidden directory
[student@krosum Parent1]$ ls
Child1  Child2  Child3  p1.c  process.log
[student@krosum Parent1]$ ls -a
.  ..  Child1  Child2  Child3  .Demo  p1.c  process.log
[student@krosum Parent1]$
```

- Some files are restricted to be displayed on the list, given by the usual format of ***ls***command.
- Hidden files often contain information that customizes your working environment.
- You can use the ***ls-a***command to list all files in a directory, including the hidden files.

Note:A single period (**.**) represents the current working directory. The double period (**..**) represents the parent directory, which itself contains the current working directory.

3. ls –t

Syntax:-

```
[student@krosum Parent1]$ ls
Child1  Child2  Child3  p1.c  process.log
[student@krosum Parent1]$
[student@krosum Parent1]$ ls -t
p1.c  process.log  Child3  Child2  Child1
[student@krosum Parent1]$
```

ls–t option (time - sort by modification time, newest first)

```
[student@krosum Parent1]$ ls
Child1  Child2  Child3  p1.c  process.log
[student@krosum Parent1]$ ls -a # list all hidden files
.  ..  Child1  Child2  Child3  .Demo  p1.c  process.log
[student@krosum Parent1]$
[student@krosum Parent1]$ ls -t # recently modified time
p1.c  process.log  Child3  Child2  Child1
[student@krosum Parent1]$
[student@krosum Parent1]$ ls -ta # -a all -t modified time
.  .Demo  p1.c  process.log  Child3  Child2  Child1  ..
[student@krosum Parent1]$
[student@krosum Parent1]$ ls -at # ls -ta and ls -at both are same
.  .Demo  p1.c  process.log  Child3  Child2  Child1  ..
[student@krosum Parent1]$
```

4. ls -r

Syntax:-

ls-r (reverse order)

```
[student@krosum ~]$ ls
Desktop    Downloads  Parent1   Public     Videos
Documents  Music      Pictures  Templates
[student@krosum ~]$ ls -r
Videos     Public     Parent1  Downloads  Desktop
Templates  Pictures   Music    Documents
[student@krosum ~]$
```

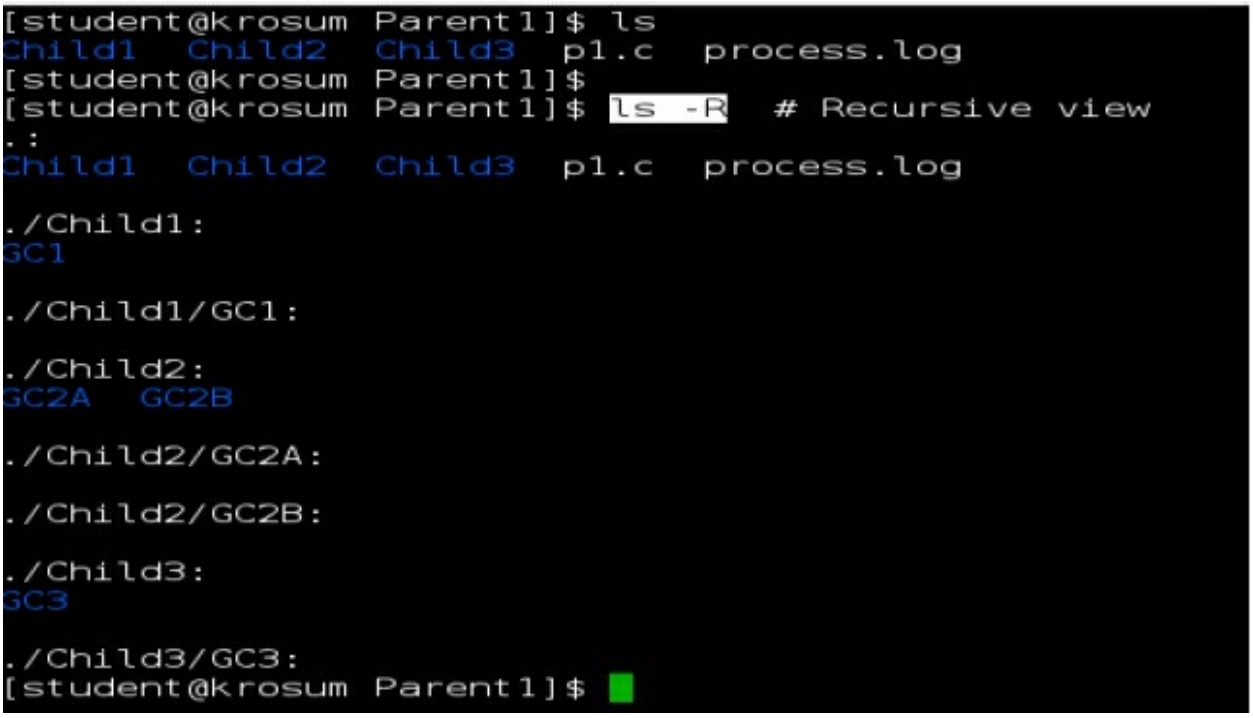
Default ***ls***command will display the list of files in *alphabetical order*. The option ***–r***shows the list of files in ***reverse order.***

Note - **reverse order**
: ***r*** **list**
- **recursive list**
R

5. ls -R

Syntax:-

ls-R **(Recursive list)**

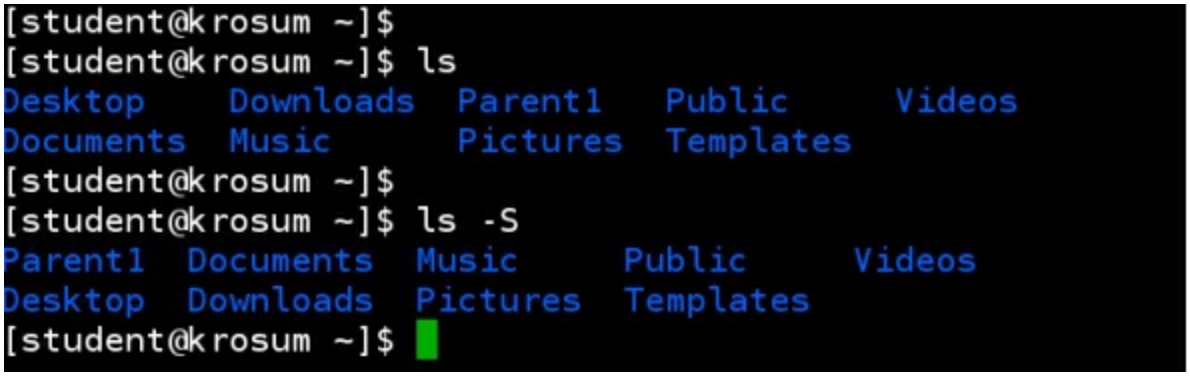


This option lists the *contents of all directories recursively*, below its corresponding current directory.

6. ls-S

Syntax:-

ls-S This option helps to *sort files by their size.*



7. Combine multiple options

Combining multiple options ***–r*** and***–l*** together (***-rl*** (or) ***–lr***) shows the **long list in reverse** order.

```
[student@krosum ~]$ ls -lr
total 0
drwxr-xr-x. 2 student student  6 Sep 23  2017 Videos
drwxr-xr-x. 2 student student  6 Sep 23  2017 Templates
drwxr-xr-x. 2 student student  6 Sep 23  2017 Public
drwxr-xr-x. 2 student student  6 Sep 23  2017 Pictures
drwxr-xr-x. 2 student student  6 Sep 23  2017 Music
drwxr-xr-x. 2 student student  6 Sep 23  2017 Downloads
drwxr-xr-x. 2 student student  6 Sep 23  2017 Documents
drwxr-xr-x. 5 student student 46 Mar 20 00:42 Desktop
[student@krosum ~]$
[student@krosum ~]$ ls -rl
total 0
drwxr-xr-x. 2 student student  6 Sep 23  2017 Videos
drwxr-xr-x. 2 student student  6 Sep 23  2017 Templates
drwxr-xr-x. 2 student student  6 Sep 23  2017 Public
drwxr-xr-x. 2 student student  6 Sep 23  2017 Pictures
drwxr-xr-x. 2 student student  6 Sep 23  2017 Music
drwxr-xr-x. 2 student student  6 Sep 23  2017 Downloads
drwxr-xr-x. 2 student student  6 Sep 23  2017 Documents
drwxr-xr-x. 5 student student 46 Mar 20 00:42 Desktop
[student@krosum ~]$
```

```
[student@krosum ~]$ ls -l /var/log/boot.log
-rw-r--r--. 1 root root 8844 Mar 20 01:52 /var/log/boot.log
[student@krosum ~]$
[student@krosum ~]$ ls -lh /var/log/boot.log
-rw-r--r--. 1 root root 8.7K Mar 20 01:52 /var/log/boot.log
```

The option –***l*** along with -***h***(human understand) format

-doption *displays only the directory details.*

```
[student@krosum ~]$ ls
Desktop    Downloads  p1.c       process.log  Templates
Documents  Music      Pictures   Public       Videos
[student@krosum ~]$ ls Desktop
DEMO2  ModuleA  ModuleB
[student@krosum ~]$ ls -l Desktop
total 0
drwxrwxr-x. 2 student student  6 Mar 20 00:08 DEMO2
drwxrwxr-x. 5 student student 33 Mar 20 00:42 ModuleA
drwxrwxr-x. 5 student student 33 Mar 19 23:54 ModuleB
[student@krosum ~]$
[student@krosum ~]$ ls -ld Desktop
drwxr-xr-x. 5 student student 46 Mar 20 00:42 Desktop
[student@krosum ~]$
```

8.

```
[student@krosum ~]$ ls
ab.sh  Desktop    Downloads  p1.c  Pictures    Public    Videos
chat   Documents  Music      p2.c  process.log  Templates
[student@krosum ~]$ ls -F
ab.sh*  Desktop/    Downloads/  p1.c   Pictures/    Public/    Videos/
chat|   Documents/  Music/      p2.c@  process.log  Templates/
[student@krosum ~]$
```

ls –F

The option ***-F*** displays list of files, indicating their file types

Refer to the table given below for more details about file types indication.

*	**Executable file**
|	**Named pipe (or) FIFO File**
/	**Directory**
@	**Symbolic file**

9. ls -i

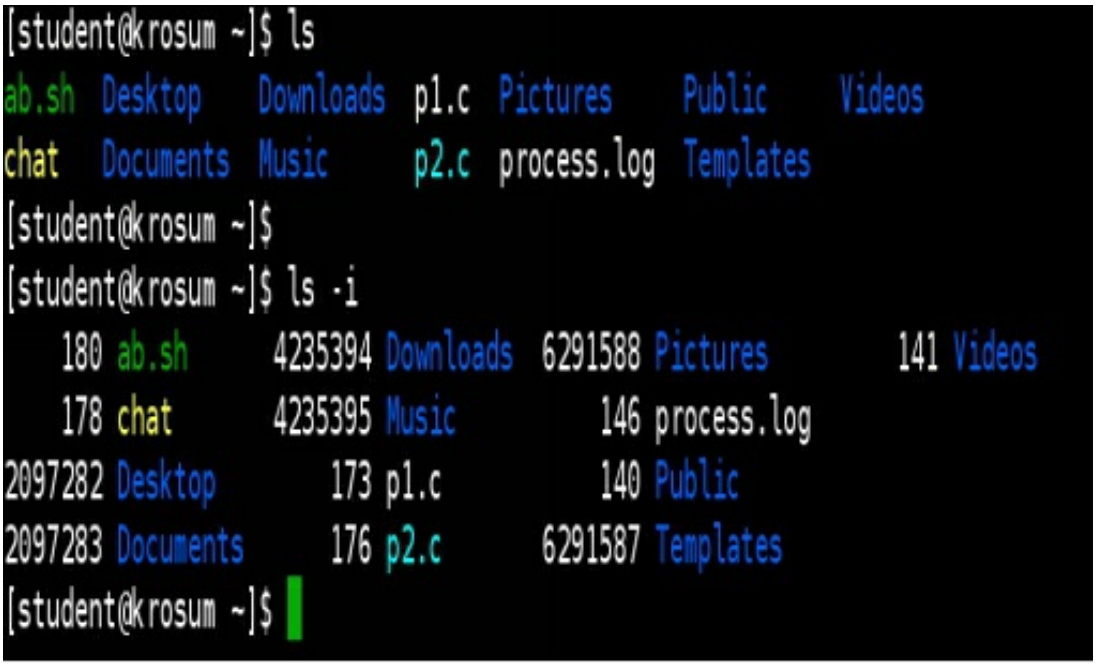
The option ***–i*** displays list of files with their corresponding file index number.

How to list recently created files in long list?

```
[student@krosum ~]$ ls -lt
total 12
-rw-rw-r--. 1 student student  84 Mar 20 18:31 process.log
-rwxrwxr-x. 1 student student   1 Mar 20 18:25 ab.sh
prw-rw-r--. 1 student student   0 Mar 20 18:23 chat
lrwxrwxrwx. 1 student student   4 Mar 20 18:23 p2.c -> p1.c
-rw-rw-r--. 1 student student 168 Mar 20 18:13 p1.c
drwxr-xr-x. 5 student student  46 Mar 20 00:42 Desktop
drwxr-xr-x. 2 student student   6 Sep 23  2017 Documents
drwxr-xr-x. 2 student student   6 Sep 23  2017 Music
drwxr-xr-x. 2 student student   6 Sep 23  2017 Pictures
drwxr-xr-x. 2 student student   6 Sep 23  2017 Public
drwxr-xr-x. 2 student student   6 Sep 23  2017 Templates
drwxr-xr-x. 2 student student   6 Sep 23  2017 Videos
drwxr-xr-x. 2 student student   6 Sep 23  2017 Downloads
[student@krosum ~]$
```

- We can use both ***–l***and***-t***options irrespective of their order.

Both ***ls–lt*** and***ls -tl***has the same effect.


[student@krosum ~]$ ls
ab.sh Desktop Downloads p1.c Pictures Public Videos
chat Documents Music p2.c process.log Templates
[student@krosum ~]$
[student@krosum ~]$ ls -i
180 ab.sh 4235394 Downloads 6291588 Pictures 141 Videos
178 chat 4235395 Music 146 process.log
2097282 Desktop 173 p1.c 140 Public
2097283 Documents 176 p2.c 6291587 Templates
[student@krosum ~]$

How to view?

These are the following ways in viewing the list of files.

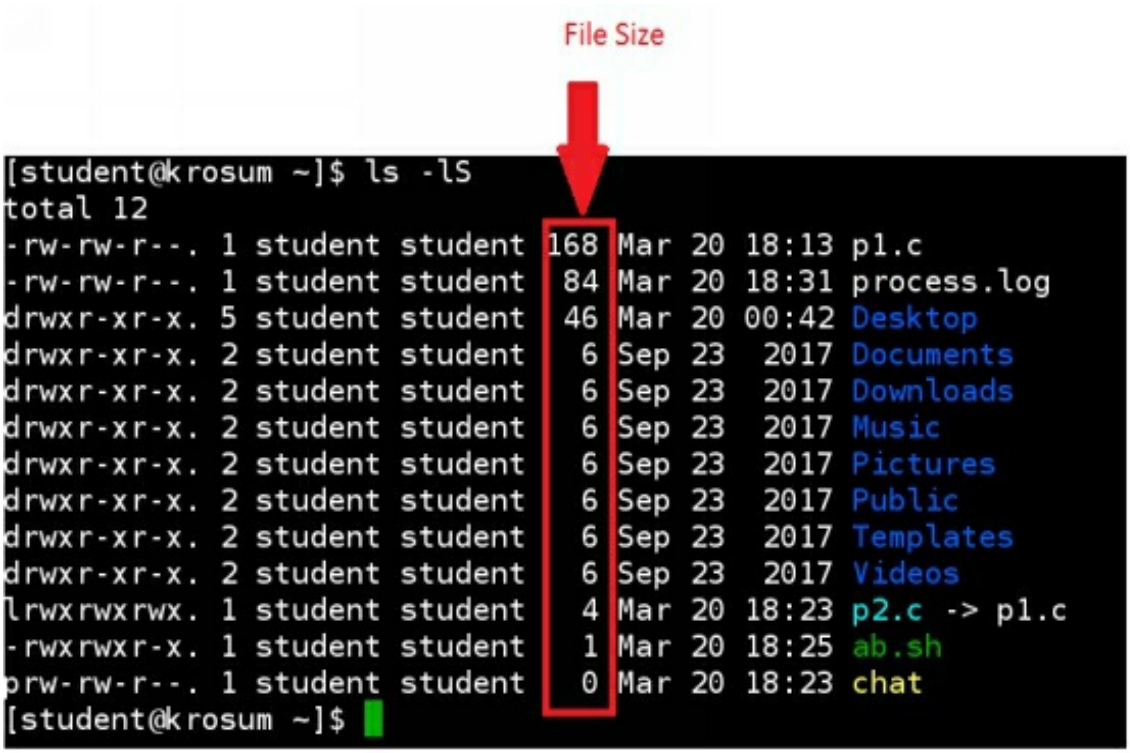
- **Long list of files in recursive order.**

ls–lR(or) ***ls-Rl***

- **Recently created files in long list of recursive order.**

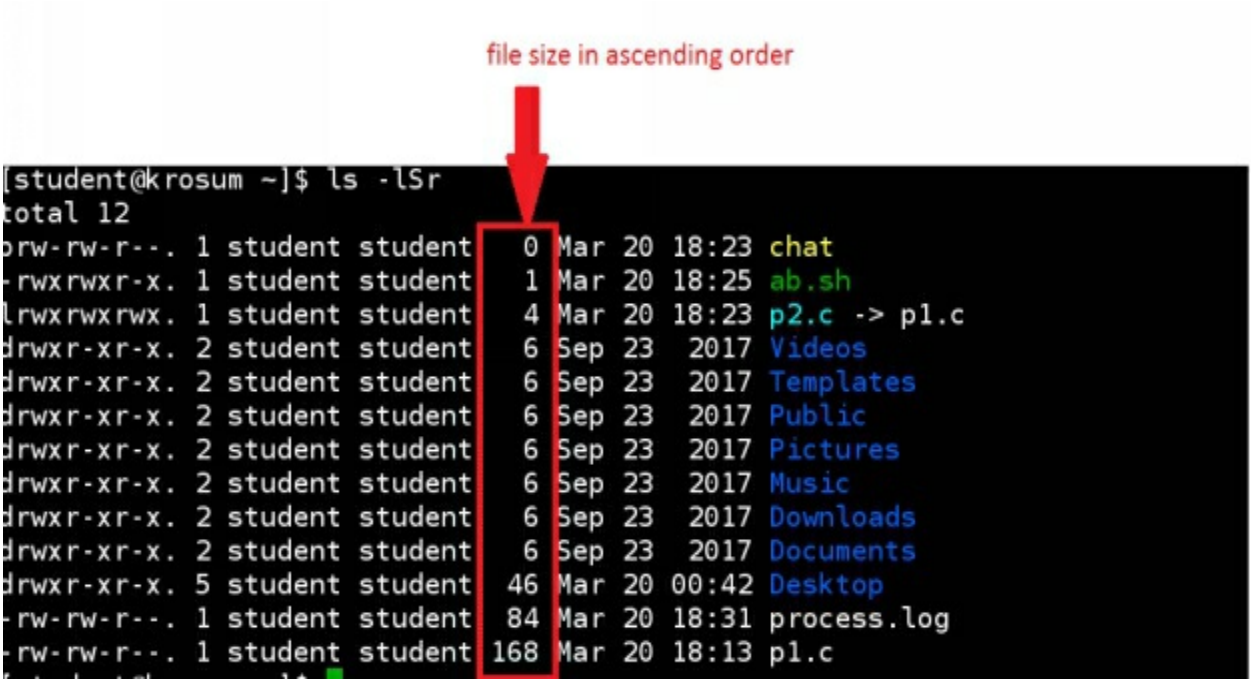
ls-ltR(or) ***ls-tRl***(or) ***ls-lRt***

- **List of files (long list) in descending (large to small size) order.**



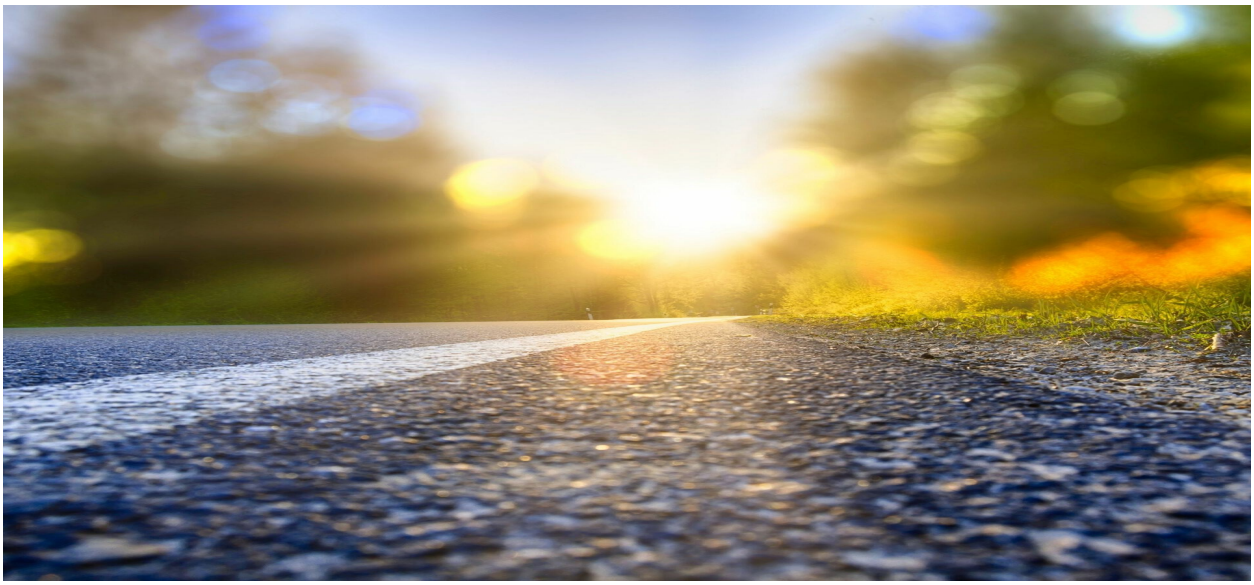
ls-lS# Upper case S

- **List of files (long list) in ascending (small file size to large size) order**



***ls-lrS*# Upper case S # *ls–lSr*(reverse order)**

OPTION	USAGE
-a	List all the files, including hidden files (prefixed with .)
-d	List directory entries
-F	File Types
-l	Long list
-t	Recently modified time
-h	Human readable format
-i	Index number
-r	Reverse order
-R	Recursive order
-S	Sorted order
-s	Block size



CHAPTER 9 - REGULAR FILE MANIPULATION COMMANDS

So far we know how to perform directory manipulation actions. Now let's discuss about regular file manipulation commands.

In Linux, *Regular file* (or) *Ordinary file* is classified in to *two* types.

1. **ASCII** (or) **TEXT File**- user ***can*** read (or) understand the format.
2. **ELF** - user **can't** read (or) understand .

How to create a new ASCII file?

There are many ways we can *create a ASCII File.*

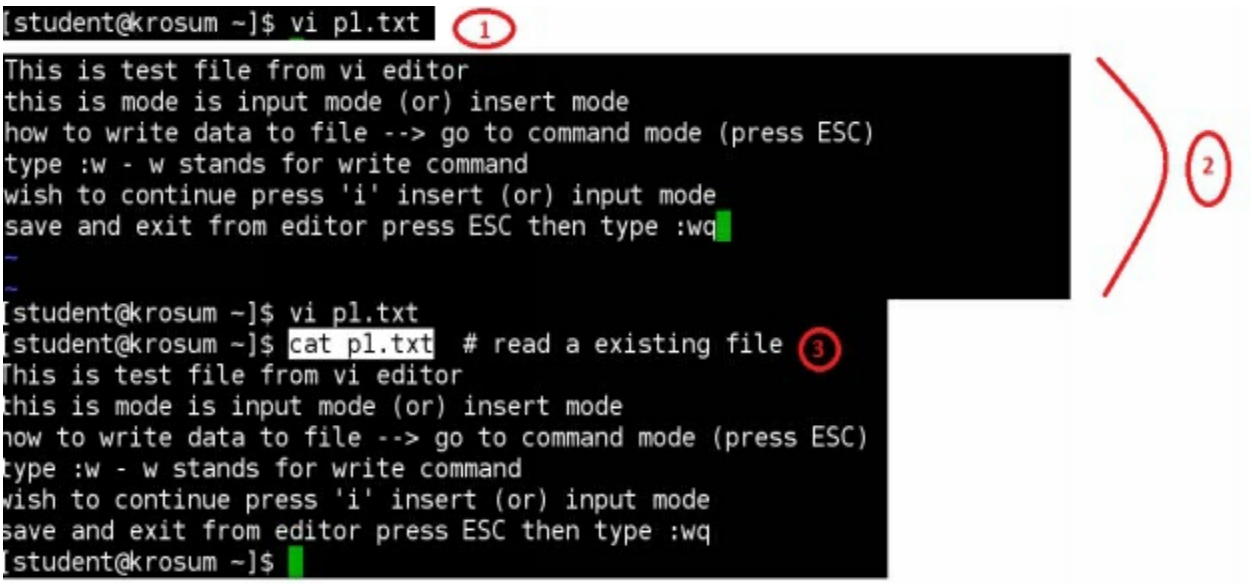
1. Using **editor**
2. Using **redirection symbols (<>>>)**

First let’s discuss about editor. Linux supports command line editors like vim, nanoetc., and GUI editors like gedit, leafpad, bluefish. Now we will discuss about vi editor.

Recap previous chapter (vi editor) on how to use vi editor commands.

cat command

cat command *is read-only command* ; we can read existing ASCII file from Linux command line.



Task : create a new file named p1.txt. Write some text in the editor, save a file to storage and then read input file using cat command.

```
[student@krosum ~]$ vi ab.txt
echo "Welcome to Linux Command line"
echo "Today date:`date +%D`"
echo "Current working kernel is:`uname`"
~
[student@krosum ~]$ cat ab.txt
echo "Welcome to Linux Command line"
echo "Today date:`date +%D`"
echo "Current working kernel is:`uname`"
[student@krosum ~]$
```

Task :Create a new file name called *ab.txt*. Write following shell commands in ab.txt file and display content to monitor.

```
[student@krosum ~]$ vi p1.c
```

```
#include<stdio.h>
int main(){
        printf("Hello Linux\n");
        return 0;
}
~
```

```
[student@krosum ~]$ cat p1.c
#include<stdio.h>
int main(){
        printf("Hello Linux\n");
        return 0;
}
[student@krosum ~]$
```

Task :How to write c program in Linux command line?

```
[student@krosum ~]$ cat p1.c # with out using -n option
#include<stdio.h>
int main(){
        printf("Hello Linux\n");
        return 0;
}
[student@krosum ~]$ cat -n p1.c # with -n option
     1  #include<stdio.h>
     2  int main(){
     3          printf("Hello Linux\n");
     4          return 0;
     5  }
[student@krosum ~]$
```

Using cat command, we can read the existing ASCII file. If you wish viewing with *line number* use **–n** option.

```
[student@krosum ~]$ cat /var/log/boot.log
```

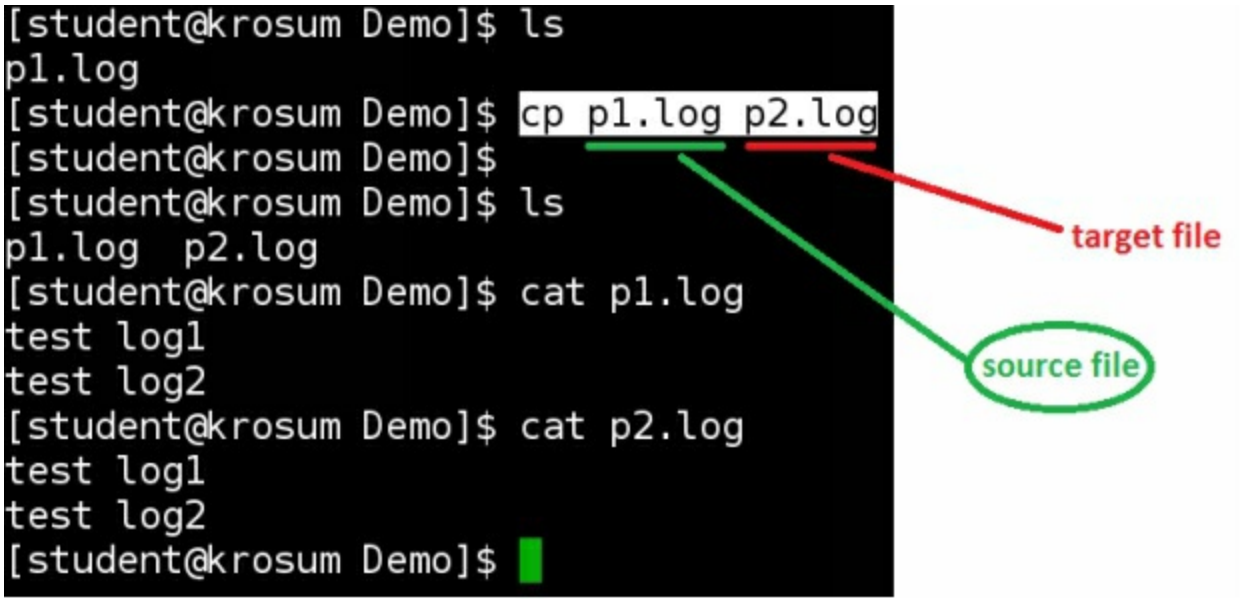
Task :How to read existing /var/log/boot.log file ?

Note : cat command is not a directory manipulation command.

Copy (cp) command

- The copy (cp) utility *creates a duplicate* of *a file*, a *set of files*, or *a directory*.
- If the source is a file, the new file (target file) contains *exact copy of the data* in the source file.

Syntax ***cp-option SourcefileTargetfile***



- If *source file doesn't exist* (or) proper *path* is *not specified* (or) *case mismatch* happens (upper/lower), ***cp action is failed.*** Refer to the below snap for error messages.

```
[student@krosum Demo]$ ls
p1.log  p2.log
[student@krosum Demo]$ cp ab.txt T1.c  ## source ab.txt file is not exists
cp: cannot stat ‘ab.txt’: No such file or directory
[student@krosum Demo]$
[student@krosum Demo]$ cp P1.log T2.log ## source file P1.log (uppercase)
cp: cannot stat ‘P1.log’: No such file or directory
[student@krosum Demo]$
[student@krosum Demo]$ cp passwd T3.log ## passwd file is not exists
cp: cannot stat ‘passwd’: No such file or directory
[student@krosum Demo]$
[student@krosum Demo]$ cp /etc/passwd backup.conf ## this is valid
```

```
[student@krosum Demo]$ ls
p1.log  p2.log  test1.c
[student@krosum Demo]$ cat -n p1.log
     1  test log1
     2  test log2
[student@krosum Demo]$ cat -n test1.c
     1  #include<stdio.h>
     2  int main(){
     3          printf("Hello Linux\n");
     4          return 0;
     5  }
[student@krosum Demo]$ cp p1.log test1.c # Note: target file(test1.c) is already
                                        #       exists(overwrite)
[student@krosum Demo]$ cat -n test1.c
     1  test log1
     2  test log2
[student@krosum Demo]$
```

If target *file already exists* , the exiting file *content will overwrite.* See the below example

- The test1.c file is *already exists* and hence its contents are replaced by the source file (p1.log) contents.
- The **cp**command copies both text and binary files.

```
[student@krosum Demo]$ ls
p1.log  p2.log  repo  test1.c
[student@krosum Demo]$ file repo # determine file type
repo: directory
[student@krosum Demo]$
[student@krosum Demo]$ cp p1.log repo # cp source(file)  target(directory)
[student@krosum Demo]$
[student@krosum Demo]$ ls repo # under repo directory
p1.log
[student@krosum Demo]$
```

If the target is directory, the source file is copied to the directory. See the below snap where p1.log file is copied to repo directory.

Under the repo directory we can duplicate new name called test1.log

```
[student@krosum Demo]$ ls
p1.log  p2.log  repo  test1.c
[student@krosum Demo]$ ls repo
p1.log
[student@krosum Demo]$ cp p1.log repo/test1.log
[student@krosum Demo]$
[student@krosum Demo]$ ls repo
p1.log  test1.log
[student@krosum Demo]$
```

We can use **wild card**to *copymore than one file* into subdirectory or remote directory.

```
[student@krosum Demo]$ ls
p1.log  p2.log  process.log  repo  test1.c  vmreport.log
[student@krosum Demo]$
[student@krosum Demo]$ # copy list of log files into /tmp directory
[student@krosum Demo]$ # ----         ==========     ----
[student@krosum Demo]$ ls /tmp
D1       Parent2   ssh-ntAsN1DQ32lF        systemd-private-fArlO7  vmware-root
ModuleB  ProjectA  systemd-private-blYHk6  systemd-private-NIcOfQ
[student@krosum Demo]$ cp *.log /tmp
[student@krosum Demo]$
[student@krosum Demo]$ ls /tmp # after copied
D1       Parent2           systemd-private-blYHk6  vmware-root
ModuleB  process.log       systemd-private-fArlO7
p1.log   ProjectA          systemd-private-NIcOfQ
p2.log   ssh-ntAsN1DQ32lF  vmreport.log
[student@krosum Demo]$
```

Rules in file copy:

To copy a file successfully, several *rules* must be followed:

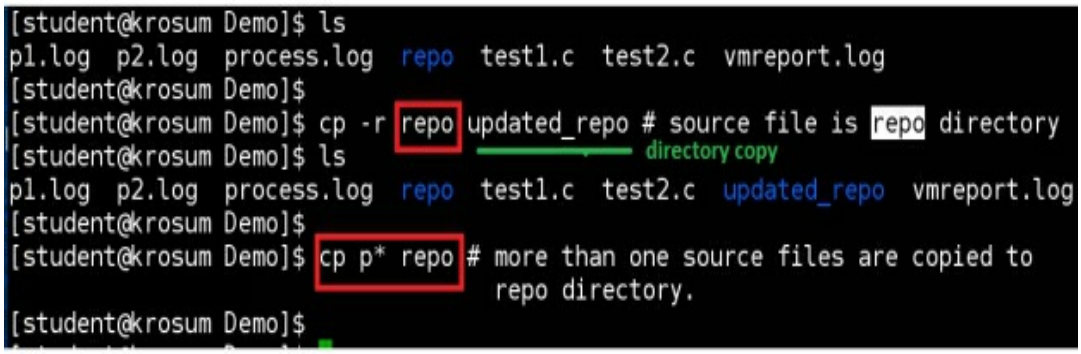
- Rule 1 : The *source must exist.* Otherwise, Linux prints the following error message:

```
[student@krosum Demo]$ ls
p1.log  p2.log  process.log  repo  test1.c  vmreport.log
[student@krosum Demo]$
[student@krosum Demo]$ cp result.log out.log # There is no result.log
cp: cannot stat ‘result.log’: No such file or directory
[student@krosum Demo]$
```

<sourcefile> - No such file (or) directory.

- Rule 2: If *no destination path* is specified, Linux *assumes* the destination as the *current* working directory.

```
[student@krosum Demo]$ ls
p1.log  p2.log  process.log  repo  test1.c  vmreport.log
[student@krosum Demo]$
[student@krosum Demo]$ cp test1.c test2.c # test2.c file will create
                                          currentdirectory
[student@krosum Demo]$ ls
p1.log  p2.log  process.log  repo  test1.c  test2.c  vmreport.log
[student@krosum Demo]$
```

- 

  Rule 3: If the *source* happens to be *multiple files* or a *directory*, then the *destination* file must be a *directory.*

- Rule 4: To *prevent an* automatic *replacement (overwrite)* of the destination file, use the ***interactive (-i) option***. When interactive is specified, Linux *issues a warning message* and *waits* for a reply. Any reply other than yes will cancel the copy of the specified file.

```
[student@krosum Demo]$ cat -n test1.c
     1  #include<stdio.h>
     2  int main(){
     3          printf("Hello\n");
     4          return 0;
     5  }
[student@krosum Demo]$ cat -n p1.log
     1  test log1
     2  test log2
[student@krosum Demo]$ cp p1.log test1.c # test1.c(target)file will overwrite
[student@krosum Demo]$ cat -n test1.c
     1  test log1
     2  test log2
[student@krosum Demo]$ cat -n test2.c
     1  #include<stdio.h>
     2  int main(){
     3          printf("Hello\n");
     4          return 0;
     5  }
```

Note, however, that if the file/destination directory is write protected, we cannot use this option to write to the directory.

```
[student@krosum Demo]$ cp -i p1.log test2.c
cp: overwrite 'test2.c'? n
[student@krosum Demo]$ cat -n test2.c # there is no changes in target file
     1  #include<stdio.h>
     2  int main(){
     3          printf("Hello\n");
     4          return 0;
     5  }
[student@krosum Demo]$
```

- Rule 5: To preserve the *modification times* and *file access permission*, use the **preserver (-p) option**. In the absence of the preserve options, the time will be the time the file was copied and the file access permissions will be the defaults.

```
[student@krosum Demo]$ ls -l p1.log
-rw-rw-r--. 1 student student 20 Mar 21 15:13 p1.log
[student@krosum Demo]$
[student@krosum Demo]$ cp p1.log p2.log
[student@krosum Demo]$ ls -l p1.log p2.log
-rw-rw-r--. 1 student student 20 Mar 21 15:13 p1.log
-rw-rw-r--. 1 student student 20 Mar 21 18:12 p2.log
[student@krosum Demo]$
[student@krosum Demo]$ cp -p p1.log p3.log # preserve attributes option
[student@krosum Demo]$
[student@krosum Demo]$ ls -l p1.log p2.log p3.log
-rw-rw-r--. 1 student student 20 Mar 21 15:13 p1.log
-rw-rw-r--. 1 student student 20 Mar 21 18:12 p2.log
-rw-rw-r--. 1 student student 20 Mar 21 15:13 p3.log
[student@krosum Demo]$
```

Note : p1.log and p3.log file preserve the time attributes.

```
[student@krosum Demo]$ ls -l r1.sh
-rw-rw-r--. 1 student student 13 Mar 21 18:43 r1.sh
[student@krosum Demo]$ ls -l r2.sh
-rwxrwxr-x. 1 student student 13 Mar 21 18:44 r2.sh
[student@krosum Demo]$
[student@krosum Demo]$ cp -p r1.sh r2.sh
[student@krosum Demo]$
[student@krosum Demo]$ ls -l r1.sh r2.sh
-rw-rw-r--. 1 student student 13 Mar 21 18:43 r1.sh
-rw-rw-r--. 1 student student 13 Mar 21 18:43 r2.sh
[student@krosum Demo]$
```

As stated in the **copy (cp) command** section, when the destination file already exists, its permissions, owner and group are used rather than the source file attributes. However, We can force the permissions, owner, and group to be changed, by using the *preserve (-p) option.*

Note how the **permissions** of **r2.sh** have been changed to match those of the source file, **r1.sh**

Move (mv) command

- Move command is used to *move* either an *individual file*, *a list of files* or *directory*. After a move, the *old file name is gone* and the new file name is found at the destination.

```
[student@krosum Demo]$ ls
ab.sh   p2.log  process.log  r2.sh  repo     test2.c       vmreport.log
p1.log  p3.log  r1.sh        r3.sh  test1.c  updated_repo
[student@krosum Demo]$ mv p1.log repo # file p1.log moved to repo directory
[student@krosum Demo]$ ls
ab.sh   p3.log       r1.sh  r3.sh  test1.c  updated_repo
p2.log  process.log  r2.sh  repo   test2.c  vmreport.log
[student@krosum Demo]$ ## there is no p1.log file under Demo directory.
[student@krosum Demo]$ ls repo
p1.log  p2.log  process.log  test1.log
[student@krosum Demo]$
```

- This is the difference between a move and a copy. After a *copy*, *the file is physically duplicated*; it exists in two places. The move format appears in below snap.

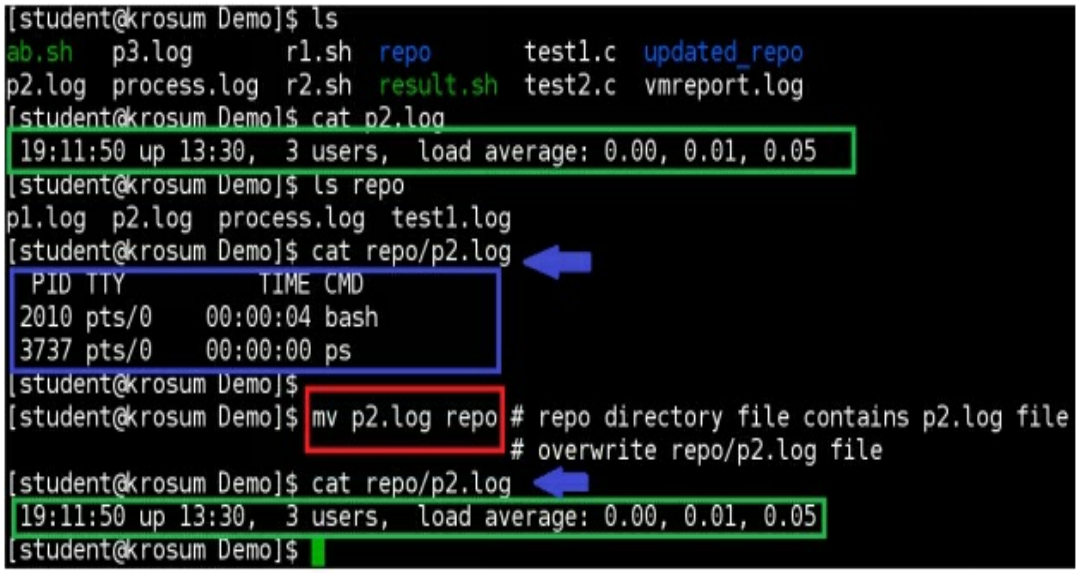
- The file p1.log is moved to repo directory - move action.
- Using mv command we can do **rename** operation.

```
[student@krosum Demo]$ ls
ab.sh   p3.log       r1.sh  r3.sh  test1.c  updated_repo
p2.log  process.log  r2.sh  repo   test2.c  vmreport.log
[student@krosum Demo]$
[student@krosum Demo]$ # mv oldfile newfile
[student@krosum Demo]$ # ================== rename action
[student@krosum Demo]$
[student@krosum Demo]$ mv r3.sh result.sh
[student@krosum Demo]$ ls
ab.sh   p3.log       r1.sh  repo       test1.c  updated_repo
p2.log  process.log  r2.sh  result.sh  test2.c  vmreport.log
[student@krosum Demo]$
```

mv options

- Move has only two options

  1. **Interactive (-i)**
  2. **Force (-f)**

- 

- If the destination file *already exists*, its *old* contents are *destroyed* unless we use the **interactive flag (-i) to***warn us* on such move operation. When the interactive flag is on, move asks if we want to destroy the existing file.

```
[student@krosum Demo]$ cat test2.c
#include<stdio.h>
int main(){
        printf("Hello\n");
        return 0;
}
[student@krosum Demo]$ cat repo/test2.c
#include<stdio.h>
int main(){
        printf("Hello\n");
        printf("This is test2 program file\n");
        printf("This is test2 program file\n");
        printf("This is test2 program file\n");
        return 0;
}
[student@krosum Demo]$
```

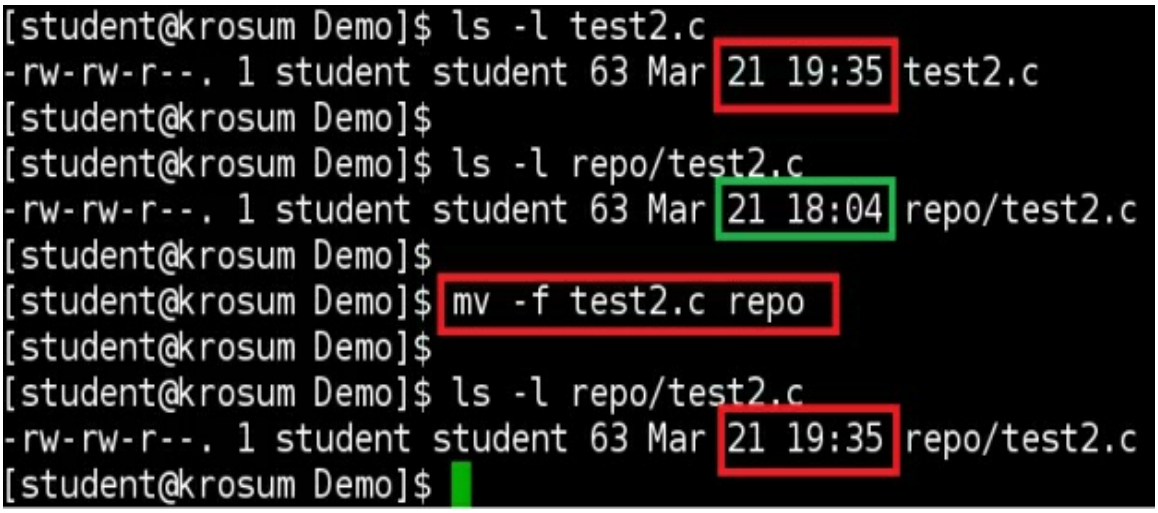
1.

```
[student@krosum Demo]$ ls
ab.sh   process.log  r2.sh  result.sh  test2.c      vmreport.log
p3.log  r1.sh        repo   test1.c    updated_repo
[student@krosum Demo]$ cat -n test2.c
     1  #include<stdio.h>
     2  int main(){
     3          printf("Hello\n");
     4          return 0;
     5  }
[student@krosum Demo]$ cat -n repo/test2.c
     1  #include<stdio.h>
     2  int main(){
     3          printf("Hello\n");
     4          printf("This is test2 program file\n");
     5          printf("This is test2 program file\n");
     6          printf("This is test2 program file\n");
     7          return 0;
     8  }
[student@krosum Demo]$ mv -i test2.c repo
[student@krosum Demo]$ mv -i test2.c repo
mv: overwrite 'repo/test2.c'? n
```

Using -i (interactive) option

See the above snap there is no modification in test2.c file placed under repo directory.

2. Force (-f) option



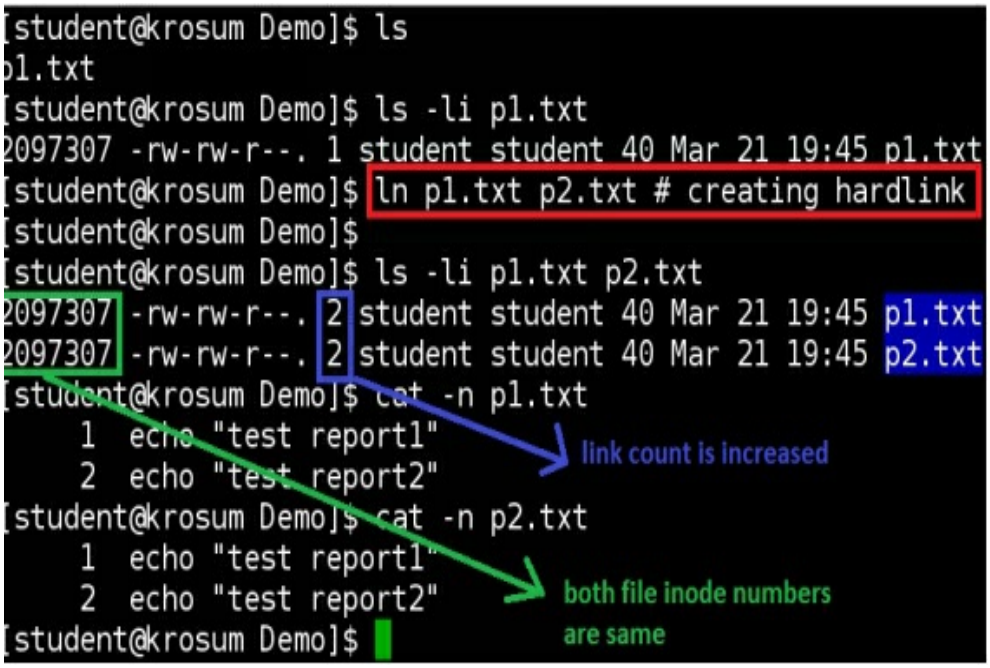
If we are sure that we want to write it, even if it already exists, we can skip the interactive message with the force (-f) option.

Link (ln) command

- The link command receives either a file or directory as input and its output is an *updated directory.*
- Two types of link in Linux

1. Hard link
2. Soft link (or Symbolic link)

1. Hard links to Files

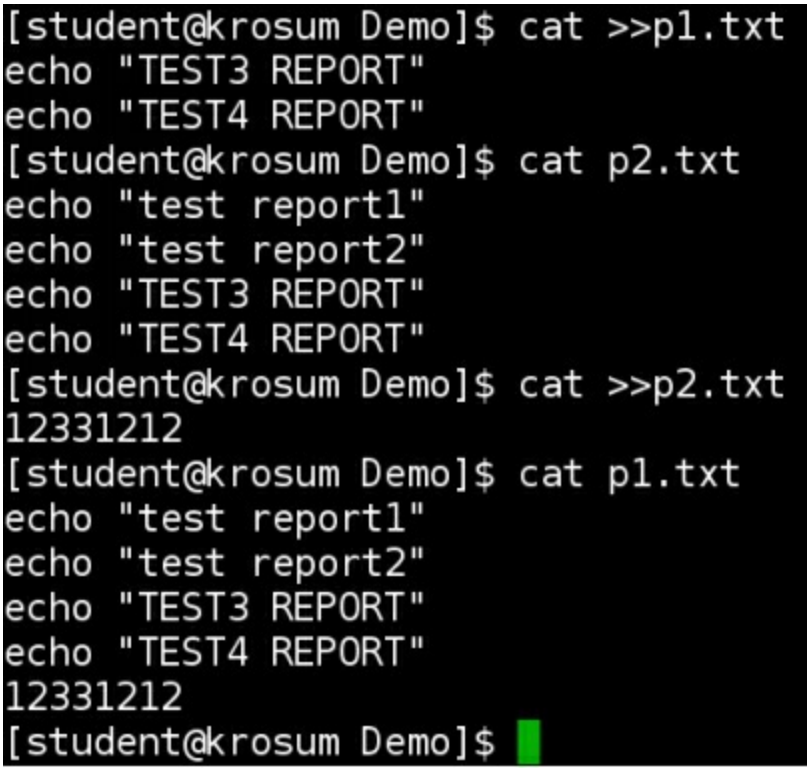
- To create a hard link to a file, we specify the source file and destination file.
- If the destination file *doesn't exist,* it is *created.*
- 

- If it *exists*, it is first *removed* and then *re-created* as a *linked file*.

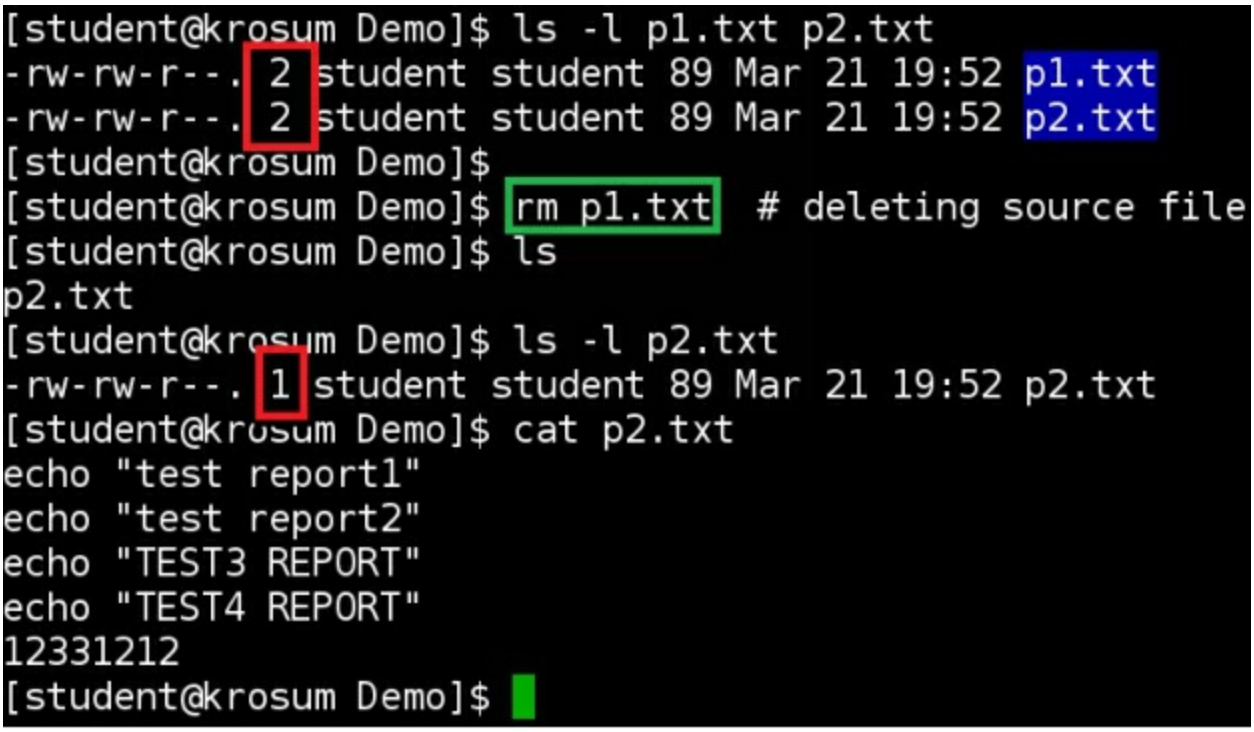
- Note in both file (p1.txt and p2.txt) *inode* (index number) is same.
- See the below snap where we have appended TEST3 REPORT TEST4 REPORT content to p1.txt file and this is

automatically updated in p2.txt file.

- Similarly if we append numbers to p2.txt file, then it is automatically updated to p1.txt file.

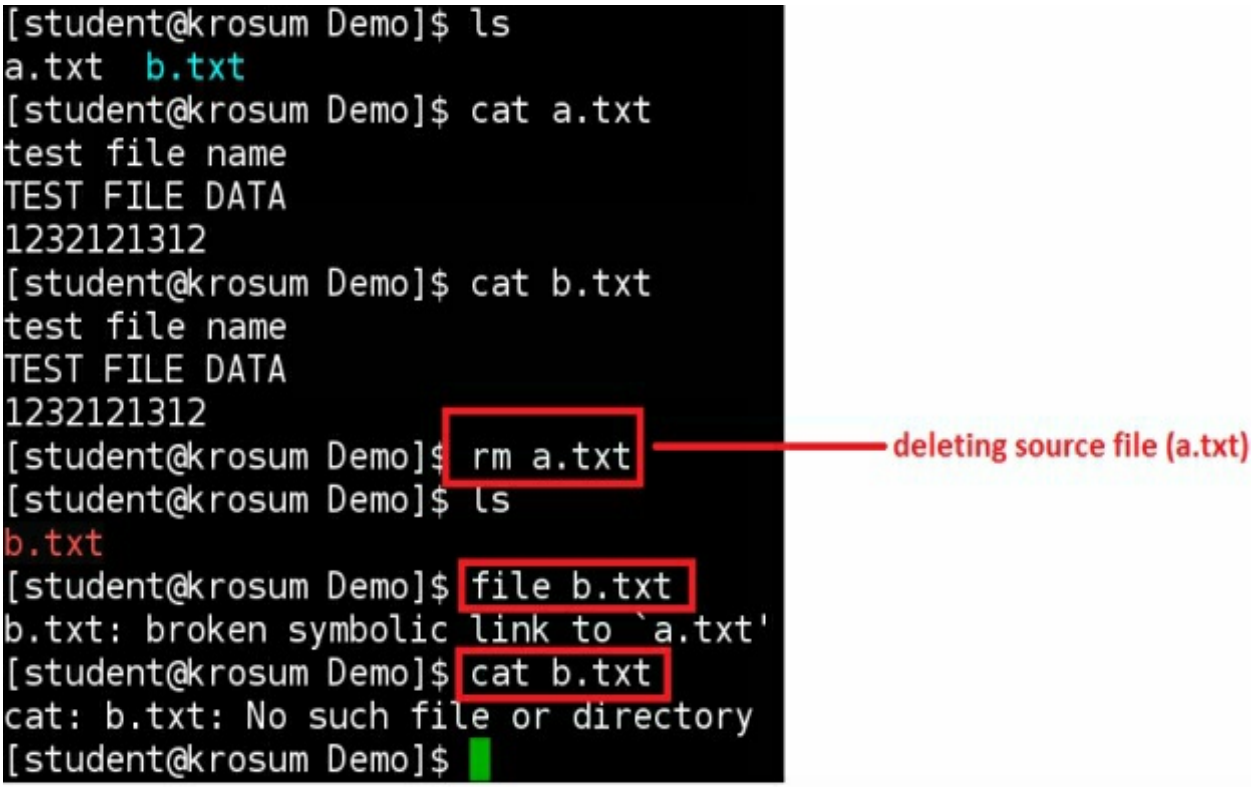


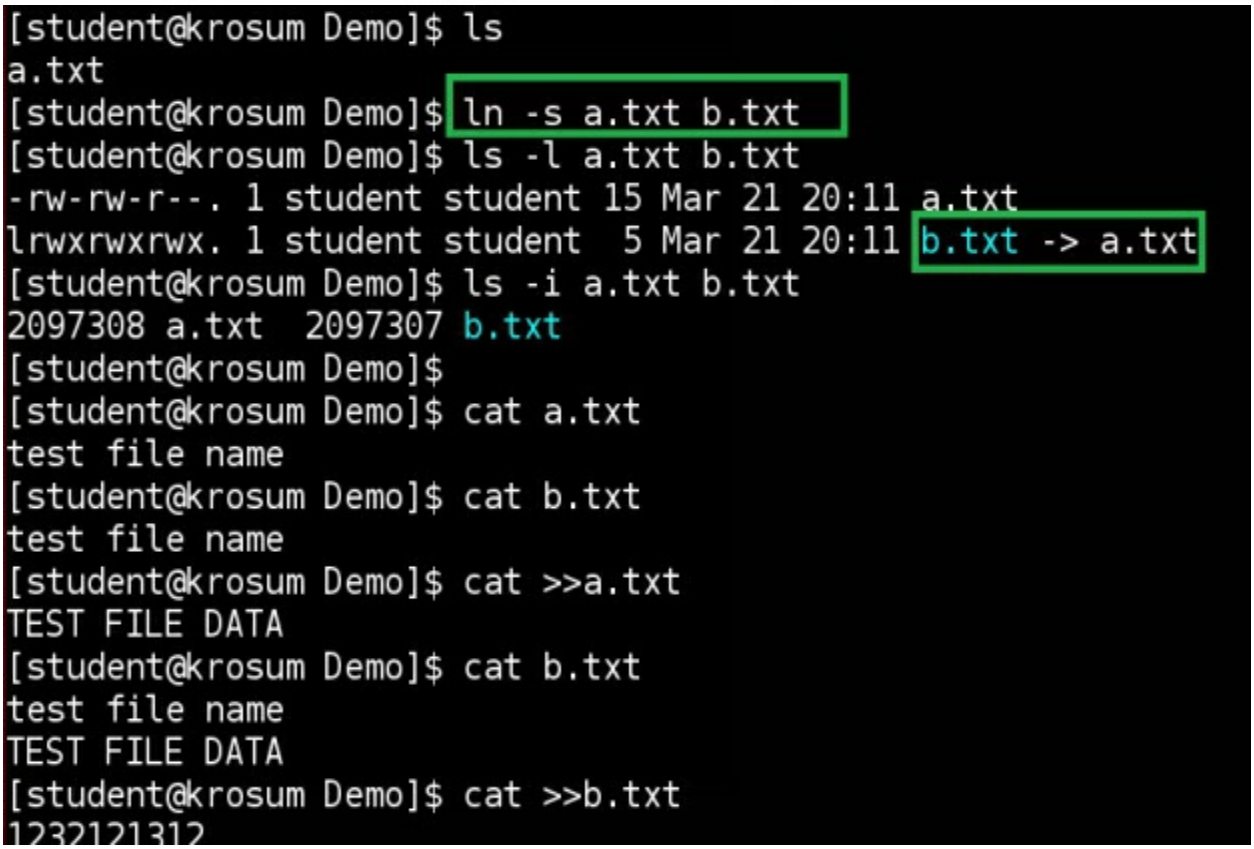
- If we *delete source* file (p1.txt), the *link count* is *decremented* but still we can read destination file (p2.txt).



2. Symbolic Links

- When the *link (ln) command* is executed *with no option*, the result is a *hard link*. If we try to create a *hard link to a different* file system, it is *rejected* because hard links *must bemade within the current directorystructure*.
- To link to a *different file system*, therefore, we must *use symbolic links*. We must also use symbolic links when we are linking to directories.
- There is a danger with symbolic links because, although they behave like files and directories, they *do not physically exist*. They only point to real directory or file.
- If the *physical file is deleted*, the file will *no longer appear on a listing* under its original name. It will still be *available under its symbolic link* name, but it is *not accessible.*
- If a physical directory is deleted, the symbolic link to the directory still exists. In that case if we try to list the symbolic directory, it lists with no files. However, if we try to move to it, we receive a message that it doesn’t exist.





DESCRIPTION	HARD LINK	SOFT LINK

SYNTAX	**ln**source target	**ln**-ssource target
INODE	Same for both source and target files.	Different for source and target file.
LINK COUNT	Increased	Single link count
FILE ACCESSIBILITY	If we delete source file, we can able to access target file.	If we delete source file ,target file is broken and we can't access

Remove (rm) command

- The **remove (rm)** utility *deletes* an entry from a directory and file. To *delete a file,* we must have write permission.

```
[student@krosum Demo]$ ls
process.log
[student@krosum Demo]$ rm process.log
[student@krosum Demo]$ ls
[student@krosum Demo]$
```

- In the above snap, process.log file is deleted from current directory.

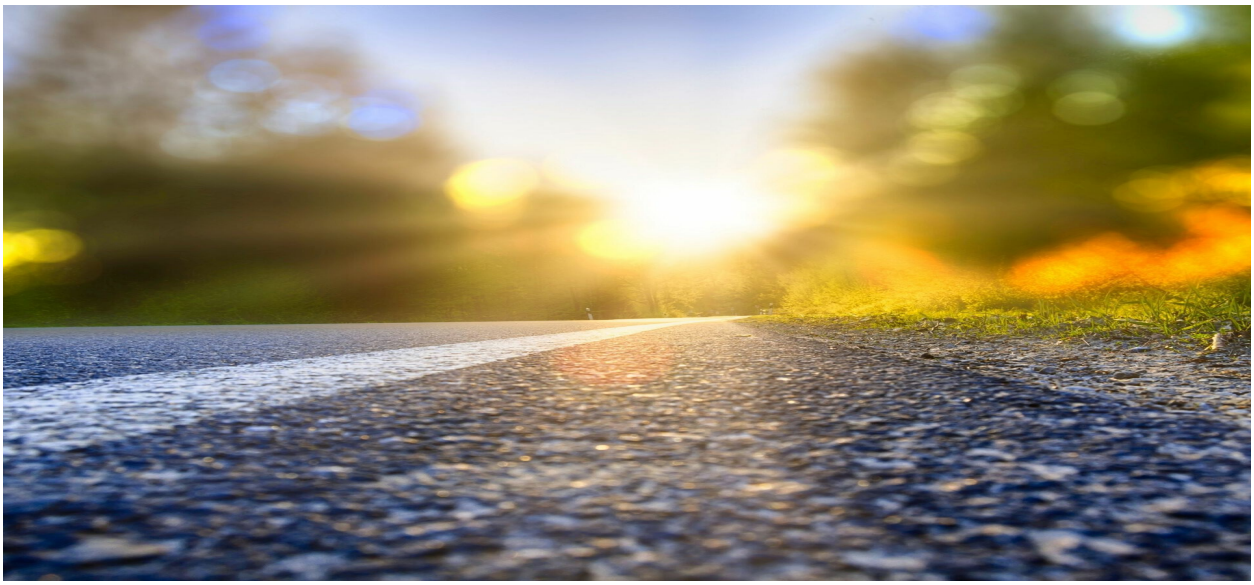
```
[student@krosum Demo]$ ls
p1.txt
[student@krosum Demo]$ rm -i p1.txt
rm: remove regular file ‘p1.txt’? n
[student@krosum Demo]$ ls
p1.txt
[student@krosum Demo]$
```

Note: to delete a directory, we used option **–r (recursive removal)**

–i (interactive) option

```
[student@krosum Demo]$ ls
p1.txt
[student@krosum Demo]$ rm -f p1.txt
[student@krosum Demo]$ ls
[student@krosum Demo]$
```

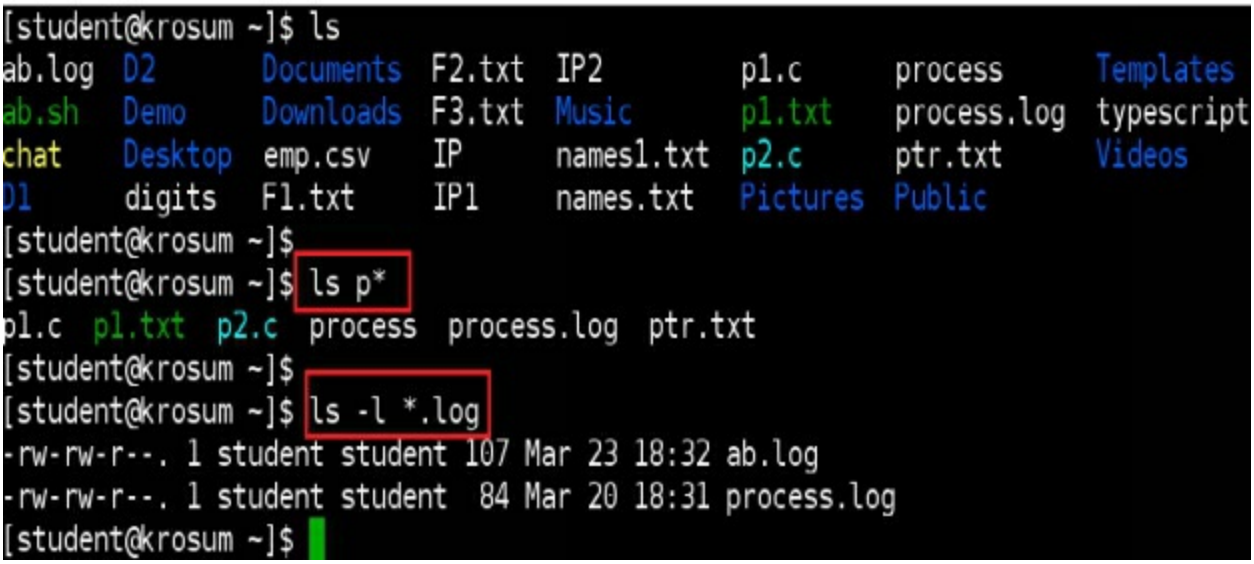
-f (force) option



CHAPTER 10 - SHELL META-CHARACTER

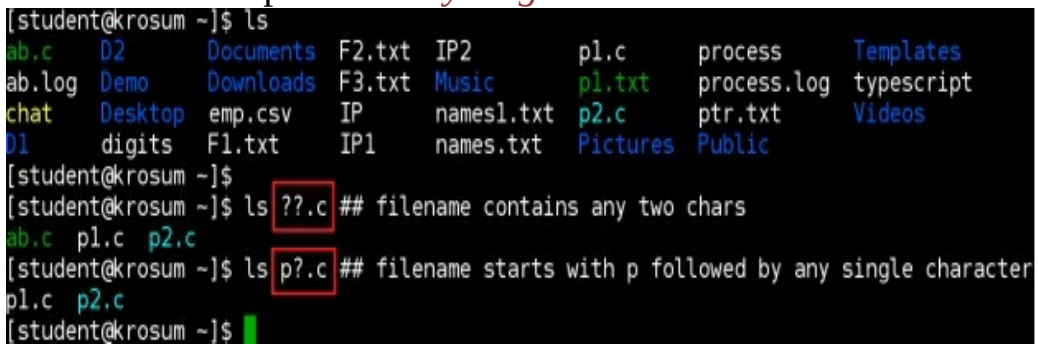
Asterisk (*) Character

- The *asterisk (*) character*is also called the *wildcard character* and *represents zero or more characters,* except the leading period *(.) of a hidden file.*
- For example, in order to list all files and directories that start with the letter ***p*** *followed by zero or more* characters and to list all files and directories that *end with the* ***.log****, preceded by zero or more characters,* refer following snap.

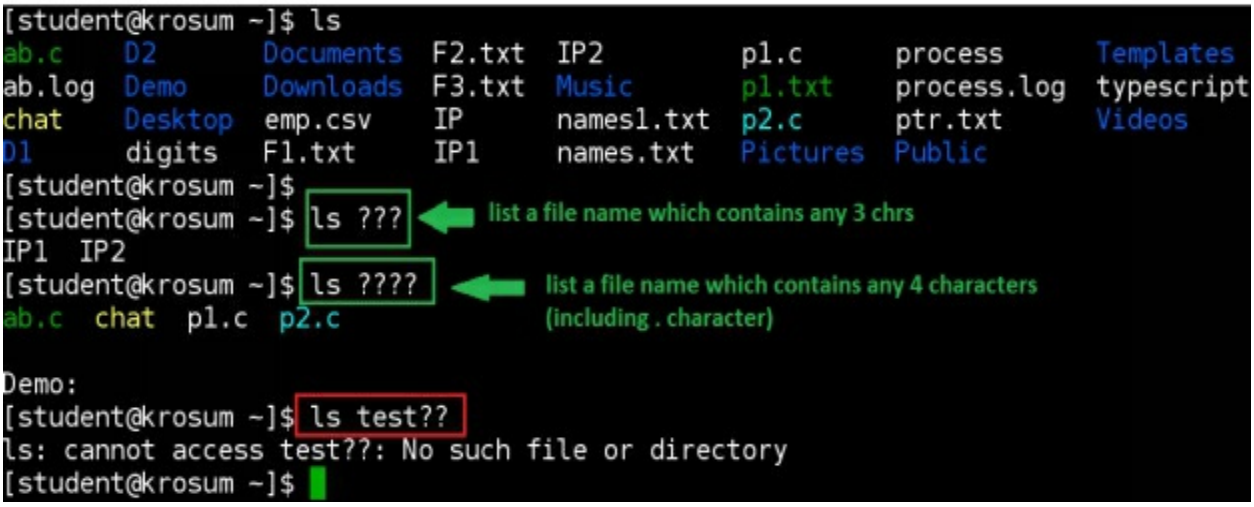


Question Mark (?) Character

- The *question mark (?)*character is also called a wildcard character that represents *any single character*.



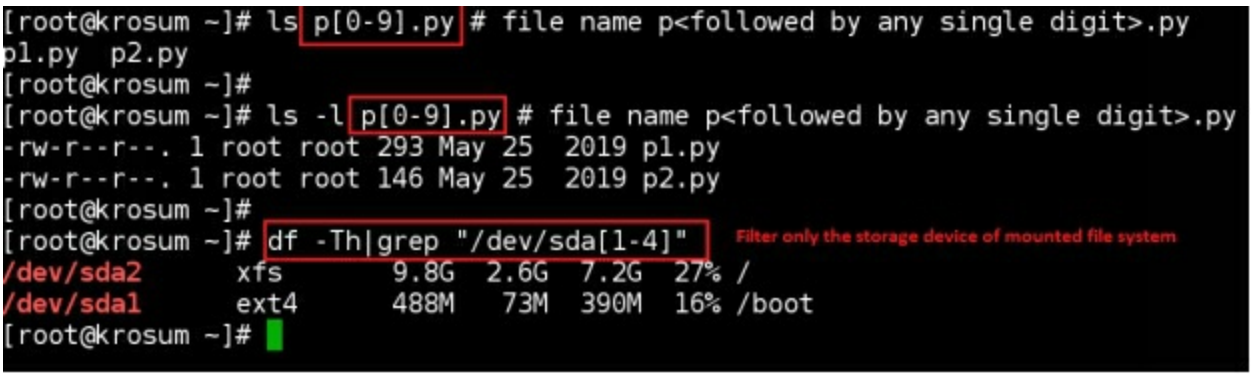
- For example, to list all files and directories where each file name contains exactly two characters that end with extension .c .



Note: If *no file matches* with an entry using the question mark (?) character, an *error message* appears.

Square Bracket ([]) Characters

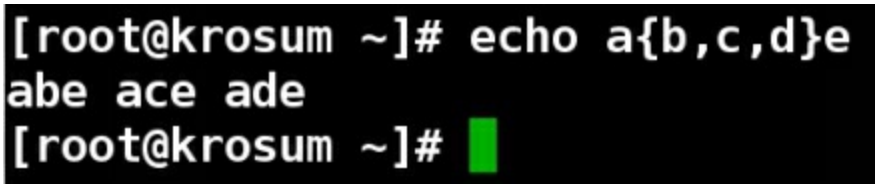
- The square bracket (**[ ]**) characters represent a set or range of characters for a single character position.
- A set of characters is any number of specific characters,
- For example, [acb].
    - The characters in a set do not necessarily have to be in any order.
    - For example, [abc] is the same as [cab].Whereas a range of characters is a series of ordered characters.
    - A range lists the first character followed by a hyphen (-) and then the last character,
    - for example, [a–z] or [0–9].
- When specifying a range, arrange the characters in the order that you want them to appear in the output.
    - For example, use [A–Z] or [a–z] to search for any uppercase or lowercase alphabetical character, respectively.



    - ls p[abc].java # file name p<followed by character a or b or c> .java extension

The Brace Expansion

- The brace {} expansion is a mechanism by which *arbitrary strings may be generated.*
- The preamble "a" is prefixed to each string contained within the braces, and the postscript "e" is then appended to each resulting string, expanding left to right.



- Shell meta-characters are specific characters, generally symbols that have special meaning within the shell.
- The meta-characters supported in bash are listed as follows:

|
&
;
(
)
<
>
Space
tab

The shell meta-characters are listed as follows:

META-CHARACTERS	DESCRIPTION
|	Sends the output of the command placed at the left of the pipe symbol as an input to the command on the right of the symbol.
&	Runs the process in the background, allowing you to continue working on the command line.
;	Allows you to list multiple commands on a single line, separated by this character.
()	Groups commands and sends their output to the same place.
<	The command placed at the left of the symbol, gets its input from the right of the symbol
>	Sends the output of the command placed on the left of the symbol, into the file named on the right of the symbol.

Caution: Do not include these meta-characters in filename/ directory name during their creation.

Redirecting Meta-characters

- Command redirection is enabled by the following shell meta-characters:
    - Redirection of standard input (<)
    - Redirection of standard output (>)
    - Redirection of standard error (2>)

The File Descriptors

- Each process works with file descriptors.
- File descriptors determine where the input to the command originates and where the output and error messages are directed to.
- The table explains the file descriptors.

FILE DESCRIPTOR NUMBER	FILE DESCRIPTION ABBREVIATION	DEFINITION
0	STDIN	Standard command input
1	STDOUT	Standard command output
2	STDERR	Standard command error

Command Redirection

- By default, the shell receives or reads input from the standard input, the keyboard and displays the output and error messages to the standard output, the screen.
- *Input redirection* forces a command to read the input from a file instead of receiving from the keyboard.

```
[root@krosum T]# uname -rs # display kernel name and version
Linux 3.9.5-301.fc19.i686
[root@krosum T]#
[root@krosum T]# uname -rs >r1.log # redirecting command result to file(r1.log)
[root@krosum T]# ls -t
r1.log  emp.csv
[root@krosum T]# cat r1.log
Linux 3.9.5-301.fc19.i686
[root@krosum T]# ps >process.log # redirect process result to process.log file
[root@krosum T]# ls
emp.csv  process.log  r1.log
[root@krosum T]#
[root@krosum T]# ps -e|grep bash|wc -l
[root@krosum T]# ps -e|grep bash|wc -l >count.log # redirect pipe result to count.log file
[root@krosum T]# ls
count.log  emp.csv  process.log  r1.log
```

- *Output redirection* sends the output from a command into a file instead of sending the output to the screen.

Redirecting Standard Input

- The less than (<) meta-character processes a file as the standard input instead of reading the input from the keyboard.

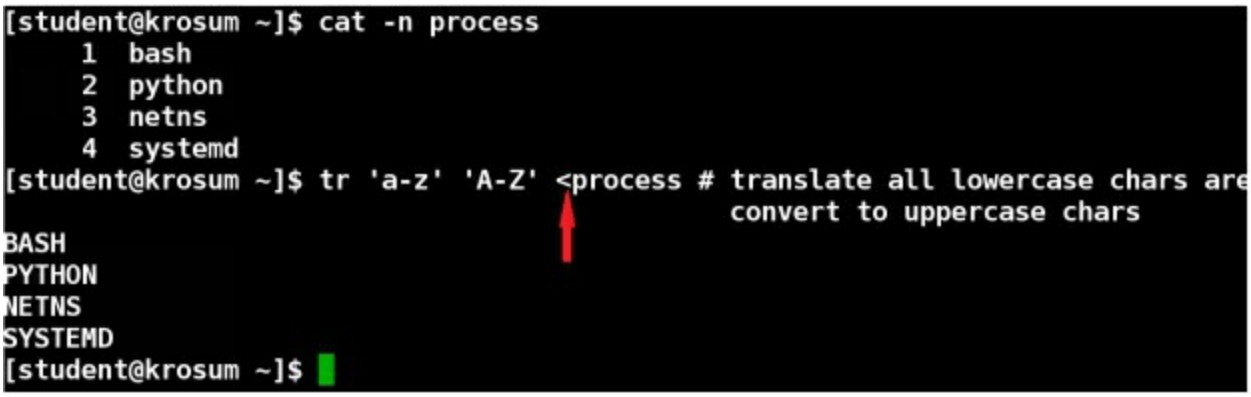
command<filename

or

command0<filename

```
[root@krosum ~]# mailx <result.log
```

For example, use the **result.log**file as the input for the **mailx** command.



Redirecting Standard Output

- The **greater-than (>)**meta-character redirects the standard output to a file instead of printing them to the screen.

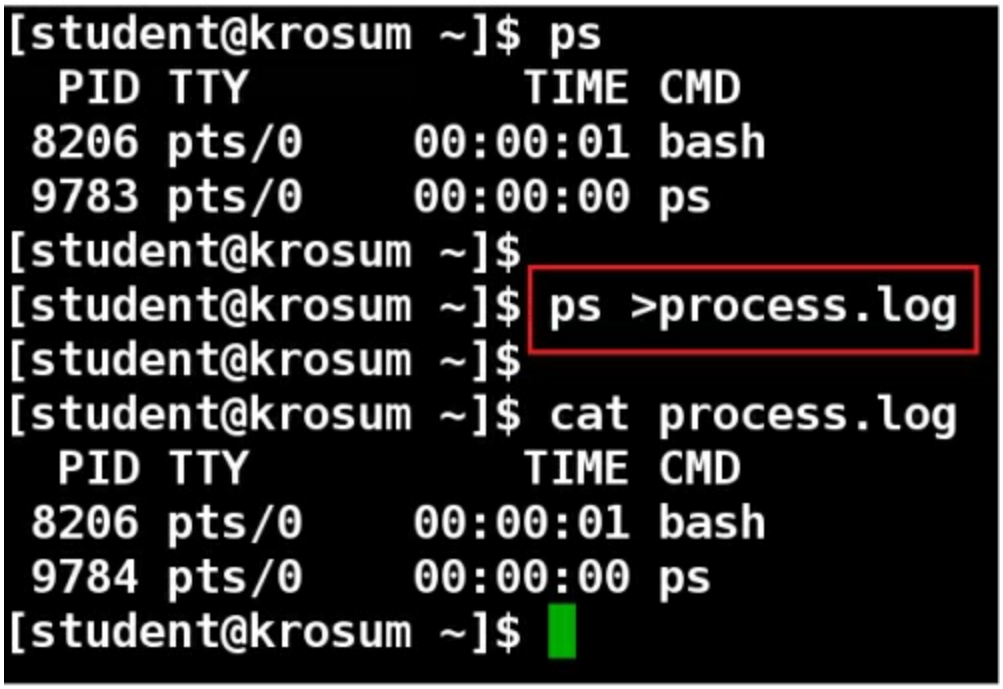
command>filename

or

command1> filename

- If the *file does not exist*, the system *creates* it. If the *file exists*, the *redirection overwrites* the content of the file.
- For example, writes the output of current process command to **process.log**file

ps>process.log



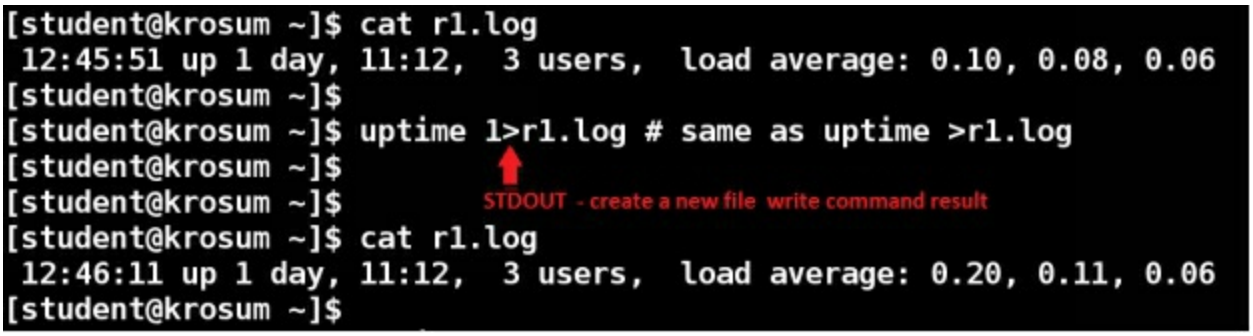
- When you use *a single greater-than* (>) meta-character and if the *file already exists*, the command *overwrites the original* contents of the file whereas when you use *double greater-than (>>)* characters, the command *appends* the output to the original

content of the file.

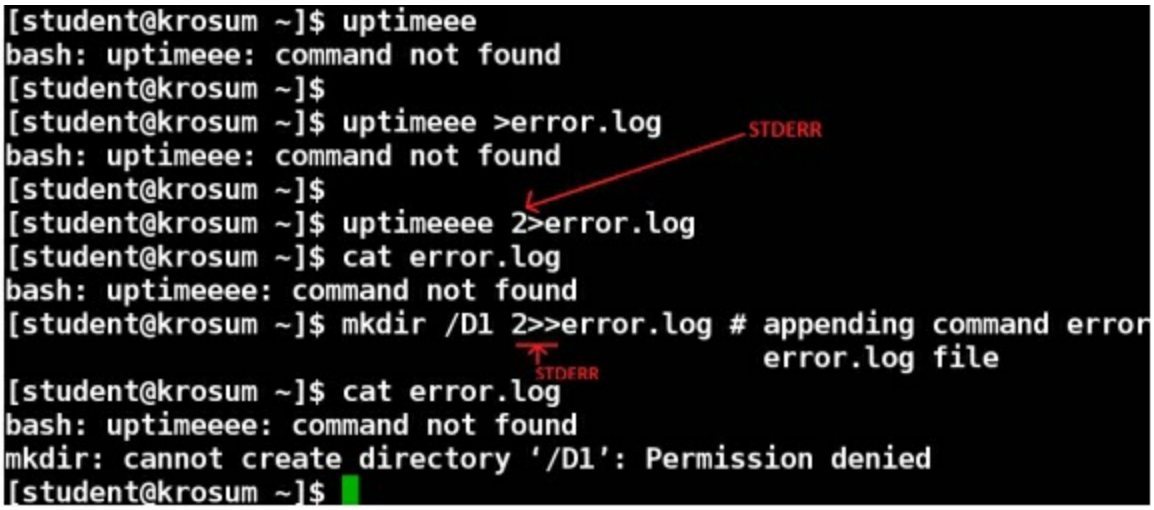
Redirecting Standard Error

- A command using the **file descriptor number (2)**and the greater-than (>) sign *redirects* any standard error messages to the **/dev/null**file.

command2>/dev/null

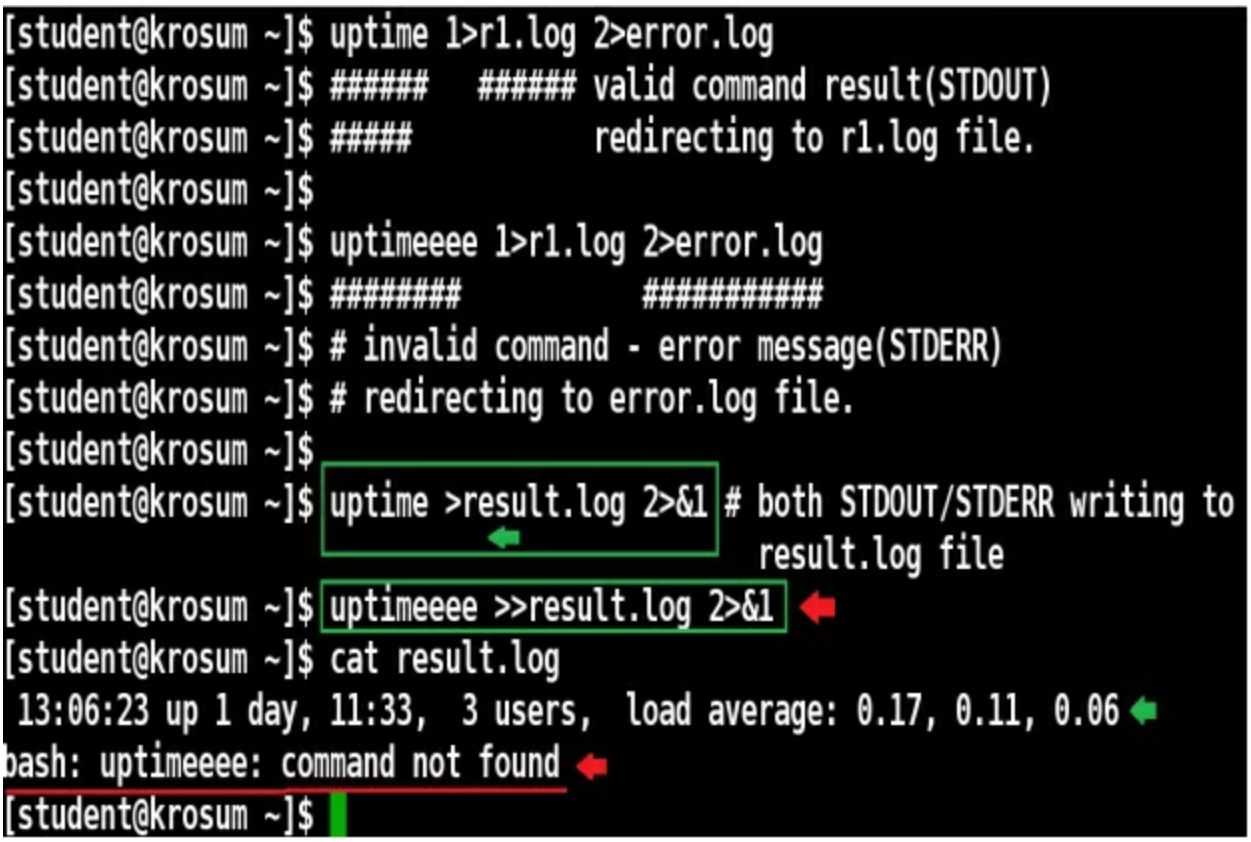


The following example shows the standard error redirected to the **error.log**file.



For redirecting both stdout and stderr message to the same file, use following syntax

2>&1



Note: The syntax 2>&1 instructs the shell to redirect stderr (2) to the same file that receives stdout (1).

Using the Pipe Character

- The pipe character *redirects* the standard output of one command to the standard input of another command.

Command1 | Command2

- For example, use the standard output from the who command as the standard input for the wc-l command

who| wc-l

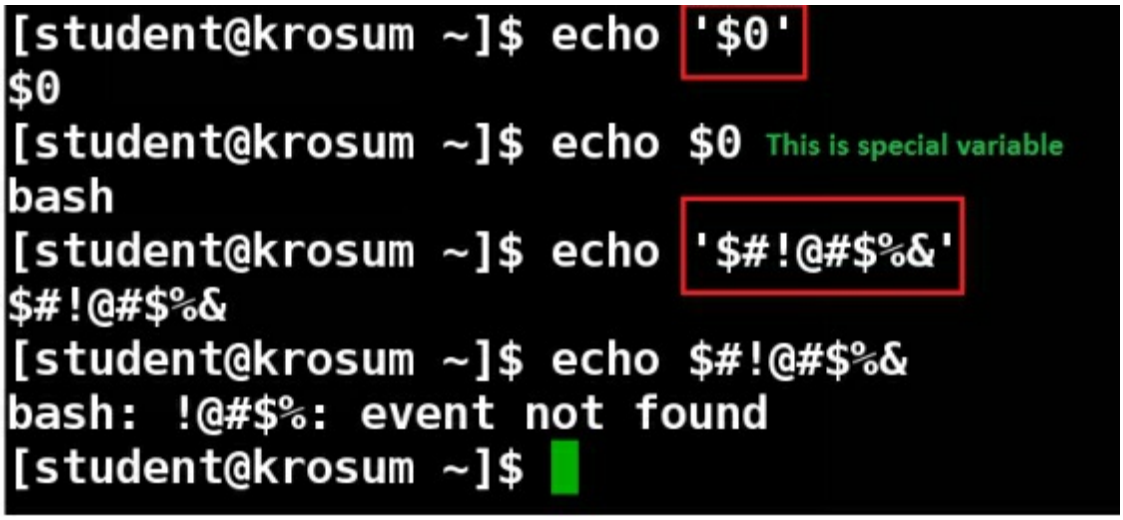
cat/etc/passwd|grepbash|wc–l

```
[student@krosum ~]$
[student@krosum ~]$ who|wc -l
3
[student@krosum ~]$ cat /etc/passwd|grep bash
root:x:0:0:root:/root:/bin/bash
student:x:1000:1000:apelix student:/home/student:/bin/bash
[student@krosum ~]$
[student@krosum ~]$ cat /etc/passwd|grep bash|wc -l
2
[student@krosum ~]$
```

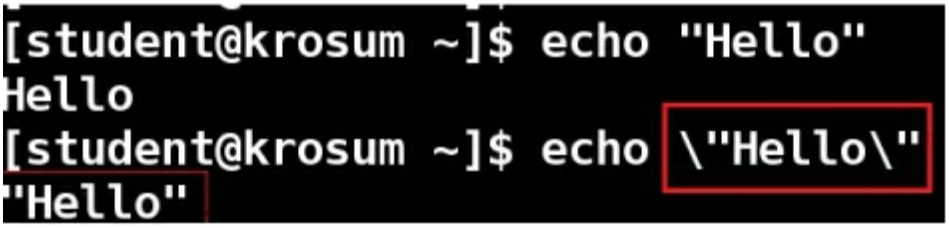
Like this we can use pipes to connect numerous commands.

Quoting Characters

- Quoting is a process that instructs the shell to mask or ignore the special meaning of shell meta characters.
- The quoting characters are:
    - Single forward quotation marks (‘ ‘): Instruct the shell to ignore all enclosed meta-characters.

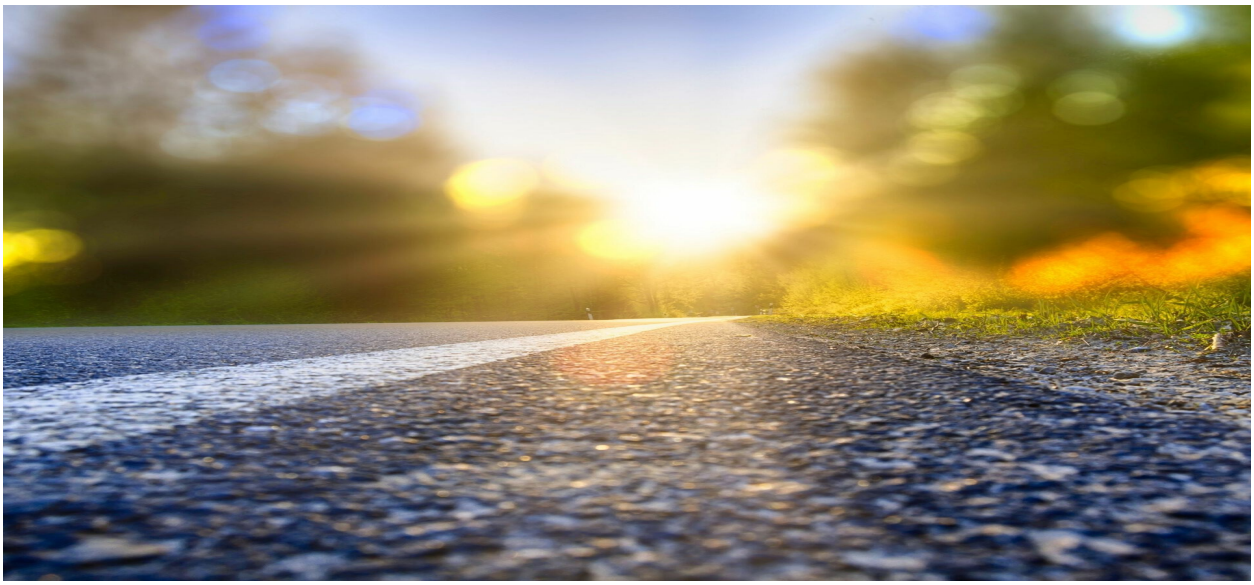


    - Backslash (\): Prevents the shell from interpreting the next character after the (\) as a meta-character.



- Single backward quotation marks ('): Instruct the shell to execute and display the output for a UNIX system command.

- Parentheses ($ (command)): Instruct the shell to execute and display the output of the command enclosed within parentheses.



CHAPTER 11 - FILE PERMISSION

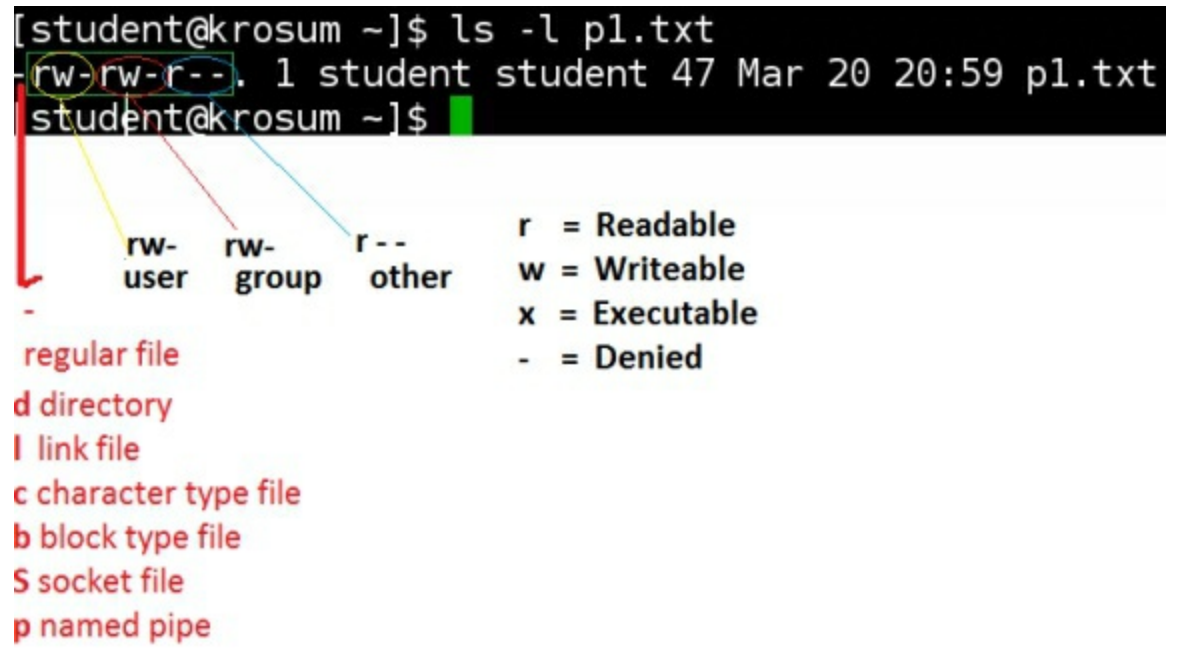
- All *files and directories* in Linux have a standard set of *access permission.*
- This access permission *controls the access to all the files* available and can provide a fundamental level of *security to the files and directories* in a system.

How to view file permission?

- To view the permissions for files and directories, use

ls–*l*
or
ls–*n*

```
[student@krosum ~]$ ls -l p1.txt
-rw-rw-r--. 1 student student 47 Mar 20 20:59 p1.txt
[student@krosum ~]$
```



- The *first field*of information displayed by the **ls-l** command is the **file type**.
    - *The file type typically specifies whether it is a file or a directory*.
    - A *file* is represented by a *hyphen (-)* whereas a *directory* is represented by the *letter* ***d***.
- The *remaining fields*represent the permission groups: ***owner, group,***and ***other.***

Permission Groups

There are three categories of permissions groups:

– Owner

– Group

– Other

• The table describes the permission groups and their scope

PERMISSION	DESCRIPTION
Owner	Permissions used by the assigned owner of the file or directory.
Group	Permissions used by members of the group that owns the file or directory.
Other	Permissions used by all other users except the file owner, and members of the group that owns the file or the directory.

Permission Set

- Each permission group acquires three permissions, ***read, write*** *and* ***execute*** which together called a ***permission set***.
- Each file or directory has a permission set specified for each permission group. Since we have 3 type of permission group, totally *each file or directory* contains *3 permission set* corresponding to *each group.*
- The *first permission* set represents the ***owner permissions***the *second set* represents the ***group permissions***, and the *last set* represents the ***other permissions***.
- The ***read, write***and ***execute***permissions are represented by the characters ***r, w,***and***x***respectively.
- The presence of any of these characters (***r***or***w*** or ***x),***indicates that the particular permission is granted.
- A *dash (-) symbol* in place of a character in a permission set indicates that a particular ***permission is denied.***
- Linux assigns initial permissions automatically when a new file or directory is created.

Interpreting File and Directory Permissions

PERMISSION	FILE ACCESS	DIRECTORY ACCESS
Read (*r*)	◦ We can display file contents and copy the file.	◦ We can list the directory contents with the ***ls*** command.
Write(*w*)	◦ We can modify the file contents.	◦ We can modify the contents of a directory, such as by creating and deleting the file.
Execute(*x*)	◦ If our file has executepermission, we can execute it both on command line and within a shell script.	◦ We can use the cd command to access the directory. ◦ By default, we have the read access and can run the ***ls*** **–** ***l*** command on the directory to list the contents. ◦ Even in case if we are restricted for a read access, we can run the ***ls*** command as long as we know the file name.

The read, write, and execute permissions are interpreted differently depending upon whether it is a file or a directory. **Note:** For a *directory to be ofgeneral use*, it must at least have *read* and *execute* permissions.

Determining File or Directory Access

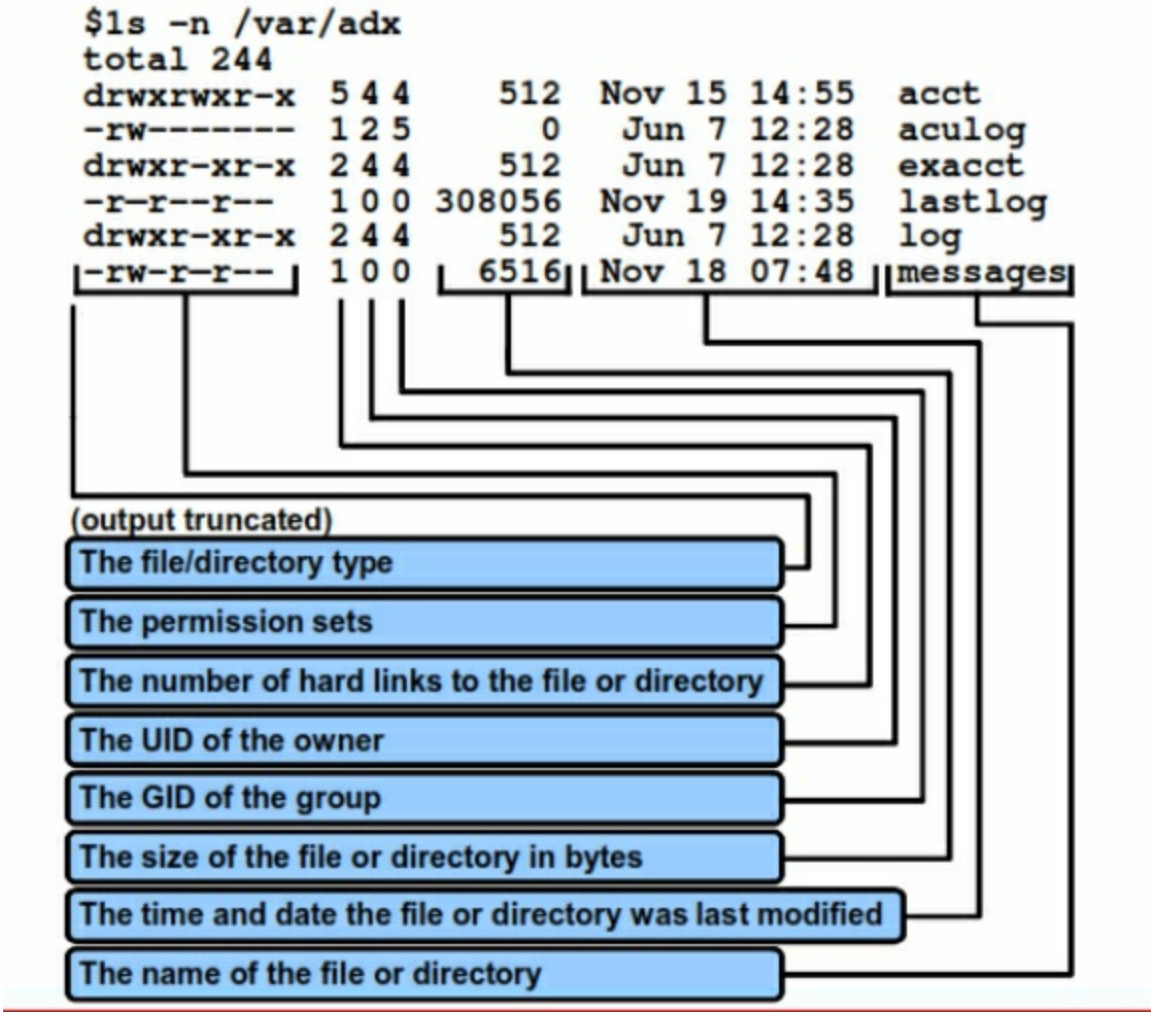
The ***ls -n*** command determines the ownership of files and directories.

- All files and directories have an associated ***useridentification number (UID)***and ***a group identification number (GID).***
- To view the UIDs and GIDs, run the***ls-n***command on the ***/var/adm***directory.

***ls-n*p1.txt**

- The**UID**identifies the *user* who owns the file or directory.
- The **GID** identifies the *group of users* who owns the file or directory.
- The Linux uses these numbers to track the ownership and group membership of files and directories.

Interpreting the ls – n Command

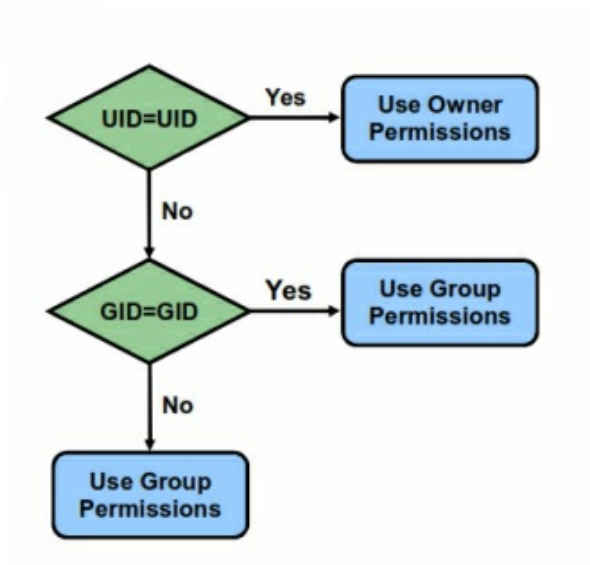


- The above image illustrates the parts of the output of the

ls-**n** *command*

- The *first character* is the **file/directory type.**
- The *second* set of *nine characters* represents the ***permission set.***
  - The *third character* represents the**number of hard links**to the file or directory.
    - A *hard link* is a *pointer* that shows the *number of files or directories* that are *linked* with a particular file in the *same file system.*

- The *fourth character* represents the **UID of the owner.**
- The *fifth character* represents the ***GID of the group*.**
- The *sixth set* represents the ***size*** *of the file or directory in bytes.*
- The *seventh set of characters* represents *the* **time** *and* **date** *of the file or directory that was last modified.*
- The *last set of characters* represents *the* **name** *of the file or directory.*



Determining Permissions

- When a user attempts to access a file or directory, the *UID of the user* is *compared with theUID of the file or directory*.
- If the UIDs match, the *permission set* for the owner determines whether the owner has access to the file or directory.
- If the UIDs do not match, the *user's GID is compared with the GID* of the file or directory. If these numbers matches, the group permissions apply.
- If the GIDs do not match, *the permission set for other* is used to determine file and directory access.

- Referring to the above image,
    - If the UID equals the UID, then use the owner permissions.
    - If not, checks whether GID equal the GID? If yes, use group permissions. If not, use other permissions.

Changing the Permissions

- We can change the permissions on files and directories by using the **chmod** command.
- Either the owner of the file/directory (or) the root user can use the ***chmod*** command to change the permissions.
- The ***chmod*** command can be used in either *symbolic* or octal *mode*.
    - *Symbolic mode* uses a combination of letters and symbols to add or remove permissions for each permission group.

    - *Octal mode*, also called the absolute mode, uses octal numbers to represent each permission group.

Note :

We can assign execute permissions on files with the ***chmod***command. The ***chmod*** command is described later in this lesson. *Execute permissions are not assigned by default* when we create a file.

Changing Permissions: Symbolic Mode

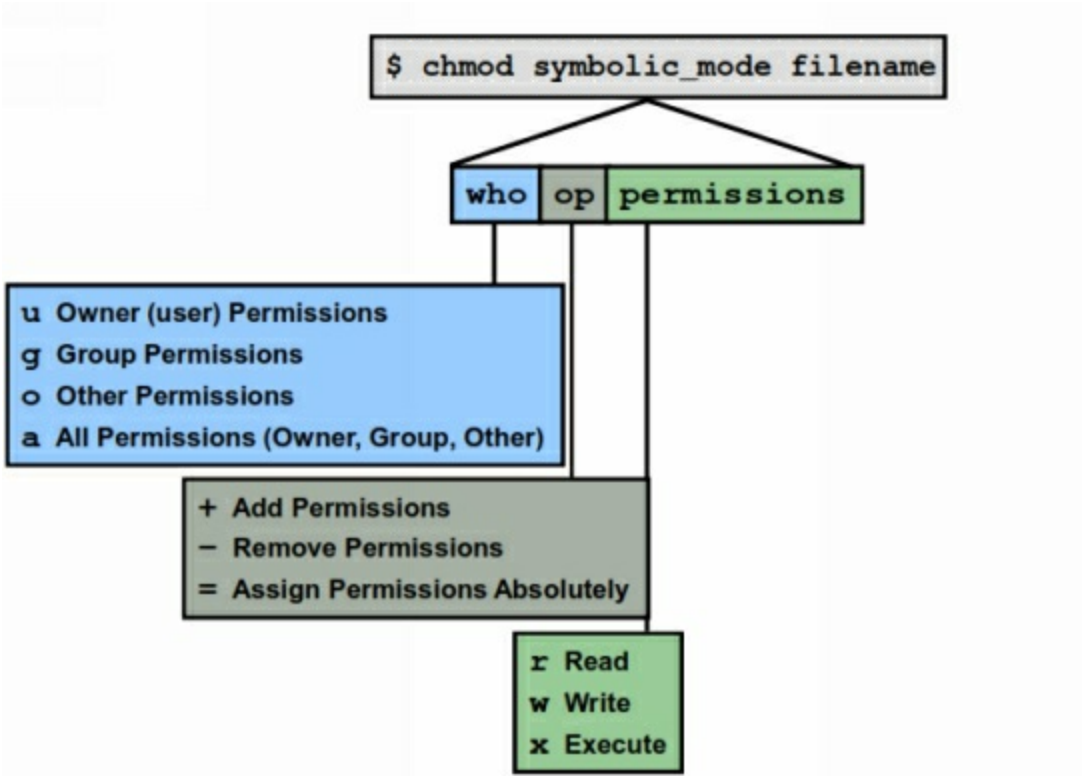
- The syntax for the chmod command in the symbolic mode is:

chmodsymbolic_modefilename

- The *symbolic_mode* option consists of three parts:
    - The *user category* (owner, group, or other) affected
    - The *function performed*
    - The *permissions affected*
- For example, if the option is ***g+x,*** the executable permission is added to the group.

```
student@krosum ~]$ ls -l p1.txt
-rw-rw-r--. 1 student student 47 Mar 20 20:59 p1.txt
student@krosum ~]$ chmod u+x p1.txt  # Add execute permission for the user
student@krosum ~]$
student@krosum ~]$ ls -l p1.txt
-rwxrw-r--. 1 student student 47 Mar 20 20:59 p1.txt
student@krosum ~]$
student@krosum ~]$ chmod o-r p1.txt # To remove the read permission for others
student@krosum ~]$
student@krosum ~]$ ls -l p1.txt
-rwxrw----. 1 student student 47 Mar 20 20:59 p1.txt
student@krosum ~]$
```

- The following examples illustrate how to modify the permissions on files and directories by using the symbolic mode.



Changing Permissions: Symbolic Mode

- The above image shows the components of the symbolic mode command syntax.
- The first three letters represent ***"who"*** and consist of the following codes:

  u : Owner (user) permission

  g : Group permissions

  o : Other permissions

  a : All permissions (owner, group, other)

- The next section is the ***"op"*** section and consists of the following:

  \+ : Add permissions

  \- : Remove permissions

  = : Assign permissions

- The last section is the ***"permissions"*** section and consists of the

following:

r : Read

w : Write

x : Execute

Changing Permissions: Octal Mode

- The***chmod*** command syntax in the *octal mode* is:

chmodoctal_modefilename

- The *octal_mode* option consists of three octal numbers, 4, 2, and 1, which represent a *combination of permissions*, from 0–7, for the file/directory.

OCTAL VALUE	PERMISSION
4	Read
2	Write
1	Execute

- The above image shows the octal number corresponding to permission.
- For each permission set, these numbers are combined to form a single number.

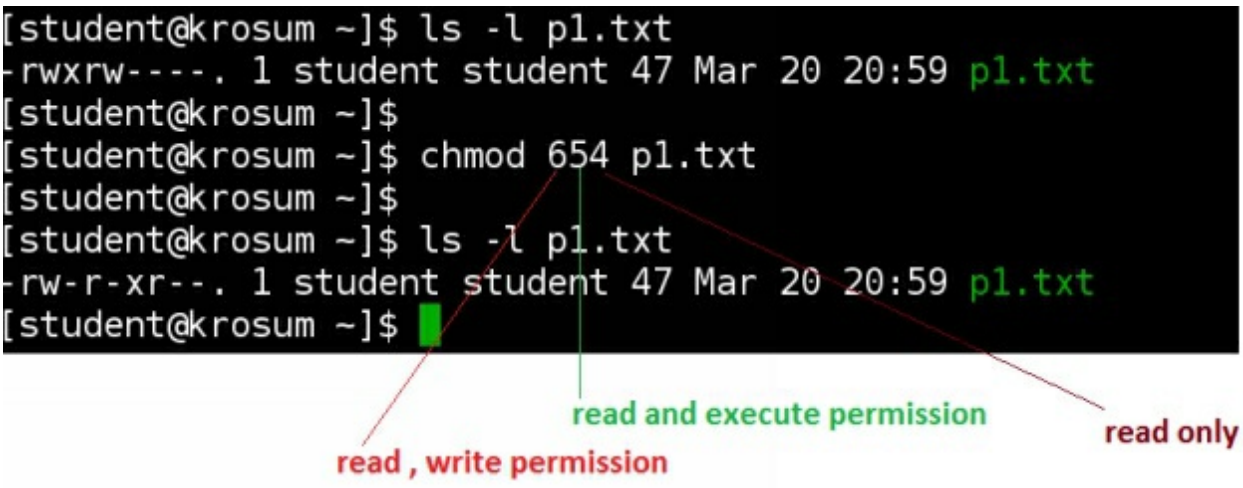
OCTAL VALUE	PERMISSION	BINARY
7	rwx	111 (4+2+1)
6	rw-	110(4+2+0)
5	r-x	101(4+0+1)
4	r--	100(4+0+0)
3	-wx	011(0+2+1)
2	-w-	010(0+2+0)
1	--x	001(0+0+1)
0	---	000(0+0+0)

Changing Permissions: Octal Mode

- The table shows the octal numbers that represent a combined set of permissions.
- We can modify the permissions for each category of the users by combining the octal numbers.
- The first set of octal number defines ***owner permissions***, the second set defines ***group permissions,*** and the third set defines ***other permissions***.

OCTAL MODE	PERMISSIC
644	rw-r--r--
751	rwxr-x—x
775	rwxrwxr-x
777	rwxrwxrwx

- The table shows the permission sets in the octal mode.***For example,*** let's set permissions so that the owner, group, and other have read and execute access only.



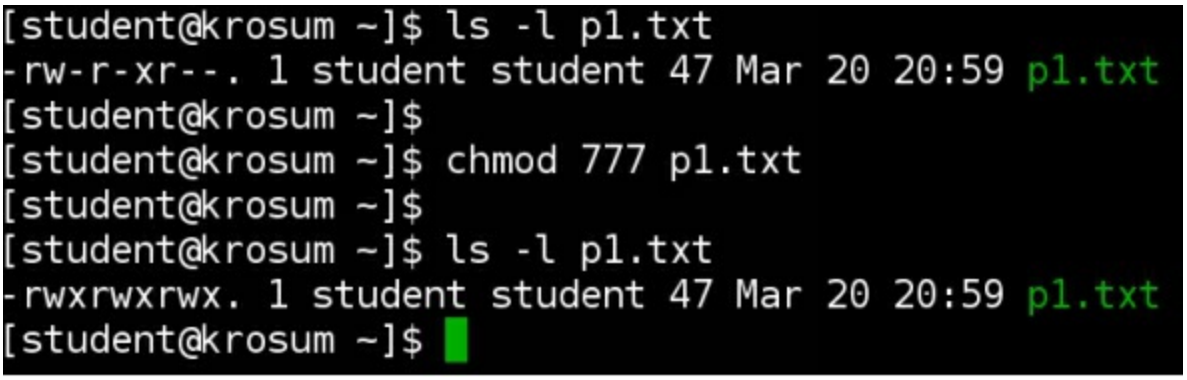
- The **chmod** command fills with zeros to the left of octal digits.

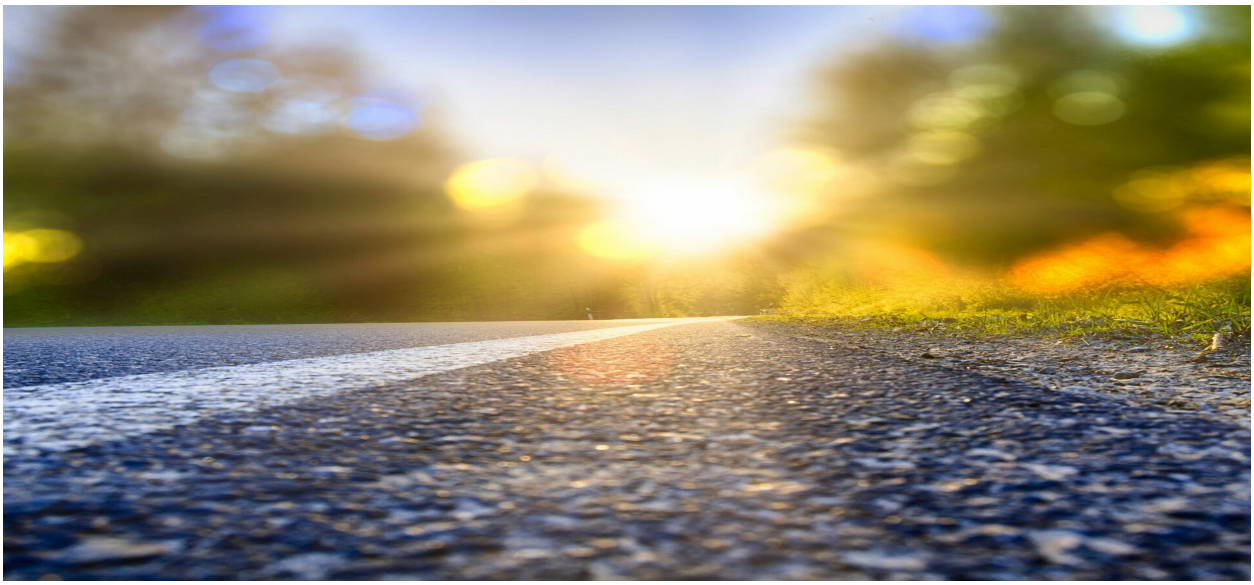
Note: **chmod44p1.txt**becomes **chmod044p1.txt**.

Caution

Note:- Missing one or more octal digits can lead to unwanted access to files or directories.

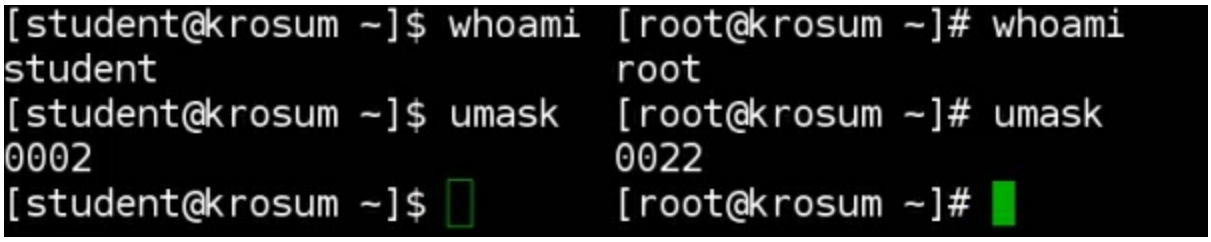
- Some additional examples show how to modify permissions on files and directories by using the octal mode.
    - Assigning all the permissions for owner, group and others, permits read, write and execute permissions fully to that particular file.


[student@krosum ~]$ ls -l p1.txt
-rw-r-xr--. 1 student student 47 Mar 20 20:59 p1.txt
[student@krosum ~]$
[student@krosum ~]$ chmod 777 p1.txt
[student@krosum ~]$
[student@krosum ~]$ ls -l p1.txt
-rwxrwxrwx. 1 student student 47 Mar 20 20:59 p1.txt
[student@krosum ~]$



CHAPTER 12 - THE UMASK COMMAND

- When files and directories are created, initial permission values are automatically assigned.
- The initial permission value for a file is ***666 (rw-rw-rw-)*** and it is ***777 (rwxrwxrwx)*** for a directory.
- The user mask affects and modifies the default file permissions assigned to the file or directory.
- We can set the user mask by using the **umask**command in a user initialization file.
- To view the umask value, run the**umask** command.



Note:

- The *default umask* value for anon-root user is ***002.***
- The *default umask* value for a root user is ***022.***

- The ***umask*** utility affects the *initial permissions* for files and directories when the files and directories are created.
- The ***umask*** utility is a *three-digit octal value* that is associated with the *read*, *write* and *execute* permissions.
- The *first digit* determines the *default* permissions *for theowner*, the *second digit* determines the *default* permissions *for the group* and the *third digit* determines the *default* permissions *for other*.
- For example, *to set the default file permissions* in a user initialization file to **rw-rw-rw-,**run the following command:

umask000

Determining umaskValue

umask OCTAL VALUE	FILE PERMISSIONS	DIRECTORY PERMISSIONS
0	**rw-**	**rwx**
1	**rw-**	**rw-**
2	**r--**	**r-x**
3	**r--**	**r--**
4	**-w-**	**-wx**
5	**-w-**	**-w-**
6	**---**	**--x**

- The table shows the file and directory permissions for each of the *umask octal value*.
- This table can also help us to determine the *umask value* that we want to set on files and directories.
- To determine the umask value for a file, subtract the value of the permissions that we want from 666 or 777 from a directory.
- Let's see an example. If we want to change the default mode for files to **644 (rw-r--r--),**the difference between **666** and **644** gives **022**, which is the value you would use as an argument to the **umask**command.

6 6 6
0 0 2 (-)
6 6 4 **Default file permission** **(non-root user)**

rw-rw-r--

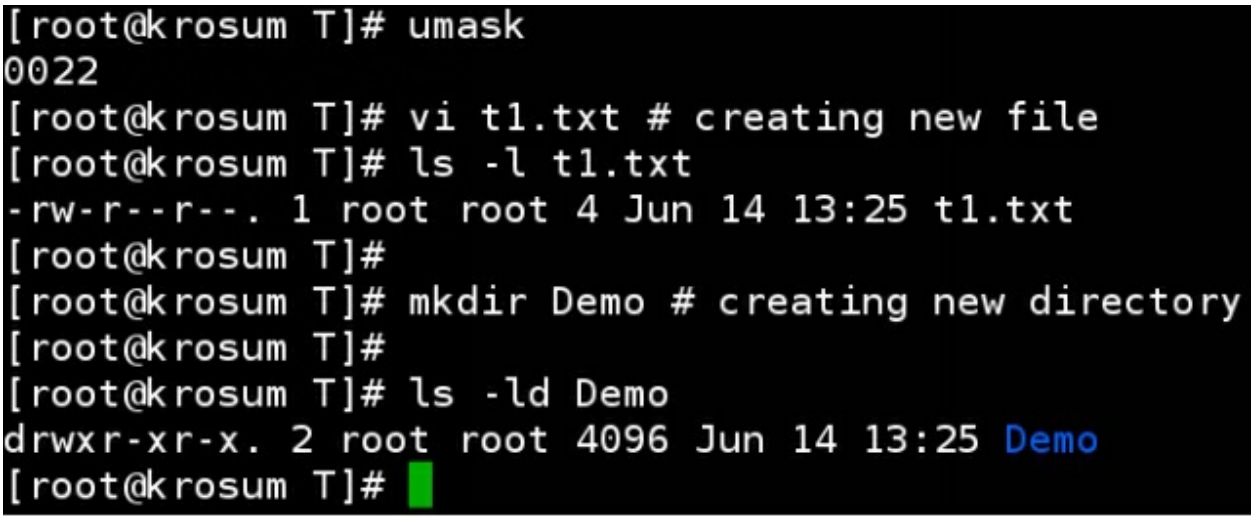
7 7 7
0 0 2 (-)
7 7 5
Default directory permission
(non-root user)
rwxrwxr-x

```
[student@krosum ~]$ umask
0002
[student@krosum ~]$ vi t1.txt # creating new file
[student@krosum ~]$ ls -l t1.txt
-rw-rw-r--. 1 student student 3 Jun 14 13:20 t1.txt
[student@krosum ~]$
[student@krosum ~]$ mkdir Demo # creating new directory
[student@krosum ~]$
[student@krosum ~]$ ls -ld Demo
drwxrwxr-x. 2 student student 4096 Jun 14 13:21 Demo
[student@krosum ~]$
```

7 7 7
0 2 2 (-)
7 5 5
Default directory permission (root user)
rwxr-xr-x

6 6 6
0 2 2 (-)

6 4 4 **Default file permission** **(root user)** **rw-r--r--**


[root@krosum T]# umask
0022
[root@krosum T]# vi t1.txt # creating new file
[root@krosum T]# ls -l t1.txt
-rw-r--r--. 1 root root 4 Jun 14 13:25 t1.txt
[root@krosum T]#
[root@krosum T]# mkdir Demo # creating new directory
[root@krosum T]#
[root@krosum T]# ls -ld Demo
drwxr-xr-x. 2 root root 4096 Jun 14 13:25 Demo
[root@krosum T]#

Applying the umask Value

- The default permissions assigned to the new files and directories are based on the initial value of *umask*which varies with the user login (whether root or non-root).
- The table displays the results in the *symbolic mode*. For example, the initial permissions for a new file in the symbolic mode is**rw-rw-rw-.**
- This set of permissions corresponds to read/write access for the owner, group, and other.
- This value is represented in the *octal mode* as, ***420420420*** or ***666***.
    - To mask out the write permission for the group and other, use ***022***, the *default umask value.*
    - The result in the octal mode is **420400400**or **644**, and ***rw-r—r–***in the symbolic mode.
- We can apply this same process to determine the default permissions for directories.
- For directories, the initial value specified by the system is ***rwxrwxrwx***. This corresponds to read, write, and execute access for the owner, group, and other.
- This value is represented in the *octal mode* as **421421421** or **777.**

PERMISSION FIELD	DESCRIPTION
rw-rw-rw-	Initial Value specified by the system for a new file
----w—w-	Default **umask** utility value to be removed
rw-r- -r- -	Default **permissions** assigned to newly created files
rwxrwxrwx	Initial values specified by the system for a new directory
----w- -w-	Default **umask** utility value to be removed
rwxr-xr-x	Default **permissions** set for newly created

	directories

- Use the default umask value of ***022***to mask out the write permission for the group and other.
- The result in the octal mode is ***421401401 or 755,***and ***rwxr-xr-x***in the symbolic mode.

Changing the umaskValue

- We can change the umask value to a new value on the command line.
- For instance, we might require a more secure umask value of say 027, which assigns the following access permissions to newly created files and directories:
    - Files with read and write permissions for the owner, read permission for the group, and no permissions for other (**rw-r-----**).
    - Directories with read, write, and execute permissions for the owner, read and execute permissions for the group, and no permissions for other (**rwxr-x---**).

Note

- The new umask value affects only those files and directories that are created from this point onwards.
- However, if the user logs out of the system, the new value (027) is replaced by the old value (022) on subsequent logins because the umask value has been changed at run time using the command line.
- If we want to make a permanent change on umask value, change the umask parameter included in the configuration file.

```
[root@krosum T]# umask # current umask value
0022
[root@krosum T]# vi t2.txt # creating new reg.file
[root@krosum T]# ls -l t2.txt
-rw-r--r--. 1 root root 0 Jun 14 13:45 t2.txt
[root@krosum T]#
[root@krosum T]# mkdir D1 # creating new directory D1
[root@krosum T]#
[root@krosum T]# ls -ld D1
drwxr-xr-x. 2 root root 4096 Jun 14 13:45 D1
[root@krosum T]#
[root@krosum T]# umask 027 # changing umask value
[root@krosum T]#
[root@krosum T]# umask
0027
[root@krosum T]# vi t3.txt # creating new reg.file
[root@krosum T]# ls -l t3.txt # default file attributes
-rw-r-----. 1 root root 0 Jun 14 13:46 t3.txt
[root@krosum T]# mkdir D2 # creating new directory
[root@krosum T]# ls -ld D2 # default directory attributes
drwxr-x---. 2 root root 4096 Jun 14 13:47 D2
[root@krosum T]#
```

The tar Command

- The tar command stores, lists, or extracts files in an archive.
- The output of using a tar command is a tar file.
- In Linux, the default output location for a tar file is the stdout.

Syntax

tar ***-option archivefileinputfiles***

OPTIONS	DEFINITION
C	Creates a new tar file
T	Lists the table of contents of the tar file
X	Extracts files from the tar file
F	Specifies the archive file or tape device.
V	Executes in verbose mode, writes to the standard output
H	Follows symbolic links as standard files or directories
Z	Reads or writes archives through gzip
J	Compresses and extracts files and directories using bzip

Creating a tar Archive

- You can use the tar command to create an archive file containing multiple files or directories onto a disk or file.
- The following example shows you how to archive your home directory onto a disk.

tar-cvfbackup.tar inputfile1..inputfileN

```
[root@krosum ~]# tar -cvf Backup1.tar JavaBackup Test emp.csv process.log
JavaBackup/
JavaBackup/sun-javadb-demo-10.6.2-1.1.i386.rpm
JavaBackup/sun-javadb-javadoc-10.6.2-1.1.i386.rpm
JavaBackup/jdk-6u37-linux-i586-rpm.bin
JavaBackup/jre-6u38-linux-i586-rpm.bin
JavaBackup/sun-javadb-docs-10.6.2-1.1.i386.rpm
JavaBackup/jdk-6u37-linux-i586.rpm
JavaBackup/install.sfx.2426
JavaBackup/sun-javadb-common-10.6.2-1.1.i386.rpm
JavaBackup/sun-javadb-client-10.6.2-1.1.i386.rpm
JavaBackup/sun-javadb-core-10.6.2-1.1.i386.rpm
Test/
Test/process.log
Test/emp.csv.gz
Test/B1.tar
Test/LOGFILEs.tar
Test/D1/
```

- The following example shows you how to archive multiple files into an archive file called ***files.tar.***

Viewing a tar Archive

- We can view the names of all the files that have been written directly to a disk or file archive.
- To view the content of the Backup1.tar directory on the disk, enter the following command:

To view the content of the files.tar archive file, enter the following command:

tar-tvf backup.tar

```
[root@krosum Temp]# ls
Backup1.tar  emp.csv
[root@krosum Temp]# tar -tvf Backup1.tar  # view the content of files
drwxrwxr-x student/student    0 2019-08-18 00:15 JavaBackup/
-rw-r--r-- student/student 969861 2010-11-03 09:56 JavaBackup/sun-javadb-demo-10.6.2-1.1.i386.rpm
-rw-r--r-- student/student 201273 2010-11-03 09:56 JavaBackup/sun-javadb-javadoc-10.6.2-1.1.i386.
rpm
-rwxrwxr-x student/student 68604823 2019-08-18 00:02 JavaBackup/jdk-6u37-linux-i586-rpm.bin
-rwxrwxr-x student/student 12154000 2019-08-18 00:05 JavaBackup/jre-6u38-linux-i586-rpm.bin
-rw-r--r-- student/student  4865183 2010-11-03 09:56 JavaBackup/sun-javadb-docs-10.6.2-1.1.i386.r
pm
-rw-r--r-- student/student 58433453 2012-09-25 01:49 JavaBackup/jdk-6u37-linux-i586.rpm
-rw-r--r-- student/student 12149445 2019-08-18 00:11 JavaBackup/install.sfx.2426
-rw-r--r-- student/student    14627 2010-11-03 09:56 JavaBackup/sun-javadb-common-10.6.2-1.1.i386
.rpm
-rw-r--r-- student/student   499375 2010-11-03 09:56 JavaBackup/sun-javadb-client-10.6.2-1.1.i386
.rpm
```

Extracting a tar Archive

- We can retrieve or extract the contents of an archive that was written directly to a disk device or to a file.
- To retrieve the files from the disk archive, enter the following command:

tar-xvf backup.tar

```
[root@krosum Temp]# ls
Backup1.tar  emp.csv
[root@krosum Temp]# tar -xvf Backup1.tar
JavaBackup/
JavaBackup/sun-javadb-demo-10.6.2-1.1.i386.rpm
JavaBackup/sun-javadb-javadoc-10.6.2-1.1.i386.rpm
JavaBackup/jdk-6u37-linux-i586-rpm.bin
JavaBackup/jre-6u38-linux-i586-rpm.bin
JavaBackup/sun-javadb-docs-10.6.2-1.1.i386.rpm
JavaBackup/jdk-6u37-linux-i586.rpm
JavaBackup/install.sfx.2426
JavaBackup/sun-javadb-common-10.6.2-1.1.i386.rpm
JavaBackup/sun-javadb-client-10.6.2-1.1.i386.rpm
JavaBackup/sun-javadb-core-10.6.2-1.1.i386.rpm
Test/
Test/process.log
```

File Compression

- With the enormous amount of enterprise data that is created and stored, there is a urgent need to conserve disk space and optimize data transfer time.
- There are various tools, utilities, and commands that are used for file compression. Some of the commonly used commands are:
    - The **compress** command
    - The**gzip**command
    - The **zip** command

Viewing a Compressed File: zcat Command

- The **zcat** command prints the uncompressed form of a compressed file to the standard output.
- To view the content of the **filename.gz**compressed file, enter the following command:
- **zcatfilename.gz**

Note: The **zcat** command interprets the compressed data and displays the content of the file as if it has not been compressed.

Compressing a File : gzip Command

- Alternatively, you can use the **gzip**command to compress files.
- The**gzip**command performs the same function as the compress command, but the **gzip**command generally produces smaller files.
- For example, to compress a set of files, **file1**, **file2**, **file3** and **file4**, enter the following command:

```
[root@krosum Temp]# cat process.log
  PID TTY          TIME CMD
26027 pts/2    00:00:00 bash
26240 pts/2    00:00:00 ps
[root@krosum Temp]#
[root@krosum Temp]# gzip process.log
[root@krosum Temp]# ls
Backup1.tar  emp.csv  JavaBackup  process.log.gz  Test
[root@krosum Temp]# cat process.log.gz
w[illegible]process.logS[illegible]tQ    [illegible]D_Wg_[illegible].#3[illegible]#s[illegible]b}#[illegible] $%[illegible]g%[illegible]L
                                                                    [illegible]H[illegible]s[illegible]e=[illegible]<T[root@krosum Temp]#
[root@krosum Temp]#
```

Note: The compressed files have a **.gz**extension.

Uncompressing a File : gunzipCommand

- The gunzip command uncompresses a file that has been compressed with the gzip command.

gunzipfilename

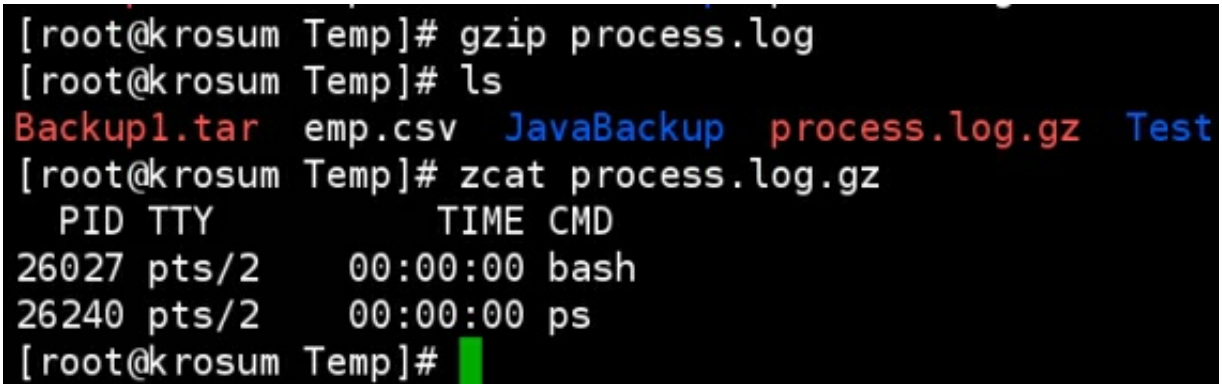
- To uncompressing or extract the **file1.gz** file, use the following command:

```
[root@krosum Temp]#
[root@krosum Temp]# ls
Backup1.tar  emp.csv  JavaBackup  process.log.gz  Test
[root@krosum Temp]#
[root@krosum Temp]# gunzip process.log.gz
[root@krosum Temp]# ls
Backup1.tar  emp.csv  JavaBackup  process.log  Test
[root@krosum Temp]#
```

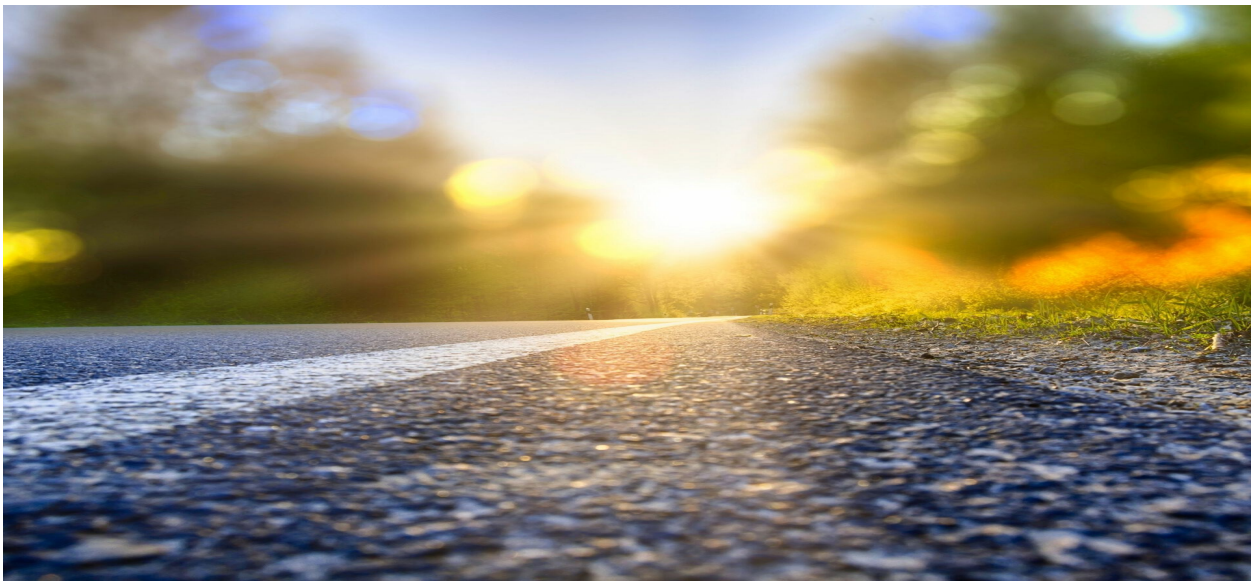
gunzipfile1.gz

Compressing and Archiving Multiple Files: zip Command

- The zip command compresses and archives multiple files into a single file in one go.
- To compress file2 and file3 into the file.zip archive file, enter the following command:



ziptarget_filenamesource_filenames



CHAPTER 13 - LINUX PROCESS

As we discussed, Linux abstractions are *file* and *process*. We discussed what is file? What is Linux file structure? What are file types? and what are file manipulation commands?

Now let's discuss about process.

A process,also known as a task, is *the running form of a program*. It means when the file enters *the execution state*, the execution instance of a file (or) program (or) command called *process* is created.

In simpler form, *the execution of a Linux command is called process.*

Whenever a new process is created, kernel will create a unique ID and this ID is called process ID (**PID**).

- Processes have a parent/child relationship.
- A process can spawn one or more children.

- Multiple processes can run in parallel.
- Linux command line is an active process (Running process).

Whenever a new command is executed on the command line, execution of new command is called that child process is created.

By the time, the shell command line process will enter in to waiting state and once child execution is done. i.e., reached to exit state, the parent process (shell command line) will resume to the active (Running)state.

Attributes of a Process

- The kernel assigns a unique identification number to each process called a process ID or **PID**.
- The Kernel uses the PID to track, control and manage the process.
    - Each process is further associated with a **UID** and a **GID**.
- UIDs and GIDs indicate the process owner.
- Generally, the UID and GID values associated with a process are the same as the UID and GID of the user who has started the process.

A process consists of an *address space* and a *metadata object*. The process space is related to all the *memory and swap space* that a process consumes. The process metadata is just an entry in the kernel's process table and stores all other information about a process.

Process States

- A process may be in any one of the following states:
  - **D**: Uninterruptible sleep (usually IO)
  - **R:**Running or runnable (on run queue)
  - **S:**Interruptible sleep (waiting for an event to complete)
  - **T:** Stopped, either by a job control signal or because it is being traced
  - **Z:**Defunct ("zombie") process, terminated but not reaped by its parent

The process state can be displayed using the **ps** command. For BSD formats and when the stat keyword is used, additional state information is displayed such as the following:

- **<:** High-priority (not nice to other users)
- **N:** Low-priority (nice to other users)
- **L:**Has pages locked into memory (for real time and custom IO)
- **s:** Is a session leader
- **l:**Is multithreaded
- **+**: Is in the foreground process group

Note: niceis a useful program that is used to lower or increase the scheduling priority of a process or batch processes. Users can assign nice values between **0 (no effect) and 19**(greatest effect). *The higher the nice value, the lower the scheduling priority.*

Process Subsystems

Each time we boot a system, execute a command, or start an application, the system activates one or more processes.

• A process as it runs uses the resources of the various subsystems:

- Disk I/O
- Network
- Memory
- CPU

A process, as it runs, uses the resources of the various subsystems:

• **The disk I/O subsystem:**Controls disk utilization and resourcing as well as file system performance.

• **The network subsystem:**Controls the throughput and directional flow of data between systems over a network connection.

• **The memory subsystem:**Controls the utilization and allocation of physical, virtual, and shared memory.

• **The CPU subsystem:**Controls CPU resources, loading, and scheduling.

If not monitored and controlled, processes can consume our system resources, causing the system to run slowly and in some cases even halt. The Linux kernel collects performance-relevant statistics on each of these subsystems, to include process information. We can view and use this information to assess the impact that the processes have on the subsystem resources.

Listing System Processes

• The process status (**ps**) command lists the processes that are associated with the shell.

• For each process, the **ps**command displays the **PID**, the terminal identifier (**TTY**), the cumulative execution time (**TIME**), and the command name (**CMD**).

• For example, to list the currently running processes on the system using the **ps** command.

```
[root@krosum Temp]# ps
  PID TTY          TIME CMD
 1195 pts/0    00:00:00 bash
25269 pts/0    00:00:00 ps
[root@krosum Temp]#
```

The **ps** command has several options that you can use to display additional process information.

• -a: Prints information about all processes most frequently requested, except process group leaders and processes not associated with a terminal.

• -e: Prints information about all the processes that currently running.

• -f:Generates a full listing.

• -l: Generates a long listing

• -o format: Writes information according to the format specification given in a format.

Examples required for –o option:

```
[root@krosum Temp]# ps
  PID TTY          TIME CMD
 1195 pts/0    00:00:00 bash
25270 pts/0    00:00:00 ps
[root@krosum Temp]#
[root@krosum Temp]# ps -o pid,cmd
  PID CMD
 1195 bash
25271 ps -o pid,cmd
[root@krosum Temp]# ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
root      1195  1191  0 12:52 pts/0    00:00:00 bash
root     25273  1195  0 14:57 pts/0    00:00:00 ps -f
[root@krosum Temp]#
[root@krosum Temp]# ps -fo pid,ppid,stime,cmd
  PID  PPID STIME CMD
 1195  1191 12:52 bash
25275  1195 14:58  \_ ps -fo pid,ppid,stime,cmd
```

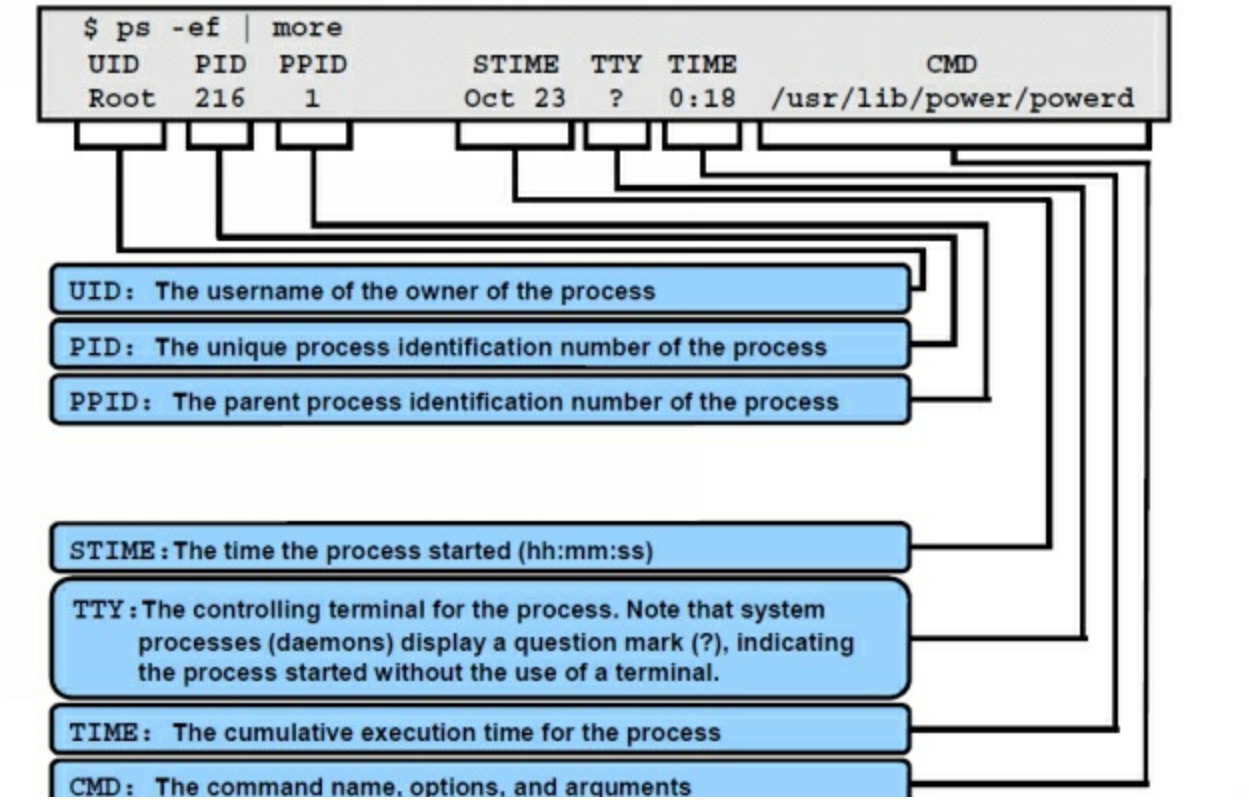
Also multiple -o options can be specified. The format specification is interpreted as the space-character-separated concatenation of all the format option arguments.

Using –o option we can filter specific fields from process command.

Listing All Processes

For example, use the **ps -ef**command to list all the processes currently scheduled to run on the system.

ps –ef|**more**



The illustration in the snap interprets the output of the **ps –ef** command.

• The first column is the **UID**, the username of the owner of the process.

• The second column is the **PID**, the unique process identification number of the process.

• The third column is the **PPID**, the parent process identification number of the process.

• The fourth column is the **STIME**, the time the process started.

• The fifth column is the **TTY**, the controlling terminal for the process. Note that system processes (daemons) display a question mark (**?**).

• The sixth column is the **TIME**, the cumulative execution time for the process.

• The seventh column is the **CMD**, the command name, options, and arguments.

Terminating a Process

• There might be times when we need to terminate an unwanted process.

• A process might have got into an endless loop, or it might have hung.

• We can kill or stop any process that we own.

• We can use the following two commands to terminate one or more processes:

- **kill**
- **pkill**

• The kill and **pkill** commands send signals to processes directing them to terminate.

• Each signal has a number, name, and an associated event.

However, there are processes that should not be terminated, such as the init process. Killing such processes can result in a system crash.

Note: A super-user can kill any process in the system.

Terminating a Process: kill Command

- We can terminate any process by issuing the appropriate signal to the process concerned.
- The kill command sends a termination signal to one or more processes.

Note: The kill command terminates only those processes that we own.

The **kill** command sends *signal 15*, the terminate signal, by default. This signal causes the process to terminate in an orderly manner.

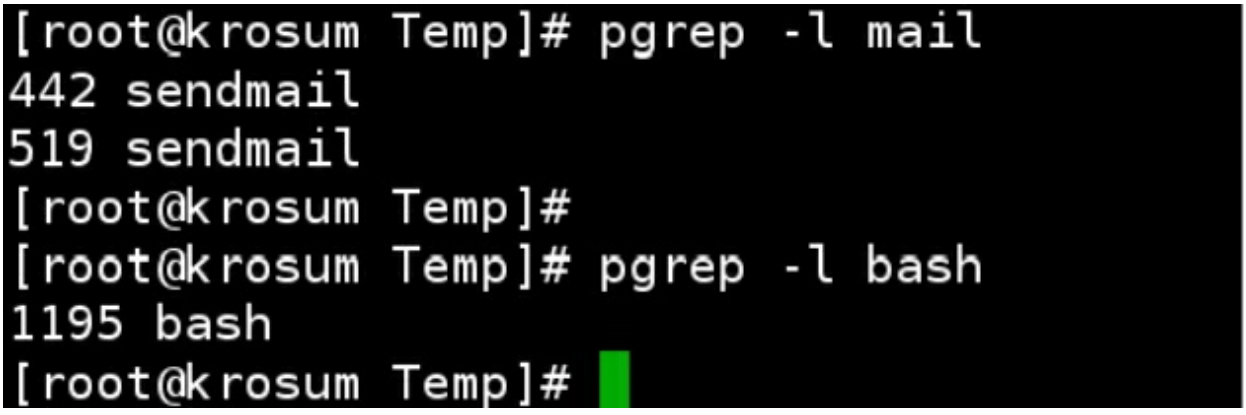
All we need to know *the PID of the process before to terminate* it. We can use either the **ps**or **pgrep**command to locate the PID of the process.

Also, we can terminate several processes at the same time by entering multiple PIDs on a single command line.

Note: The root user can use the kill command on any process.

Other way of terminating a Process: using **kill**Command

Use the kill command to terminate the mail process.



Terminating a Process: pkill Command

Alternatively, we can use the **pkill**command to send termination signal to processes.

pkill[-options]pattern

The **pkill**command requires us to specify the name instead of the **PID** of the process.

Forcefully Terminating a Process:

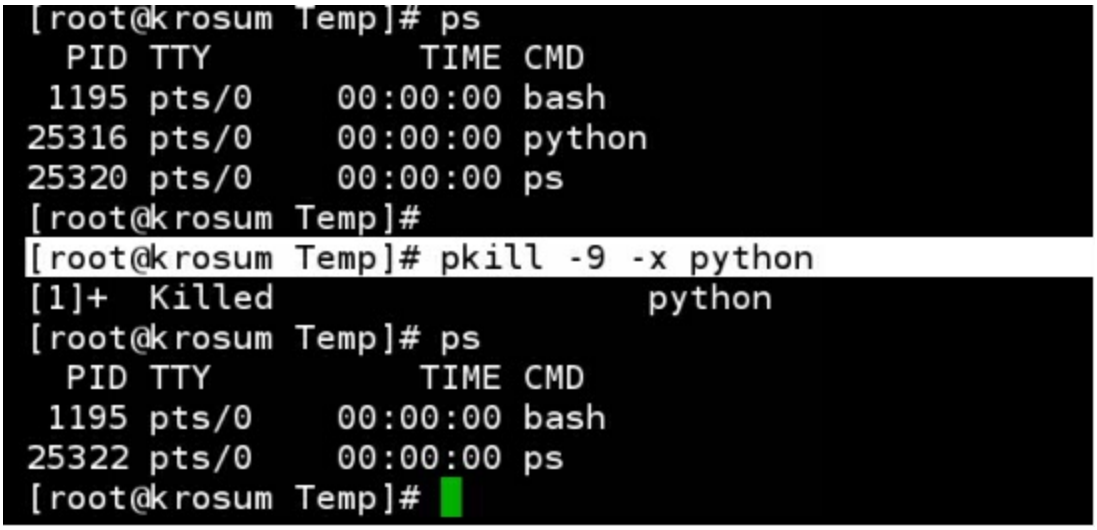
```
[root@krosum Temp]# kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
[root@krosum Temp]#
```

Some processes ignore the default signal 15 that the kill command sends.

• If a process does not respond to signal 15, you can force it to terminate by using *signal 9* with the kill or **pkill**command.

```
[root@krosum Temp]# ps
  PID TTY          TIME CMD
 1195 pts/0    00:00:00 bash
25312 pts/0    00:00:00 python
25314 pts/0    00:00:00 ps
[root@krosum Temp]#
[root@krosum Temp]# # To kill a python process
[root@krosum Temp]# # -------------------------
[root@krosum Temp]#
[root@krosum Temp]# kill -9 25312
[1]+  Killed                  python
[root@krosum Temp]# ps
  PID TTY          TIME CMD
 1195 pts/0    00:00:00 bash
25315 pts/0    00:00:00 ps
```

(or)



Note : Sending *signal 15 does not necessarily kill a process gracefully.* Only if the signal is caught by the process, it cleans itself up in order and dies. If not, it just dies.

Caution

Use the **kill -9 command**only when necessary. When you use the kill -9 command on an active process, the process terminates instantly. Using signal 9 on processes that control databases or programs that update files could cause data corruption.

Performing Basic Process Control

Process control Block (PCB) (or) *Task control block is a data structure* that contains information about **process register**,**state,program counter, priority, register, memory limits, list of open files** etc.,

In Linux Process table is a collection of PCB's that means logically contains a PCB for all of the current processes in the system.

```
[root@krosum Temp]# # Listing Current Process :-
[root@krosum Temp]# # -------------------------
[root@krosum Temp]# ps
  PID TTY          TIME CMD
 1195 pts/0    00:00:00 bash
25339 pts/0    00:00:00 ps
[root@krosum Temp]#
```

This practice covers the following topics:

1. **Display current process**

2. **Display list of all process**

ps–e (or) ps-A

```
[root@krosum Temp]# ps -e
  PID TTY          TIME CMD
    1 ?        00:00:01 systemd
    2 ?        00:00:00 kthreadd
    3 ?        00:00:00 ksoftirqd/0
    5 ?        00:00:00 kworker/0:0H
    6 ?        00:00:00 kworker/u:0
    7 ?        00:00:00 kworker/u:0H
    8 ?        00:00:00 migration/0
    9 ?        00:00:00 rcu_bh
   10 ?        00:00:00 rcu_sched
   11 ?        00:00:00 watchdog/0
   12 ?        00:00:00 khelper
   13 ?        00:00:00 kdevtmpfs
   14 ?        00:00:00 netns
   15 ?        00:00:00 bdi-default
   16 ?        00:00:00 kintegrityd
   17 ?        00:00:00 kblockd
   18 ?        00:00:00 ata_sff
```

3. **Display list of process in BSD format**

```
[root@krosum Temp]# ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.4   7964  5116 ?        Ss   12:46   0:01 /usr/lib/systemd/systemd --swit
root         2  0.0  0.0      0     0 ?        S    12:46   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S    12:46   0:00 [ksoftirqd/0]
root         5  0.0  0.0      0     0 ?        S<   12:46   0:00 [kworker/0:0H]
root         6  0.0  0.0      0     0 ?        S    12:46   0:00 [kworker/u:0]
root         7  0.0  0.0      0     0 ?        S<   12:46   0:00 [kworker/u:0H]
root         8  0.0  0.0      0     0 ?        S    12:46   0:00 [migration/0]
root         9  0.0  0.0      0     0 ?        S    12:46   0:00 [rcu_bh]
root        10  0.0  0.0      0     0 ?        S    12:46   0:00 [rcu_sched]
root        11  0.0  0.0      0     0 ?        S    12:46   0:00 [watchdog/0]
root        12  0.0  0.0      0     0 ?        S<   12:46   0:00 [khelper]
root        13  0.0  0.0      0     0 ?        S    12:46   0:00 [kdevtmpfs]
root        14  0.0  0.0      0     0 ?        S<   12:46   0:00 [netns]
root        15  0.0  0.0      0     0 ?        S    12:46   0:00 [bdi-default]
root        16  0.0  0.0      0     0 ?        S<   12:46   0:00 [kintegrityd]
```

4. **Select all processes owned by you**

```
[root@krosum Temp]# ps -x
  PID TTY      STAT   TIME COMMAND
    1 ?        Ss     0:01 /usr/lib/systemd/systemd --switched-root --system --deserialize 20
    2 ?        S      0:00 [kthreadd]
    3 ?        S      0:00 [ksoftirqd/0]
    5 ?        S<     0:00 [kworker/0:0H]
    6 ?        S      0:00 [kworker/u:0]
    7 ?        S<     0:00 [kworker/u:0H]
    8 ?        S      0:00 [migration/0]
    9 ?        S      0:00 [rcu_bh]
   10 ?        S      0:00 [rcu_sched]
   11 ?        S      0:00 [watchdog/0]
   12 ?        S<     0:00 [khelper]
   13 ?        S      0:00 [kdevtmpfs]
   14 ?        S<     0:00 [netns]
   15 ?        S      0:00 [bdi-default]
   16 ?        S<     0:00 [kintegrityd]
   17 ?        S<     0:00 [kblockd]
```

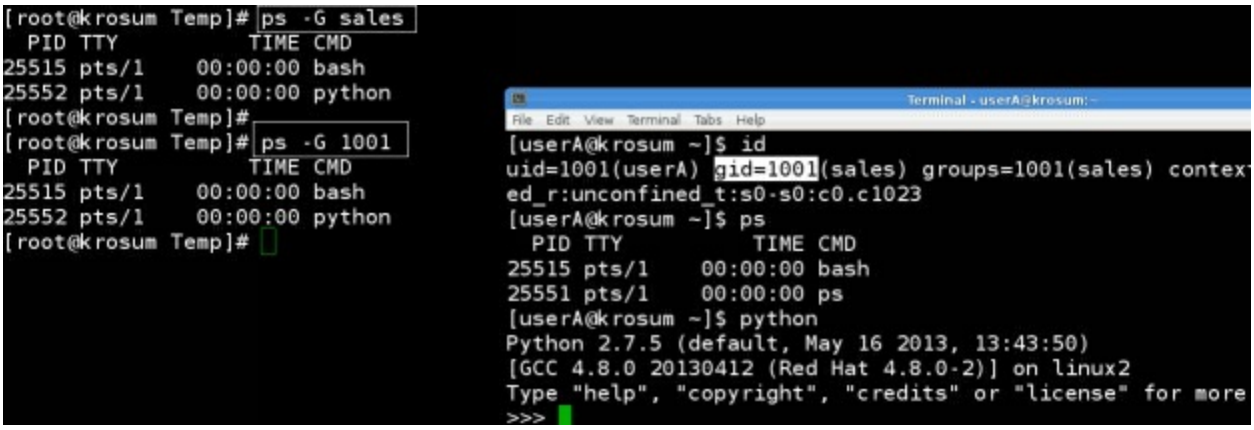
5. **To display a user's processes by real user ID (RUID) or name, use the -U flag.**

ps- u <username> (or) ps–u <UID>

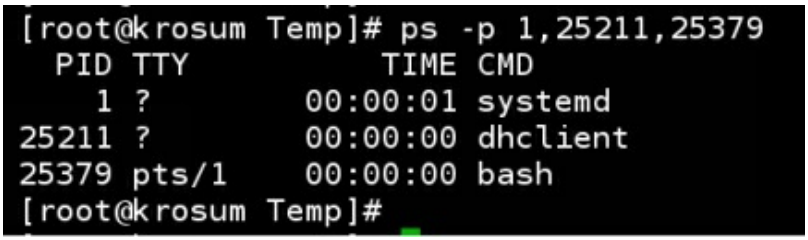
```
[root@krosum Temp]# ps
  PID TTY          TIME CMD
 1195 pts/0    00:00:00 bash
25444 pts/0    00:00:00 ps
[root@krosum Temp]#
[root@krosum Temp]# ps -u student
  PID TTY          TIME CMD
25397 pts/1    00:00:00 bash
25447 pts/1    00:00:00 ping
[root@krosum Temp]#
[root@krosum Temp]# ps -u 1000
  PID TTY          TIME CMD
25397 pts/1    00:00:00 bash
25447 pts/1    00:00:00 ping
[root@krosum Temp]#
```

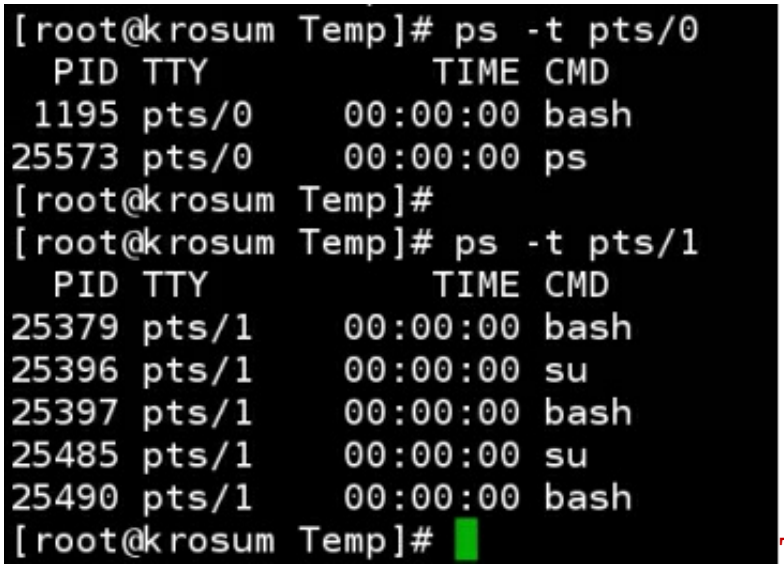
```
Terminal - student@
File  Edit  View  Terminal  Tabs  Help
[student@krosum ~]$ ps
  PID TTY          TIME CMD
25397 pts/1    00:00:00 bash
25445 pts/1    00:00:00 ps
[student@krosum ~]$
[student@krosum ~]$ id -u
1000
[student@krosum ~]$ ping 127.0.0.1 >r1
```

6. **List all processes owned by a certain group**



7. **Display Processes by PID**



8. 

To select processes by tty, use the -t flag as follows.

A process tree shows how processes on the system are linked to each other; processes whose parents have been killed are adopted by the init (or systemd).

```
[root@krosum Temp]# ps -e --forest
  PID TTY          TIME CMD
    2 ?        00:00:00 kthreadd
    3 ?        00:00:00  \_ ksoftirqd/0
    5 ?        00:00:00  \_ kworker/0:0H
    6 ?        00:00:00  \_ kworker/u:0
    7 ?        00:00:00  \_ kworker/u:0H
    8 ?        00:00:00  \_ migration/0
    9 ?        00:00:00  \_ rcu_bh
   10 ?        00:00:00  \_ rcu_sched
   11 ?        00:00:00  \_ watchdog/0
   12 ?        00:00:00  \_ khelper
   13 ?        00:00:00  \_ kdevtmpfs
   14 ?        00:00:00  \_ netns
   15 ?        00:00:00  \_ bdi-default
   16 ?        00:00:00  \_ kintegrityd
   17 ?        00:00:00  \_ kblockd
   18 ?        00:00:00  \_ ata_sff
   19 ?        00:00:00  \_ khubd
```

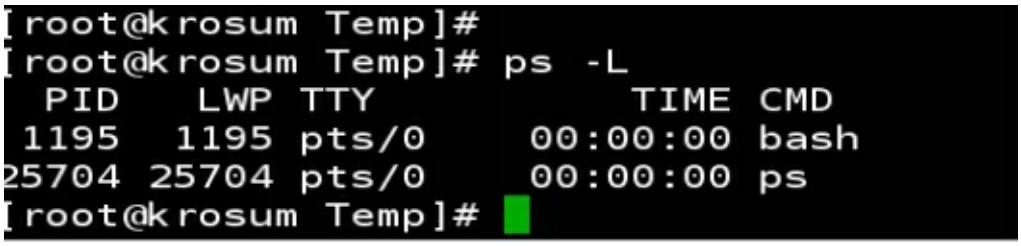
9. **Display process tree.**

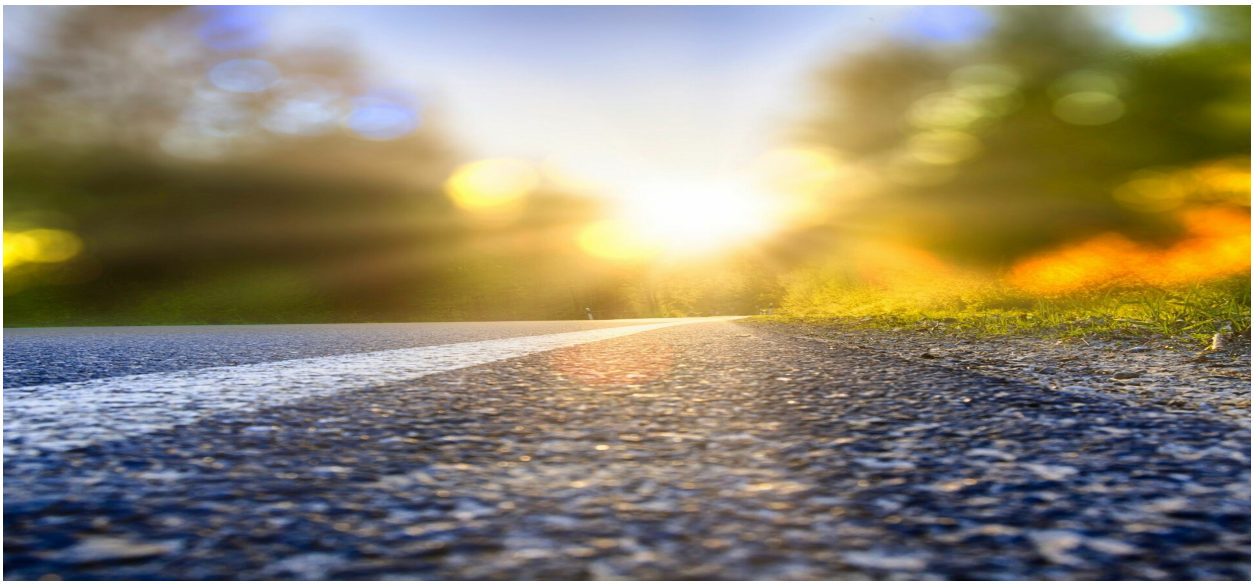
```
[root@krosum Temp]# pstree
systemd─┬─NetworkManager─┬─dhclient
        │                └─3*[{NetworkManager}]
        ├─Thunar
        ├─accounts-daemon───2*[{accounts-daemon}]
        ├─alsactl
        ├─applet.py───{applet.py}
        ├─at-spi-bus-laun─┬─dbus-daemon───{dbus-daemon}
        │                 └─3*[{at-spi-bus-laun}]
        ├─at-spi2-registr───{at-spi2-registr}
        ├─atd
        ├─auditd─┬─audispd─┬─sedispatch
        │        │         └─{audispd}
        │        └─{auditd}
        ├─avahi-daemon───avahi-daemon
        ├─blueman-applet───{blueman-applet}
        ├─bluetoothd
        ├─chronyd
        ├─crond
        ├─cupsd
```

10. print process tree for given process (ex: bash)

```
[root@krosum Temp]# ps -f --forest -C bash
UID          PID  PPID  C STIME TTY          TIME CMD
root       25490 25485  0 16:24 pts/1    00:00:00 -bash
student    25397 25396  0 16:09 pts/1    00:00:00 -bash
root       25379  1191  0 16:09 pts/1    00:00:00 bash
root        1195  1191  0 12:52 pts/0    00:00:00 bash
[root@krosum Temp]#
```

11. Print Process Threads (LWP) use –L option


[root@krosum Temp]#
[root@krosum Temp]# ps -L
PID LWP TTY TIME CMD
1195 1195 pts/0 00:00:00 bash
25704 25704 pts/0 00:00:00 ps
[root@krosum Temp]#



CHAPTER 14 - FILTERS

In Unix/Linux, a filter is any command that gets its input from the standard input stream, manipulates the input, and then sends the result to the standard output stream. Some filters can receive data directly from a file.

Common Filters

cut	**Passes only specified columns**
paste	**Combines columns**
cmp	**Compares two files**
comm	**Identifies common lines in two files**
diff	**Identifies differences between two files or between common files in two directories**
head	**Passes the number of specified lines at the beginning of the data**
tail	**Passes the number of specified lines at the end of the data**
sort	**Arranges the data in sequence**
tr	**Translate one or more characters as specified**
uniq	**Delete duplicate (continues repeated) lines**
grep	**Print matched pattern lines**
Find	**Search file and directories**

Filters and pipes

Unix/Linux filters depend on the input file or input pipe. All the filter command will read data from input file (or) pipe, stores it to buffer. From the buffer, filter command will execute and display command result to monitor.

cut command

The basic purpose of the cut command is *to extract one or more columns of data* from standard input or from the input files.

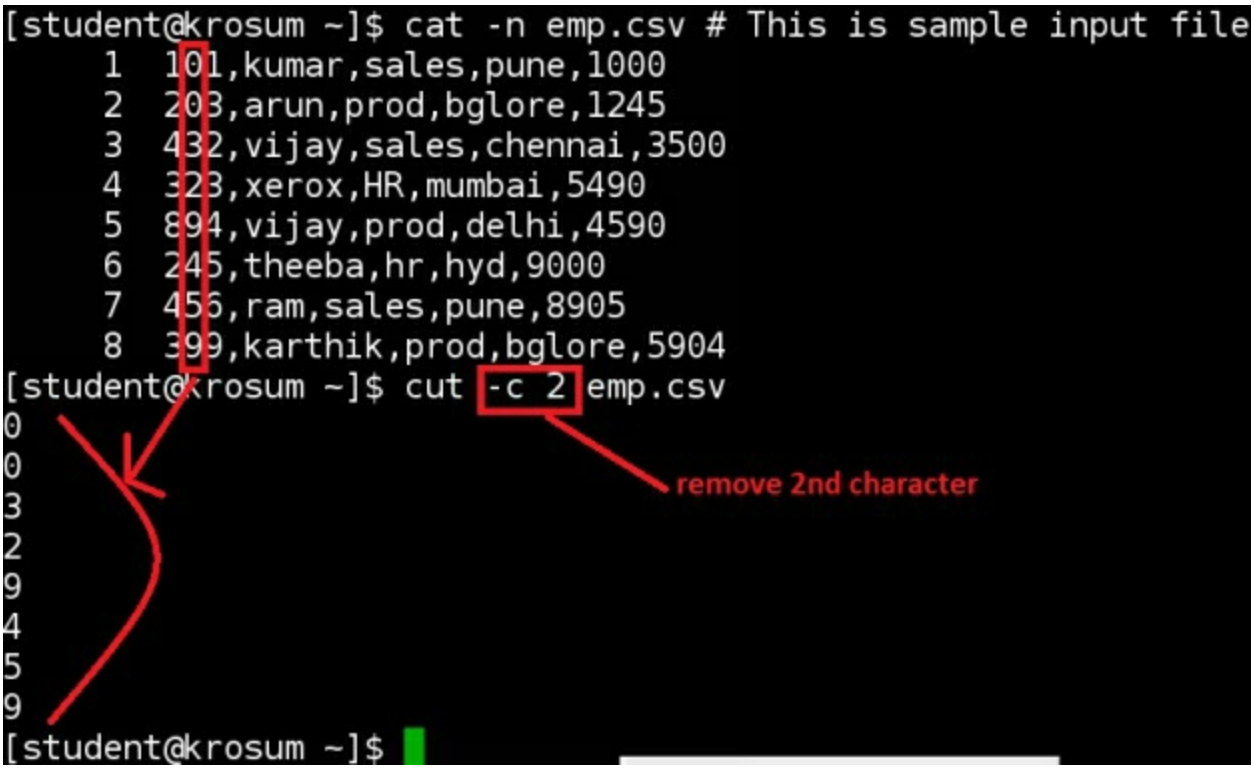
The syntax of the cut command

cut-optionfilter_rangeinputfile (or)

commandresult| cut–option filter_range

Here filter_range are numbers. We can *perform character based filtering* or field based filtering based on input field delimiter.

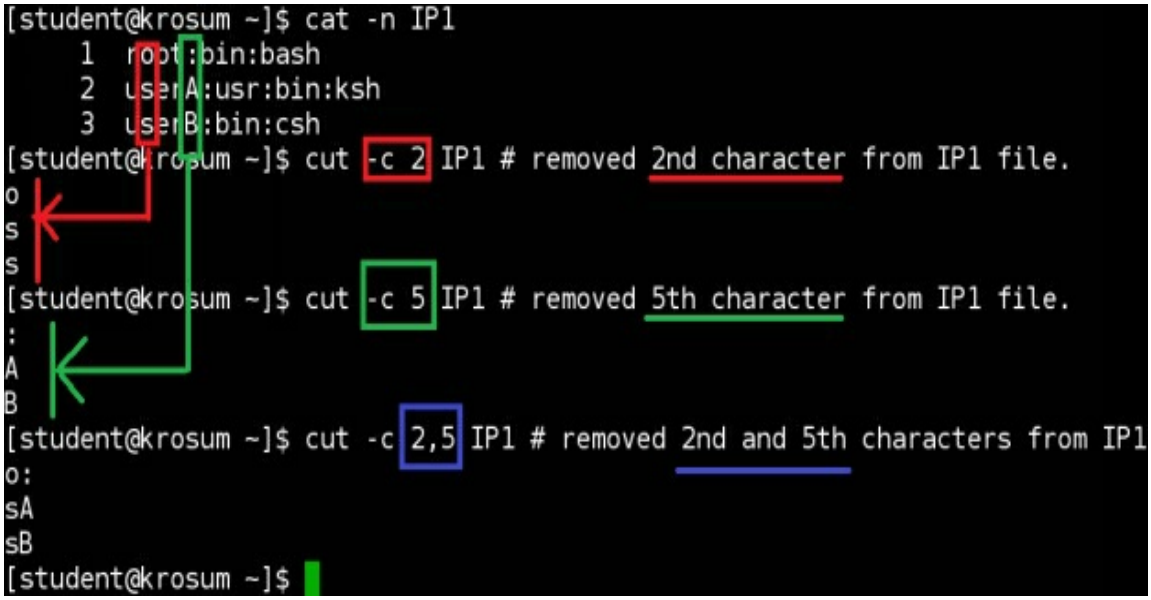
For character based filter, use **–c option**. See the below example snap that shows filtering **2nd characterfrom** input file (emp.csv). Character positions work well when the data are aligned in fixed columns.



```
[student@krosum ~]$ cat -n emp.csv
     1  101,kumar,sales,pune,1000
     2  203,arun,prod,bglore,1245
     3  432,vijay,sales,chennai,3500
     4  323,xerox,HR,mumbai,5490
     5  894,vijay,prod,delhi,4590
     6  245,theeba,hr,hyd,9000
     7  456,ram,sales,pune,8905
     8  399,karthik,prod,bglore,5904
[student@krosum ~]$
```

So *the cut command will display the removed result to monitor*. This action takes on the buffer space and not in input file (emp.csv). So there are no changes in input file (emp.csv) after removing the 2nd character from emp.csv file.

Specifying character positions



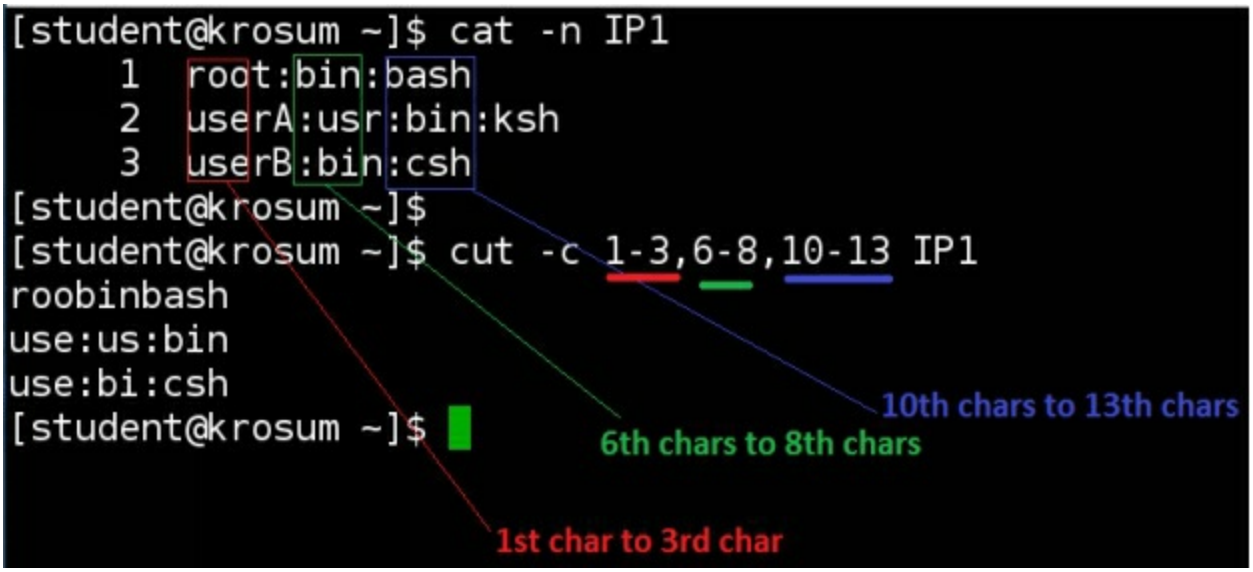
To specify that the file is formatted with fixed columns, we use the character option, **-c** followed by one or more column specifications. *A column specification can be one column or a range of columns in the format N-M,*

where N is the start column and M is the end column, inclusively. **Multiple columns are separated by commas.**

Note: here comma (,) is used as a separator.

```
[student@krosum ~]$ cat -n IP1
     1  root:bin:bash
     2  userA:usr:bin:ksh
     3  userB:bin:csh
[student@krosum ~]$ cut -c 2,5 IP1 # from 2nd character and 5th character
o:
sA
sB
[student@krosum ~]$ cut -c 2-5 IP1 # from 2nd chars to 5th chars
oot:
serA
serB
[student@krosum ~]$
```

Note the difference between comma (**,**) and hyphen (**-**) symbol in cut command where comma (**,**) is a separator and hyphen (**-**) is a range.



Note in the above example, both (**,**) and (**–**) special chars are used and from IP1 file we have filtered multiple range of characters.

1-3 means from 1st to 3rd characters

6-8 means from 6th to 8th characters

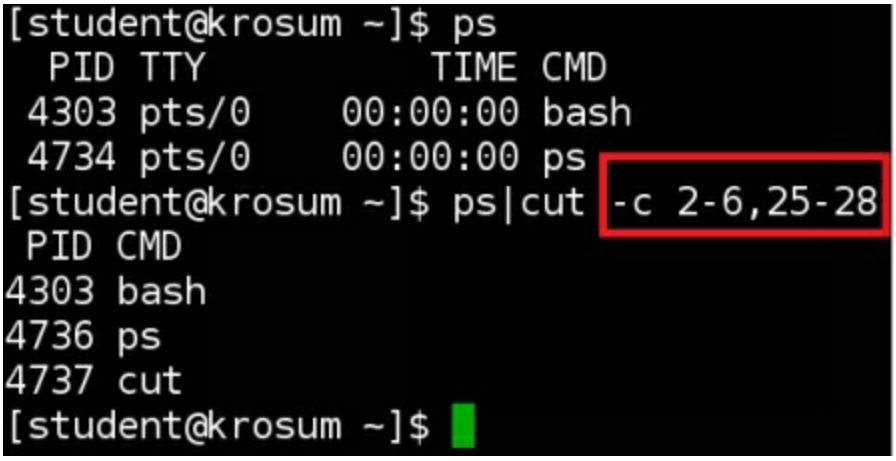
10-13 means from 10th to 13th characters

Using comma (**,**) we filtered different range of characters from IP1 file.

The previous example cuts the columns from a file.

But if the data are already in the input stream (pipe) from a previous

command operation, then there is no way to specify a filename. For example, using **ps**command, display process PID and process name list to monitor.



Field specification

While the column specification works well when the data are organized around fixed columns, it doesn't work in other situation.

For example see the below snap. The fields of file emp.csv are separated by comma (**,**). Here if we want to filter based on the 1st field (emp id), use –d option (-d delimiter).

So when the data are separated by (,) or any special characters, it is easier to use fields to extract the data from the file.

To specify a field, we use the field option (-f). Fields are numbered from the beginning of the line with the first being field number one. Like the character option, multiple fields are separated by commas with no space after the comma. Consecutive fields may be specified as a range.

```
[student@krosum ~]$ cat -n emp.csv
     1  101,kumar,sales,pune,1000
     2  203,arun,prod,bglore,1245
     3  432,vijay,sales,chennai,3500
     4  323,xerox,HR,mumbai,5490
     5  894,vijay,prod,delhi,4590
     6  245,theeba,hr,hyd,9000
     7  456,ram,sales,pune,8905
     8  399,karthik,prod,bglore,5904
[student@krosum ~]$ cut -d, -f 1 emp.csv
101
203
432
323
894
245
456
399
```

delimeter ,

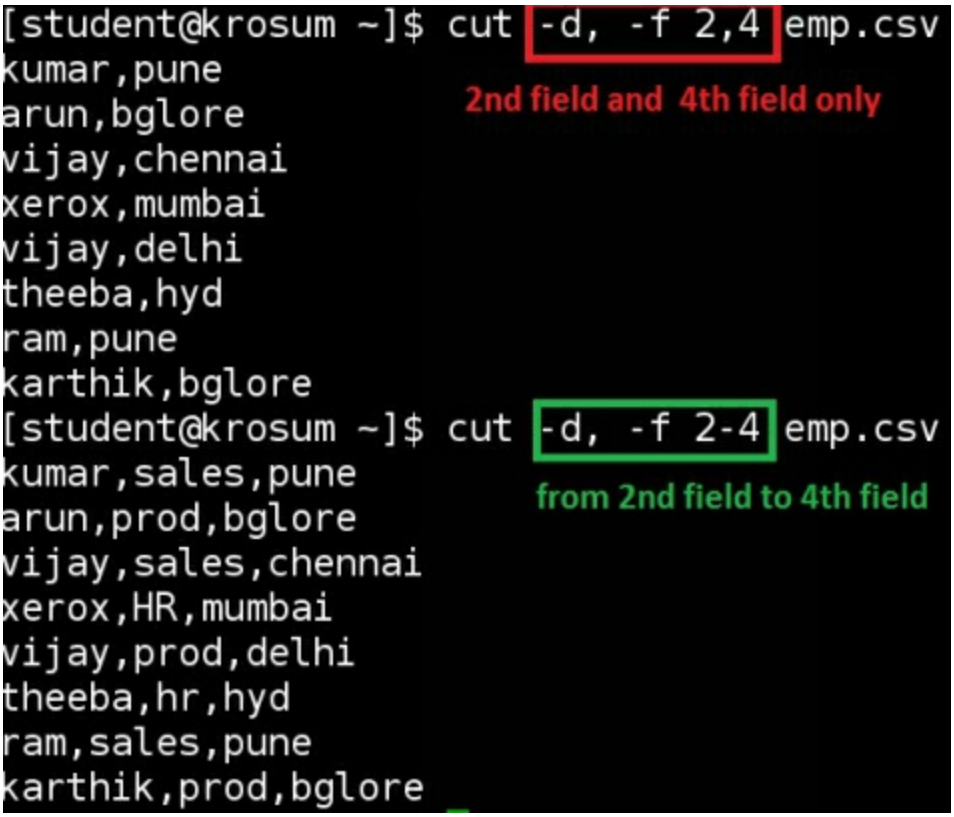
1st field

```
[student@krosum ~]$ cut -d, -f 2,4 emp.csv
kumar,pune
arun,bglore
vijay,chennai
xerox,mumbai
vijay,delhi
theeba,hyd
ram,pune
karthik,bglore
[student@krosum ~]$
```

2nd field and 4th field

How to filter employee name, and employee working city name ?

How to filter from 2nd field to 4th field(range) from emp.csv file?

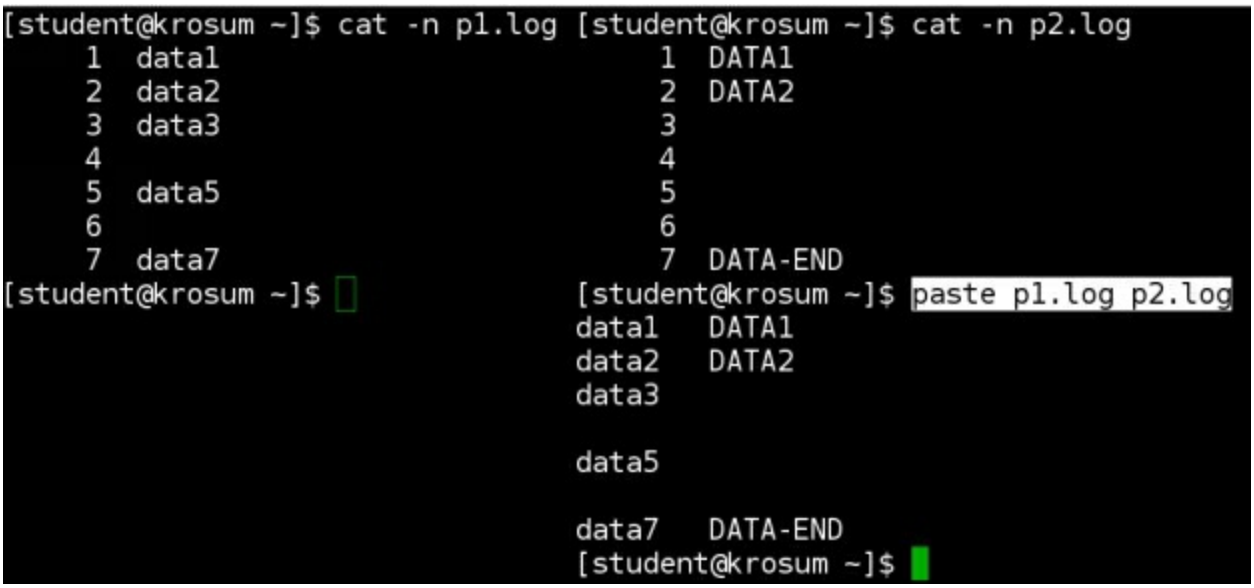

[student@krosum ~]$ cut -d, -f 2,4 emp.csv
kumar,pune
arun,bglore
vijay,chennai
xerox,mumbai
vijay,delhi
theeba,hyd
ram,pune
karthik,bglore
2nd field and 4th field only
[student@krosum ~]$ cut -d, -f 2-4 emp.csv
kumar,sales,pune
arun,prod,bglore
vijay,sales,chennai
xerox,HR,mumbai
vijay,prod,delhi
theeba,hr,hyd
ram,sales,pune
karthik,prod,bglore
from 2nd field to 4th field

Paste command

The paste command combines lines together. It gets its input from two or more files.

Syntax

pastefilename1 filename2 .. filename



Using **paste**command we can combine multiple files.

paste command treats each file as a column. If there are more than two files, the corresponding lines from each file, separated by tabs, are written to the output stream. The above *input file p1.log* file contains 7lines along including 2 empty lines (line 4 and line 6). The *input file p2.log* file contains 7lines including 5 empty lines. So when we combine p1.log and p2.log file together, the input file p1.log becomes 1st column and file p2.log becomes 2nd column.

Note : the cat and paste commands are similar for combining multiple files. However the cat command combines files vertically (by lines) whereas *the paste command combines files horizontally (by columns).*

```
[student@krosum ~]$ cat p1.log p2.log
data1
data2
data3

data5

data7
DATA1
DATA2
```

```
[student@krosum ~]$ paste p1.log p2.log
data1   DATA1
data2   DATA2
data3

data5

data7   DATA-END
[student@krosum ~]$
```

```
[student@krosum ~]$ cat -n IP1
     1  data1
     2  data2
     3  data3
     4  data4
     5  data5
[student@krosum ~]$ cat -n IP2
     1  DATA1
     2  DATA2
[student@krosum ~]$ paste IP1 IP2
data1   DATA1
data2   DATA2
data3
data4
data5
[student@krosum ~]$
```

If the files differ in its length–that is, if each file contains different number of lines, then all data are still written to the output with a delimiter. For example, if the first file is longer than the second file,paste writes the extra data from the first file with a separation delimiter, such as tab.

If the first file is shorter than the second file, paste writes a delimiter followed by the extra data from the second file to the output stream. The below example shows input file IP2 and IP1 combined. See the highlighted box for the above case.

```
[student@krosum ~]$ cat -n IP1
     1  data1
     2  data2
     3  data3
     4  data4
     5  data5
[student@krosum ~]$ cat -n IP2
     1  DATA1
     2  DATA2
[student@krosum ~]$ paste IP1 IP2
data1   DATA1
data2   DATA2
data3
data4
data5
[student@krosum ~]$
```

```
[student@krosum ~]$ paste IP2 IP1
DATA1   data1
DATA2   data2
        data3
        data4
        data5
[student@krosum ~]$
```

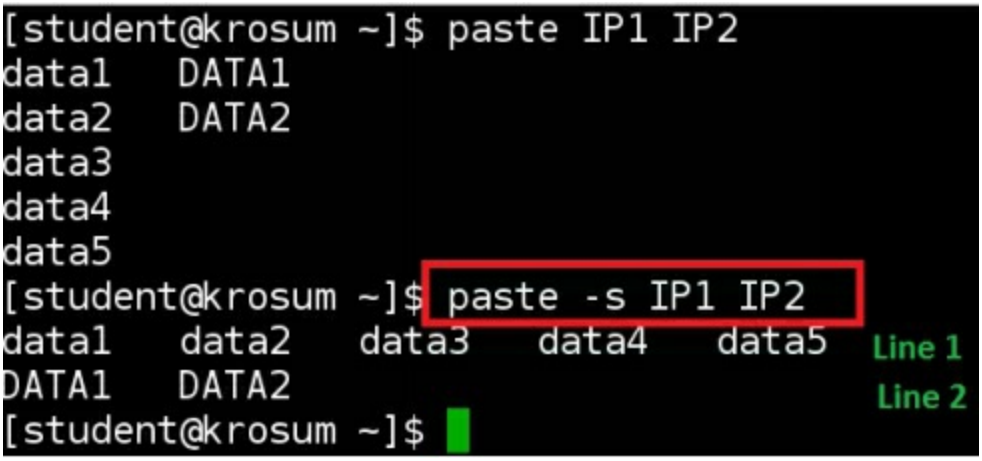
```
[student@krosum ~]$ paste IP1 IP2
data1   DATA1
data2   DATA2
data3
data4
data5
[student@krosum ~]$ paste -d: IP1 IP2
data1:DATA1
data2:DATA2
data3:
data4:
data5:
[student@krosum ~]$
```

Option –d

We can specify the delimiters using **-d option**to separate the data.

Option –s

The option –s converts each file to single line format. Input file1 is placed at line1 and input file2 is placed at line2


[student@krosum ~]$ paste IP1 IP2
data1 DATA1
data2 DATA2
data3
data4
data5
[student@krosum ~]$ paste -s IP1 IP2
data1 data2 data3 data4 data5
DATA1 DATA2
[student@krosum ~]$
Line 1
Line 2

sort command

The *sort* command is one of the important filter commands in Linux.

When dealing with large volume of data, we need to organize them for analysis and efficient processing. One of the simplest and powerful organizing techniques is sorting.

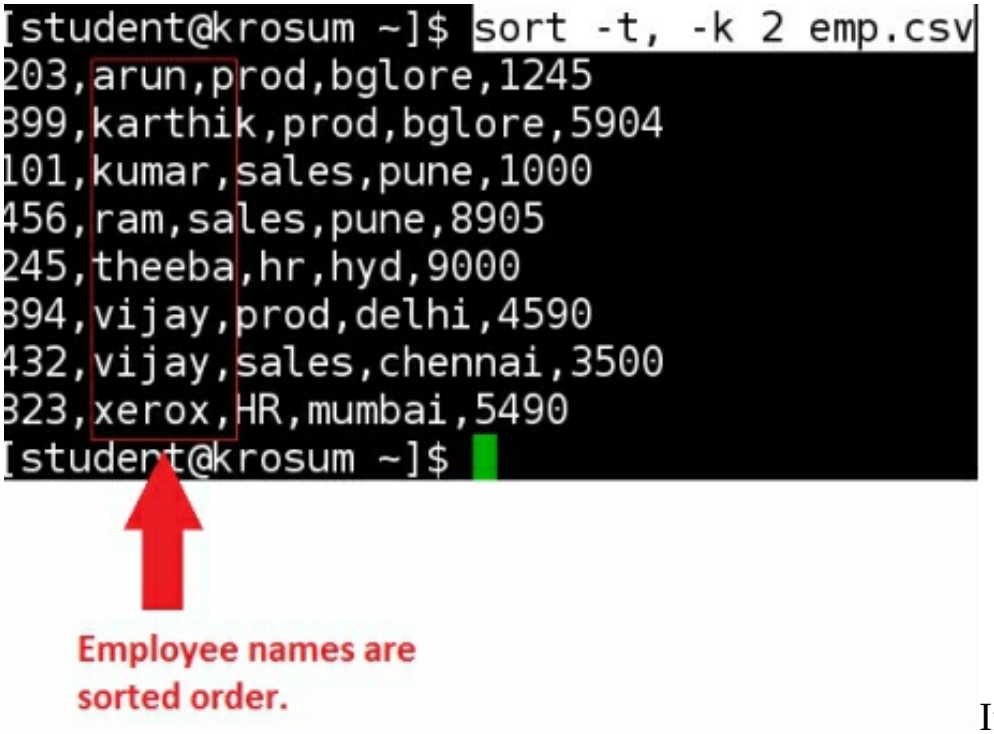
When *we sort data, we arrange them in sequence*. Usually, we use ascending order. *By default all elements are sorted based on the first character of the elements in ascending order*. We can also sort them in descending order, in which each piece of data is smaller than its predecessor.

Sort uses the ASCII value of each character.

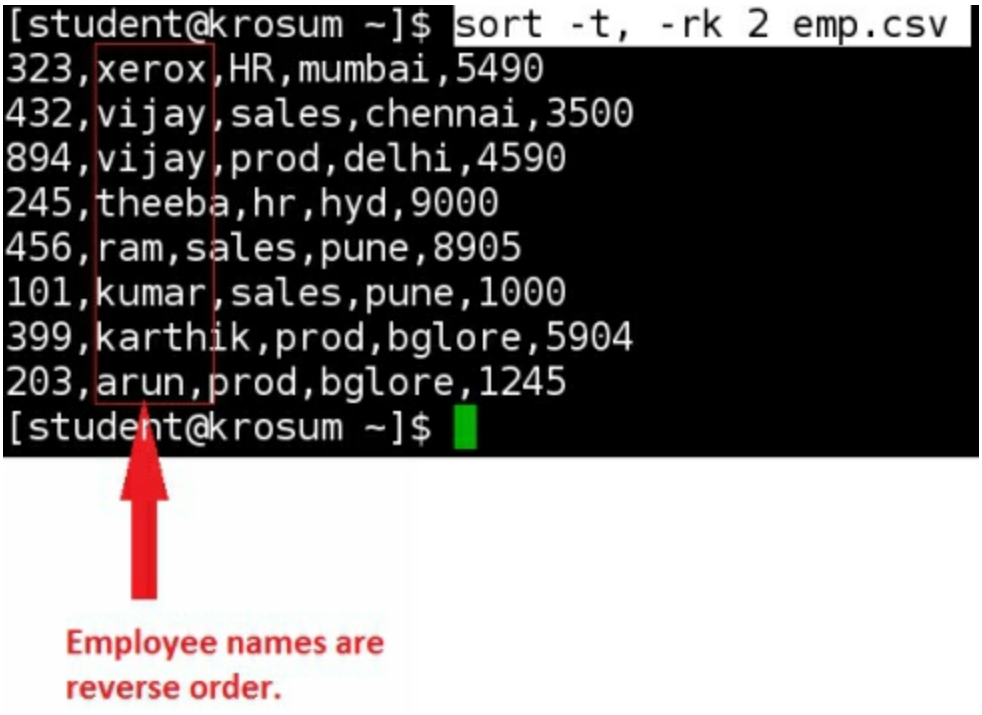
If you want to sort them in reverse order, use **–r (reverse)**option

```
[student@krosum ~]$ sort -r names.txt
xerox
vijay
vijay
theeba
ram
kumar
karthik
arun
[student@krosum ~]$
```

Sort based on the fields



If we want to sort the content based on the input field separator, there is an option called **–t (delimiter)**. Once the delimiter is identified, all the data are separated in to multiple columns and using **–k** option, we can specify the particular column where the sorting is to be done.



See the above snap we used multiple options. –t to specify the delimiter to be specified in input file. Here the delimiter used is comma (**,**) .

-k (key) field value (-k 2 – 2nd field) the option –r (reverse) descending order.

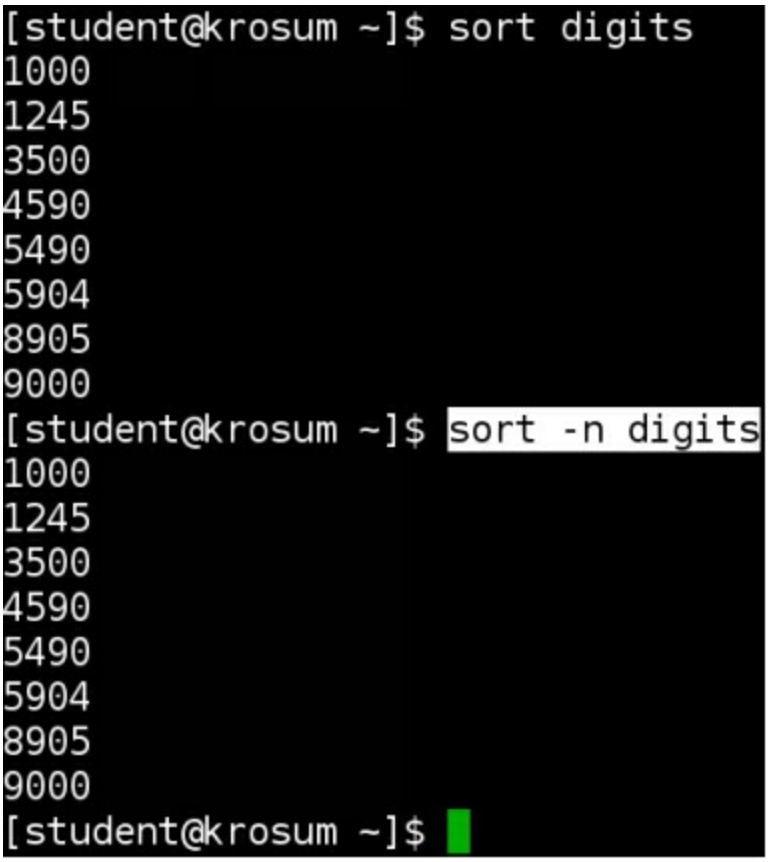
The above example (**sort–t, -rk2 emp.csv**) display employee names in descending order format.

Numerical sort fields

Sort filter considers *every data in ASCII format*. In other words it sorts data as though they are strings of characters.

In the below example, input file contains numbers (digits) which when sorted displays the numbers in ascending order (ASCII code) format.

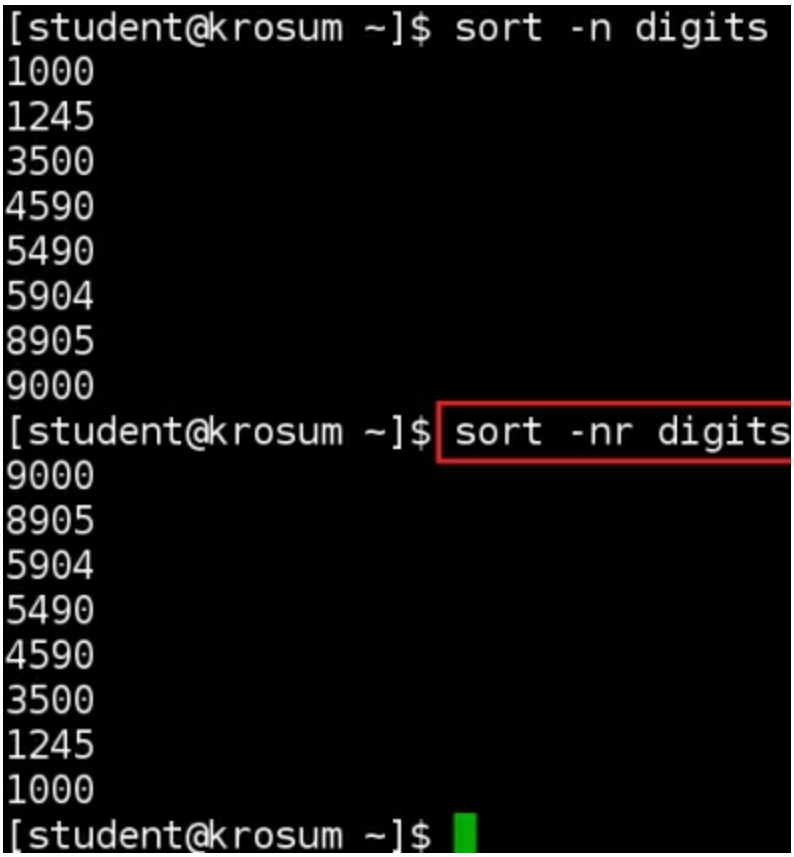
Using –n (numeric) option, we can perform numerical sorting.



See the above input file (digits) sort–n option displayed in numerical sorting.

Reverse (Descending) order

To sort the data from largest to smallest value (descending order), we can combine –n and –r options.



Task :From the below sample file emp.csv file, we can understand how to sort employee salary amount in descending order (largest to smallest)?

```
[student@krosum ~]$ sort -t, -nrk5 emp.csv
245,theeba,hr,hyd,9000
456,ram,sales,pune,8905
399,karthik,prod,bglore,5904
323,xerox,HR,mumbai,5490
894,vijay,prod,delhi,4590
432,vijay,sales,chennai,3500
203,arun,prod,bglore,1245
101,kumar,sales,pune,1000
[student@krosum ~]$
```

-tinput file data are separated by comma (,) – delimiter

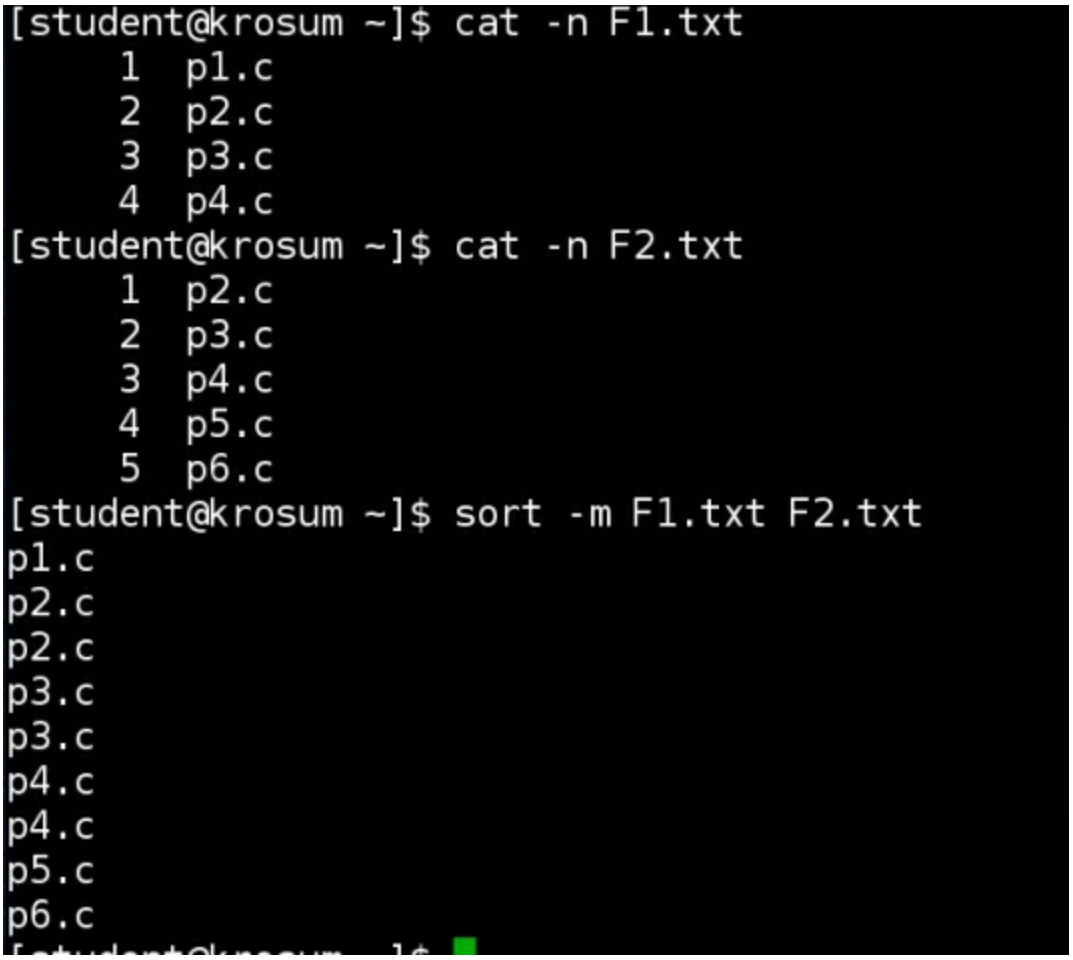
-nnumerical sort

-rreverse order

-k(key) field value , our input file salary (digits) are 5th field –k5

Merge files

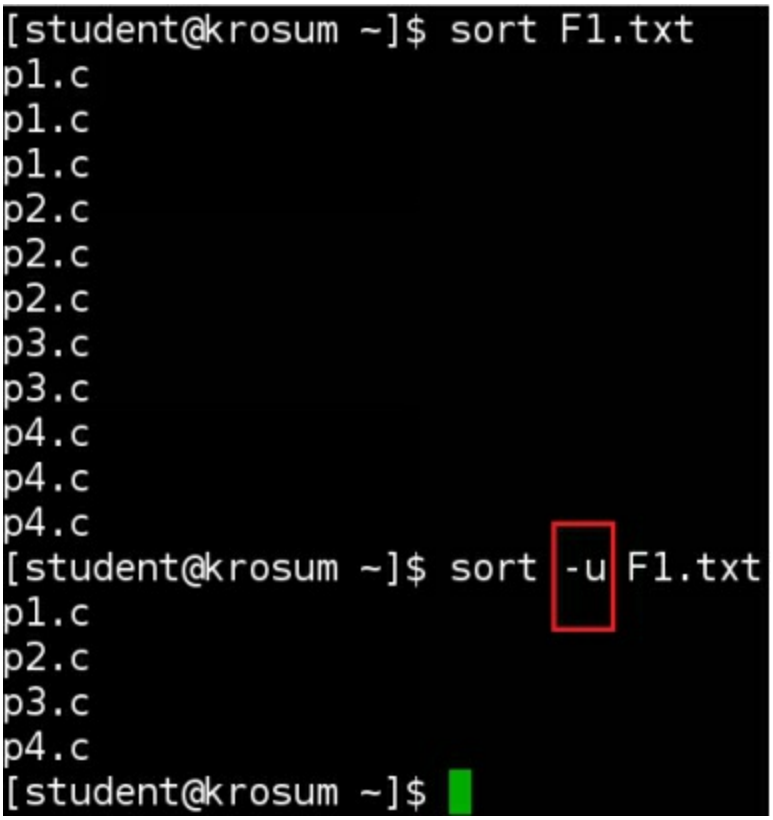
A merge combines multiple sorted files into single sorted file. If we know that the files are already ordered, we can save time by using the merge option (**-m**). However, that if the files are not ordered, sort will not give you an error message.



The above input files F1.txt and F2.txt both are sorted order format.

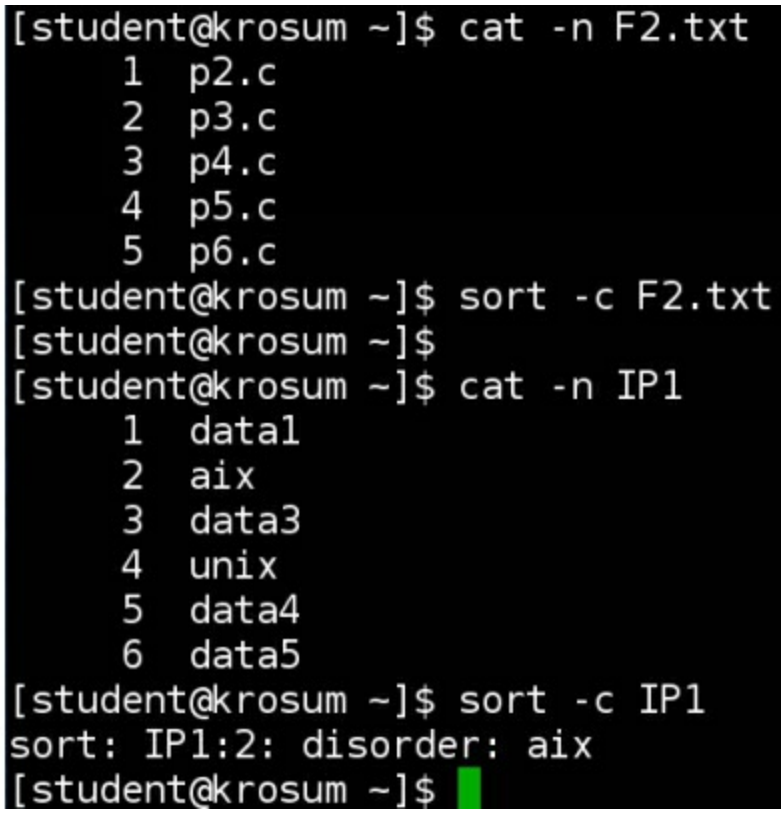
Unique Sort Fields

The unique option (**-u**) eliminates the identical fields. See the below example.



Check sequence (-c)

Verifies that data are correctly sorted or not. Using **–c** option, if *input file content is not in sorted order, command will display disorder element.* If input *file is in sorted order, it will print an empty line.*



```
[student@krosum ~]$ cat emp.csv
101,kumar,sales,pune,1000
203,arun,prod,bglore,1245
432,vijay,sales,chennai,3500
323,xerox,HR,mumbai,5490
894,vijay,prod,delhi,4590
245,theeba,hr,hyd,9000
456,ram,sales,pune,8905
399,karthik,prod,bglore,5904
```

Now we got an idea about the usage of cutcommand and sortcommand. Based on the use case, we can combine multiple filter commands in single pipe line.

```
[student@krosum ~]$ cat emp.csv |cut -d, -f 2 |sort
arun
karthik
kumar
ram
theeba
vijay
vijay
xerox
```

```
[student@krosum ~]$ cat emp.csv |cut -d, -f 2 # filter emp name
kumar
arun
vijay
xerox
vijay
theeba
ram
karthik
[student@krosum ~]$
```

Display employee names which are in sorted order (Ascending order)

```
[student@krosum ~]$ cat emp.csv |cut -d, -f 2 |sort -r
xerox
vijay
vijay
theeba
ram
kumar
karthik
arun
[student@krosum ~]$
```

Display employee names which are in descending order.

```
[student@krosum ~]$ cut -d, -f 2 emp.csv |sort
arun
karthik
kumar
ram
theeba
vijay
vijay
xerox
[student@krosum ~]$ sort -t, -k 2 emp.csv |cut -d, -f 2
arun
karthik
kumar
ram
theeba
vijay
vijay
xerox
```

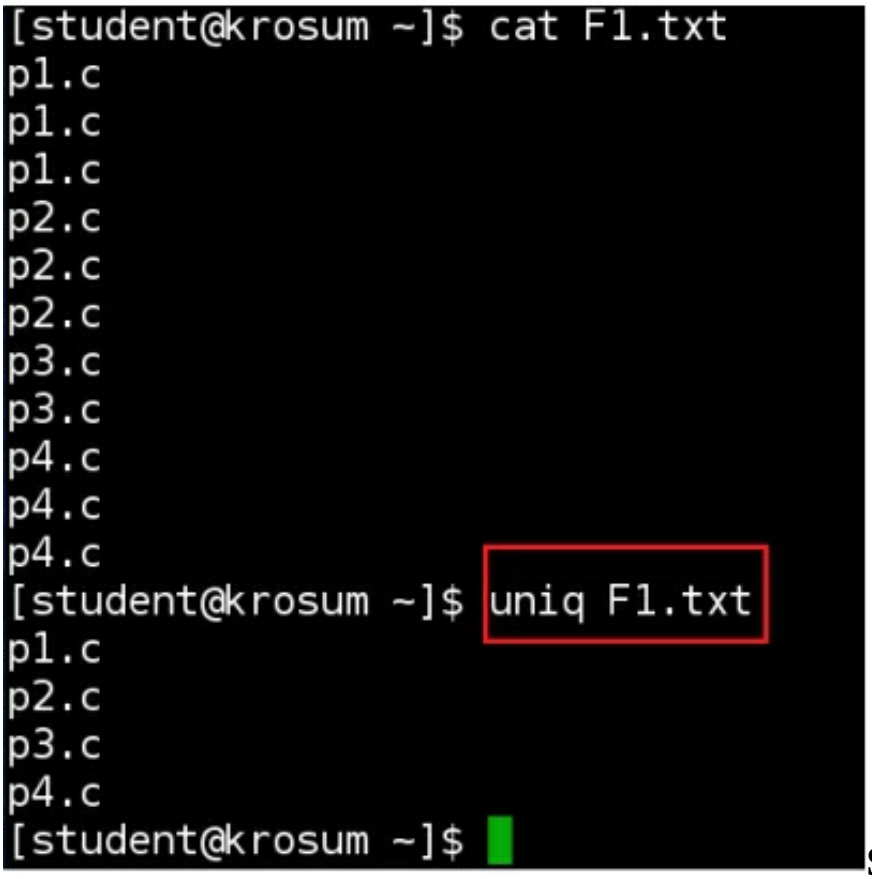
process 1 (cut)
stdout to PIPE

process 2 (sort)
STDIN(reading) from PIPE

We can combine **cut**| **sort**(or) **sort**| **cut**in any order. See the below snap where both possibility results are same.

uniq command

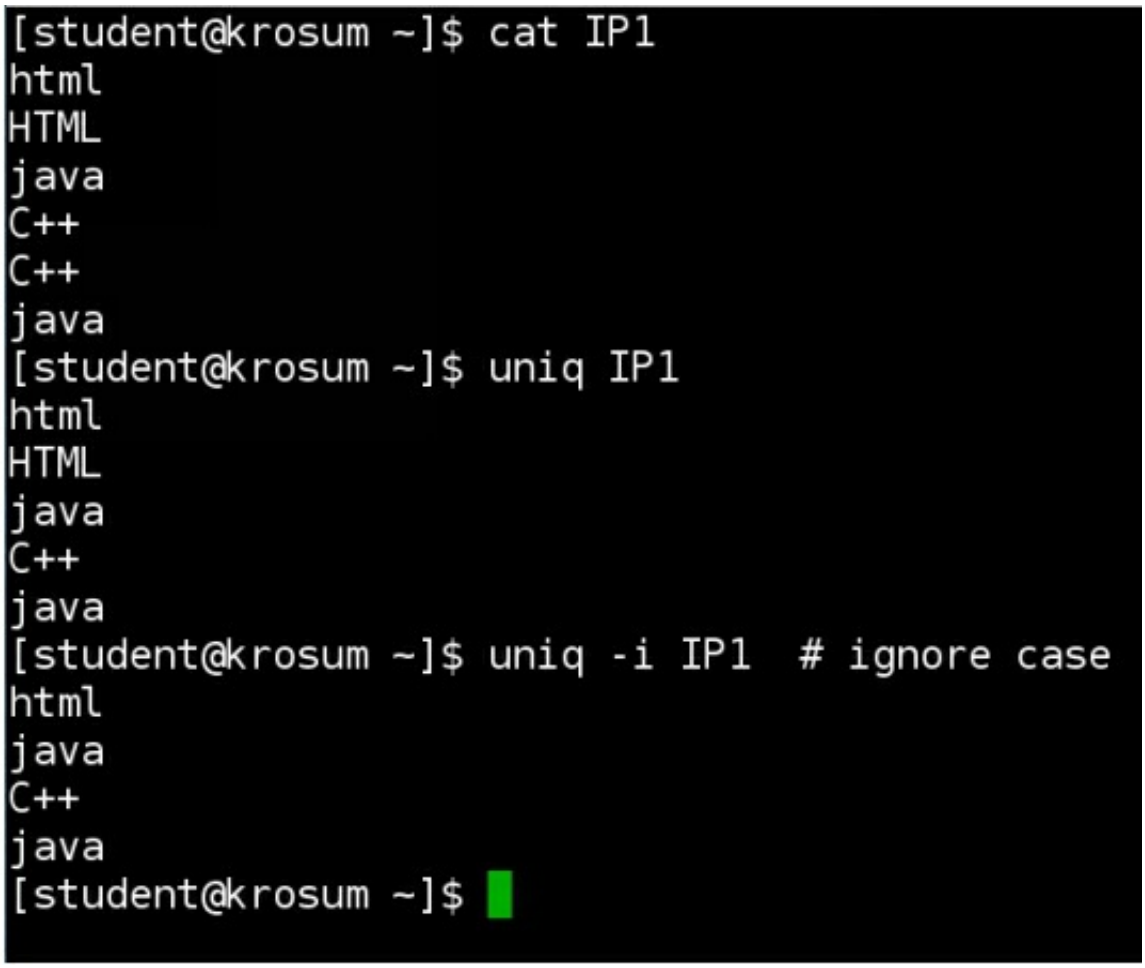
The **uniq**command *deletes the duplicate lines* but doesn't sort. It is case sensitive.



Syntax

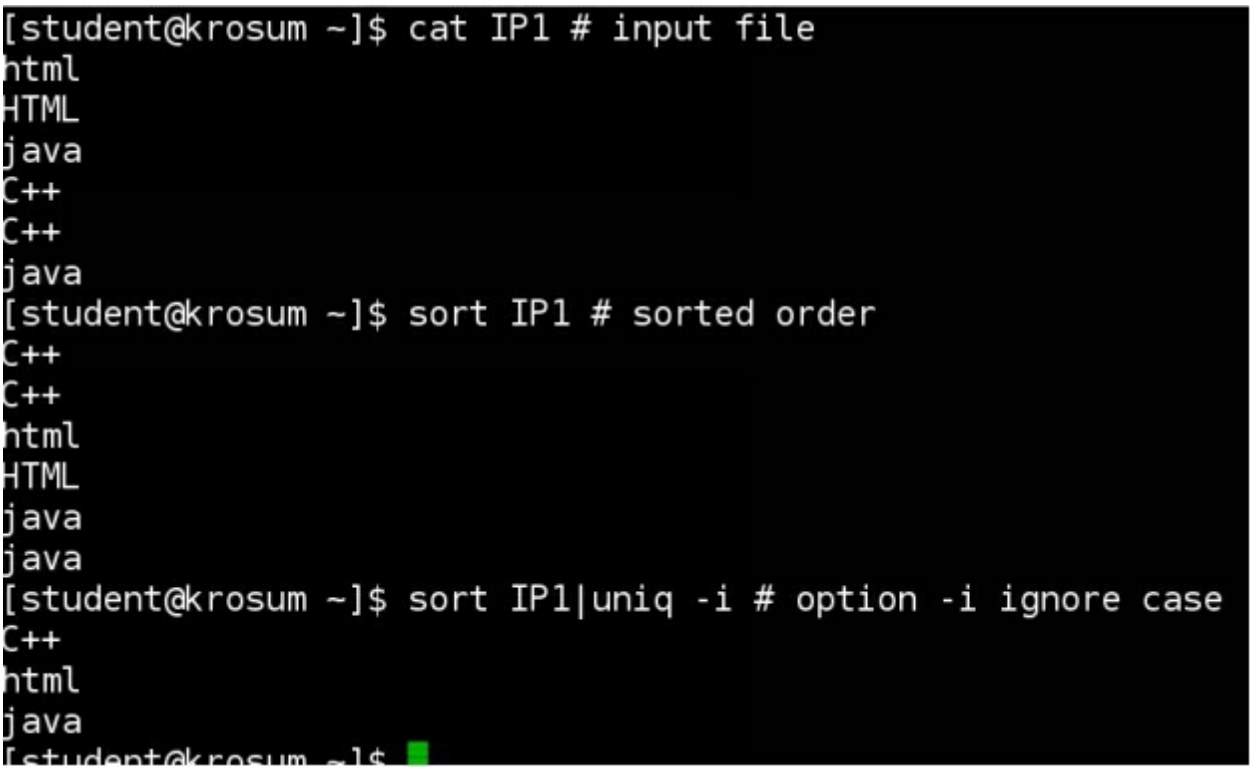
uniq–option inputfile

Remember all the filter commands outputs are runtime results that won't modify the input file.

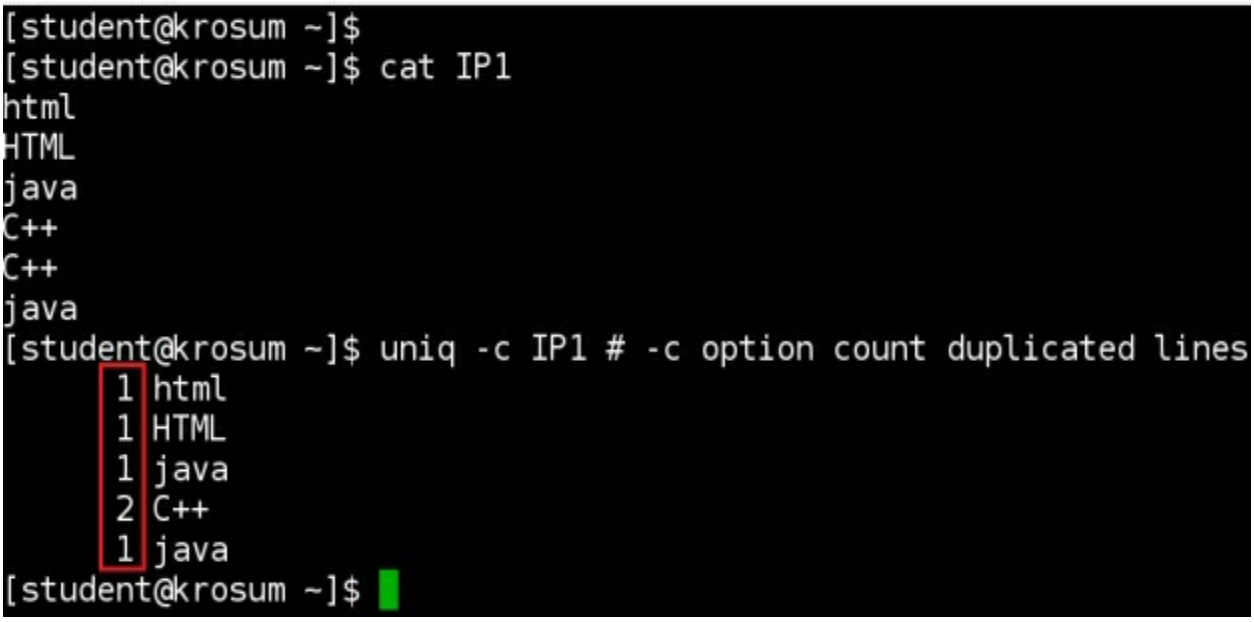


In case if the duplicates are in continuous format (one after the other), **uniq**command omits the duplicates whereas if the duplicates are not in continuous format, say adjacent format, the duplicates will still persist.

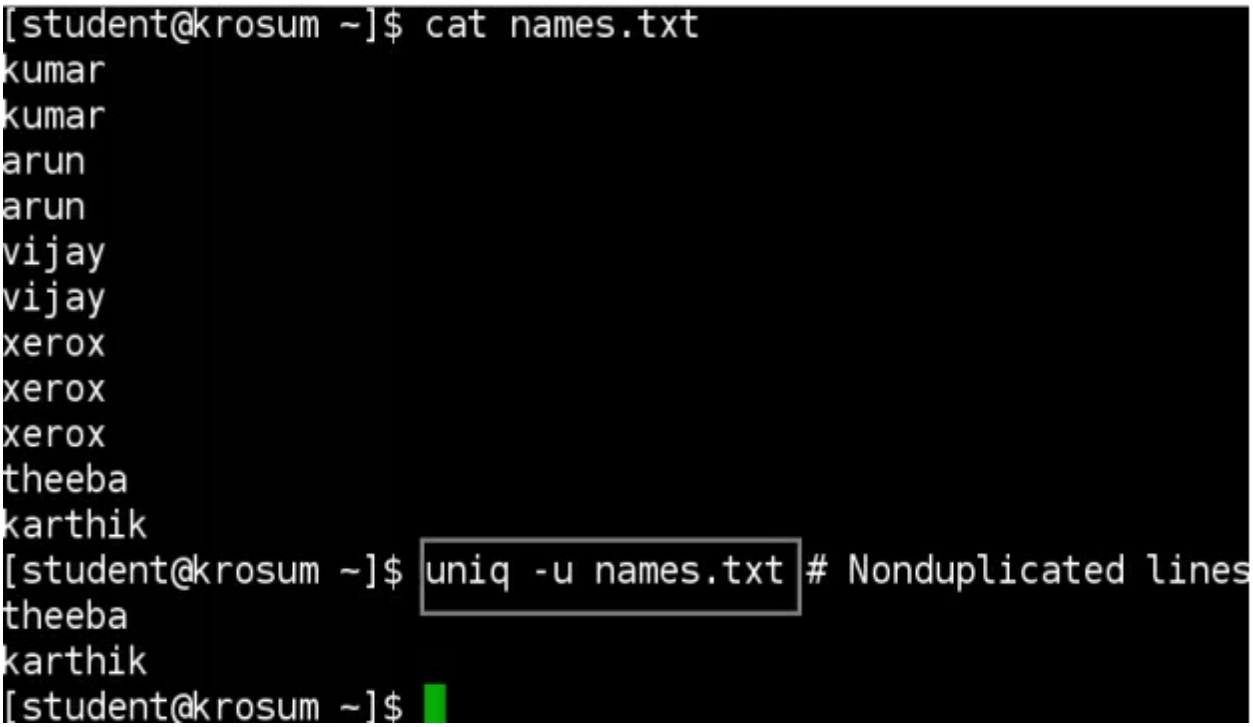
Duplicates of java are not in continuous format. So it is not omitted. In such case, before omitting the duplicate line, we can sort them first and then use the unique command.


[student@krosum ~]$ cat IP1 # input file
html
HTML
java
C++
C++
java
[student@krosum ~]$ sort IP1 # sorted order
C++
C++
html
HTML
java
java
[student@krosum ~]$ sort IP1|uniq -i # option -i ignore case
C++
html
java

Count duplicated lines (-c) option



Non duplicated lines (-u) option



The non-duplicated lines option is **–u**. It *suppresses the outputof the duplicated line* and lists only the unique lines in the file.

cmp command

cmp command in Linux/UNIX is used to *compare the two files byte by byte* and helps you to find out whether the two files are identical or not.

When cmp is used for comparison between two files, it reports the location of the first mismatch to the screen if difference is found and if no difference is found i.e if the files are identical, cmp displays no message and simply returns the prompt.

```
[student@krosum ~]$ cat -n F1.txt
     1  p1.c
     2  p2.c
     3  p3.c
     4  p4.c
[student@krosum ~]$ cat -n F3.txt
     1  p1.c
     2  p2.c
     3  p3.c
     4  p4.c
[student@krosum ~]$ # both file contents are same
[student@krosum ~]$
[student@krosum ~]$ cmp F1.txt F3.txt
[student@krosum ~]$
```

If file contents are not unique, it will get display message as below.

```
[student@krosum ~]$ cat -n F1.txt
     1  p1.c
     2  p2.c
     3  p3.c
     4  p4.c
[student@krosum ~]$ cat -n F2.txt
     1  p2.c
     2  p3.c
     3  p4.c
     4  p5.c
     5  p6.c
[student@krosum ~]$ cmp F1.txt F2.txt
F1.txt F2.txt differ: byte 2, line 1
```

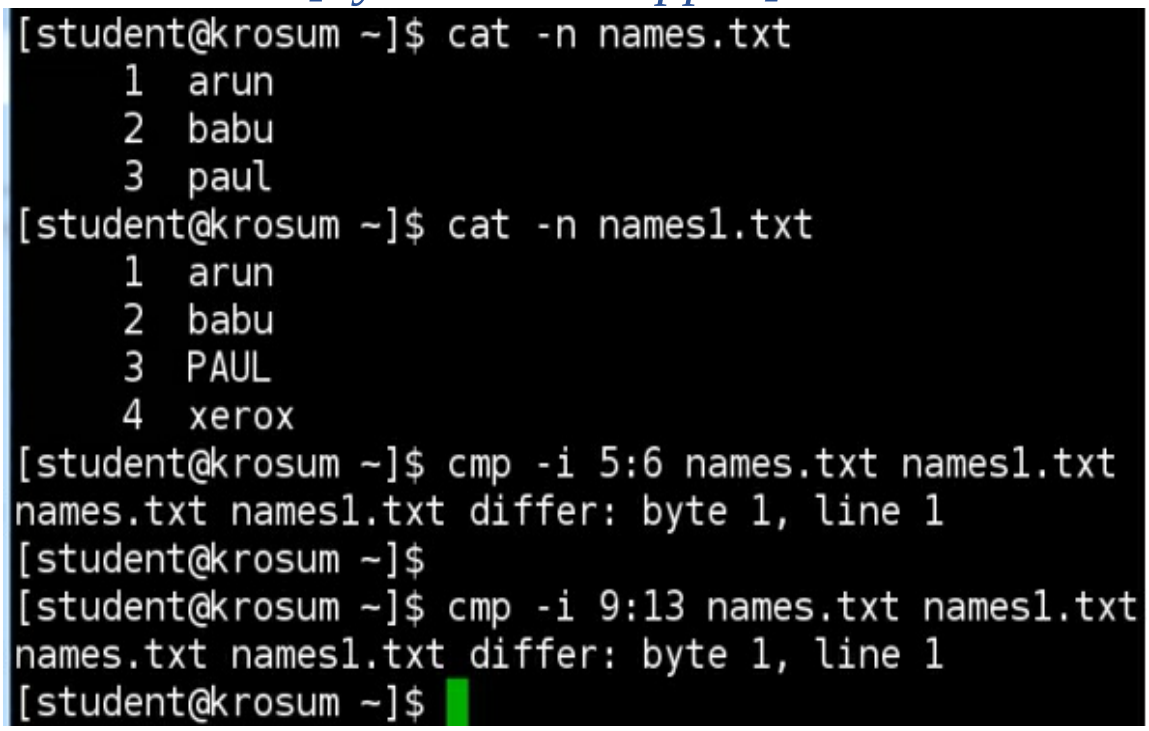
Options for cmp command

-b(print-bytes)

If we want cmp to display the differing bytes to the output, use **-b** option.

```
[student@krosum ~]$ cat -n names.txt
     1  arun
     2  babu
     3  paul
[student@krosum ~]$ cat -n names1.txt
     1  arun
     2  babu
     3  PAUL
     4  xerox
[student@krosum ~]$ cmp names.txt names1.txt
names.txt names1.txt differ: byte 11, line 3
[student@krosum ~]$
[student@krosum ~]$ cmp -b names.txt names1.txt
names.txt names1.txt differ: byte 11, line 3 is 160 p 120 P
[student@krosum ~]$
```

-i [bytes-to-be-skipped]



Now, this option when used with **cmp**command helps to skip a particular number of initial bytes from both the files and then after skipping it compares the files. This can be done by specifying the number of bytes as argument to the **-i**command line option.

Difference (diff) command

The *diff command shows the line-by-line difference between two files*. The first file is compared to the second file. The differences are identified so that the first file could be modified to make it match the second file.

Syntax

diff -optionfiles (or) directories

the diff command always works on files. The arguments can be two files, a file and a directory or two directories. When one file and one directory are specified, the utility looks for a file with the same name in the specified directory.

If the two directories are provided, all files with matching names in each directory are used. Each difference is displayed using the following format:

range1 action range2

< text from file1

>text from file2

The first line defines what should be done at range1 in file1 to make it match the lines at range2 in file2.

If the range spans multiple lines, there will be a text entry for each line in the specified range.

The action can be change (**c**) , append (**a**) , delete (**d**)

Change (**c**) indicates what action should be taken to make file1 the same as file2.

Append (**a**) indicates what lines need to be added to file1 to make it the same as file2. Appends can take place only at the end of file1; they occur only when file1 is shorter that file2.

Delete (**d**) indicates what lines must be deleted from file1 to make it the same as file2. Delete can occur only if file1 is longer than file2.

diff command report Interpretation	
Example	**Interpretation**
5c5	Change: Replace line 5 in file1 with line 5 in file2
10 a 16,17	Append: at the end of file1 (after line10), insert lines 16,17 from file2 Note that for append,there is no separator(dash) line and no file1(<) lines
28,29d30	Delete : The extra lines at the end of file1 should be deleted. The text of the lines to be deleted is shown. Note again that there is no separator line and in this case ,no file2 (>) lines.

```
[student@krosum ~]$ cat -n F1.txt
     1  p1.c
     2  p2.c
     3  p3.c
     4  p4.c
[student@krosum ~]$ cat -n F2.txt
     1  p2.c
     2  p3.c
     3  p4.c
     4  p5.c
     5  p6.c
[student@krosum ~]$ diff F1.txt F2.txt
1d0
< p1.c
4a4,5
> p5.c
> p6.c
[student@krosum ~]$
```

Examples :

```
[student@krosum ~]$ cp F1.txt F3.txt
[student@krosum ~]$ cat >>F3.txt
test1.c                 append operation
test2.c
[student@krosum ~]$ cat -n F3.txt
     1  p1.c
     2  p2.c
     3  p3.c
     4  p4.c
     5  test1.c
     6  test2.c
[student@krosum ~]$ diff F1.txt F3.txt
4a5,6
> test1.c          append
> test2.c
[student@krosum ~]$
```

Directory Differences

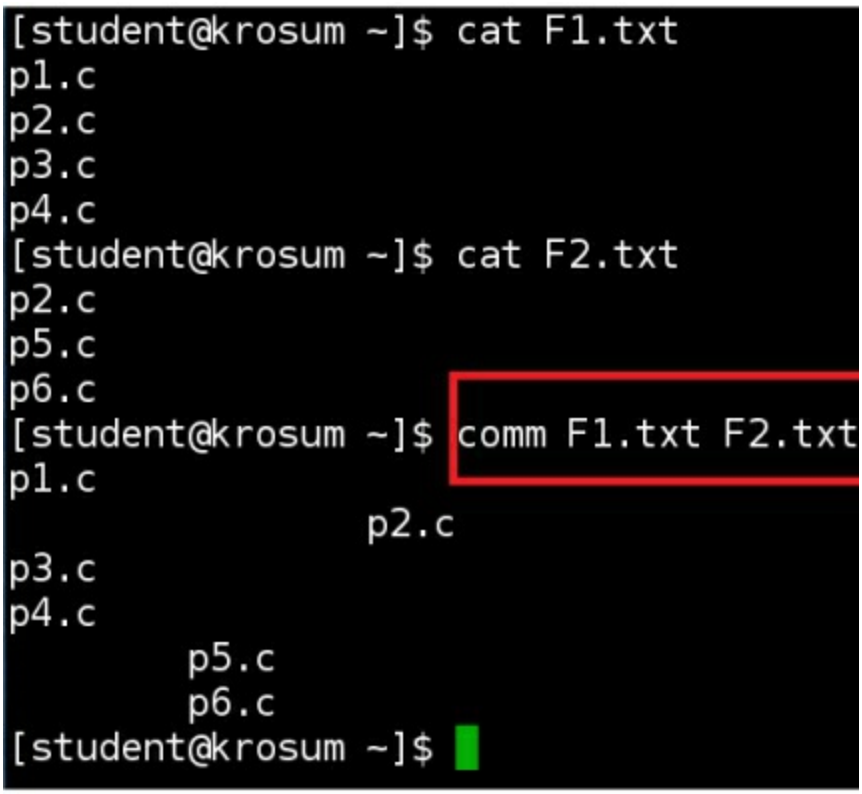
Common (comm) command

The **comm**command finds lines that are identical in two files. *It compares the files line by line and displays the results in three columns.* The left column contains unique lines in the file1 ; the center contains unique lines in file2 ; and the right column contains lines found in both files.

Note : Both input files (file1 and file2) must be sorted order.

Syntax

comm –optionfile1 file2



```
[student@krosum ~]$ comm F1.txt F2.txt
p1.c
                p2.c
p3.c
p4.c
        p5.c
        p6.c
[student@krosum ~]$ comm -23 F1.txt F2.txt  # ommit 2nd and 3rd column
p1.c
p3.c
p4.c
[student@krosum ~]$
```

Task : Filter 1st column only (unique elements from F1.txt file content)

Task : Filter 2nd column only (unique elements from F2.txt file content)

```
[student@krosum ~]$ comm F1.txt F2.txt
p1.c
                p2.c
p3.c
p4.c
        p5.c
        p6.c
[student@krosum ~]$ comm -13 F1.txt F2.txt # ommit 1st column and 3rd column
p5.c
p6.c
[student@krosum ~]$
```

```
[student@krosum ~]$ comm F1.txt F2.txt
p1.c
                p2.c
p3.c
p4.c
        p5.c
        p6.c
[student@krosum ~]$ comm -12 F1.txt F2.txt
p2.c
[student@krosum ~]$
```

Task : Filter 3rd column only (common content from F1.txt and F2.txt)

tr command

The **tr** (translate) command *replaces each character in a the first set of characters with a corresponding character in the second specified set.* Each set is specified as a string.

The first character in the first set is replaced by the first character in the second set, the second character in the first set is replaced by the second character in the second set,and so forth until all matching characters have been replaced.

Syntax :

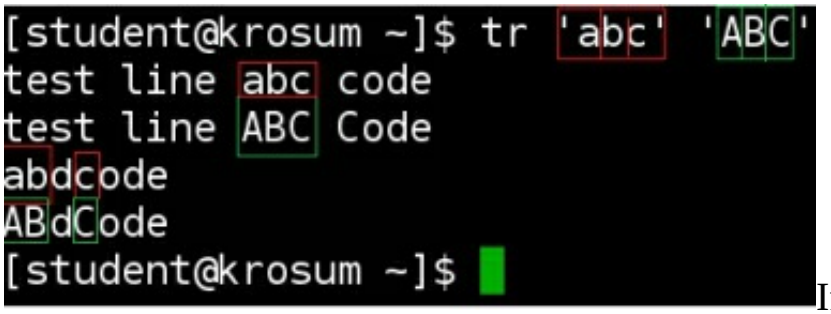
tr option oldstring newstring

-ddelete characters

-ssqueeze duplicates

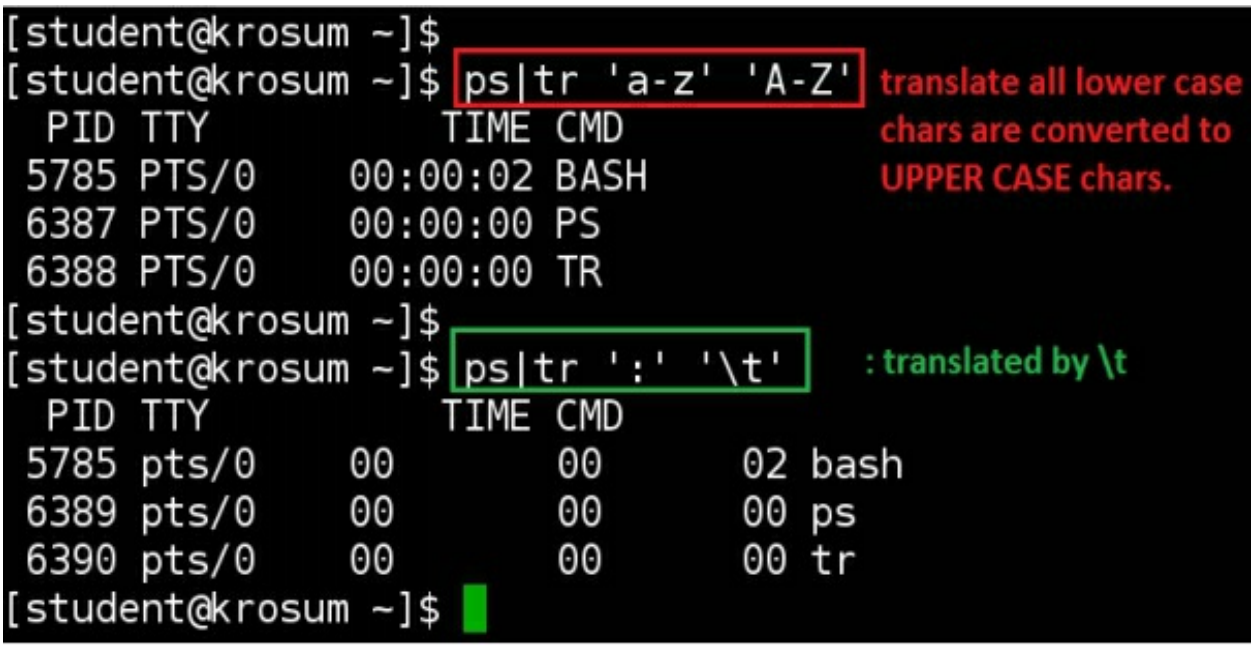
-ccomplement set

Translate receives its input from standard input and writes its output to standard output.



If no options are specified, the text is matched against the **oldstring** set, and any matching characters are replaced with the corresponding characters in the **newstring** set. Unmatched characters are unchanged.

range of characters are denonted by – symbol. For example a-z means match all lowercase characters.

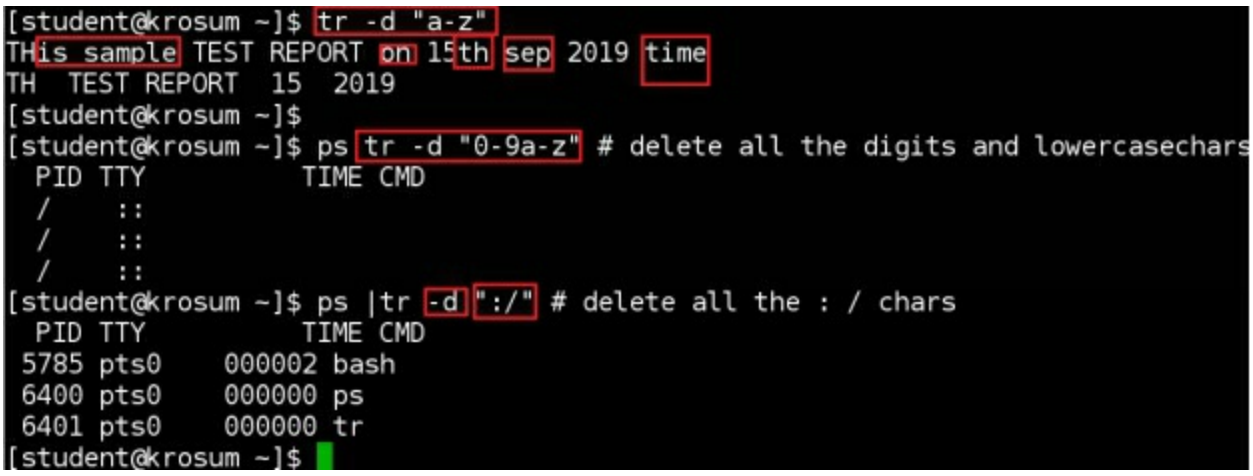


1-5 means match range of number 1 2 3 4 5. See the below example to convert all characters from lower case to uppercase. **tr** command reads the input string from pipe STDIN.

Delete characters

To *delete all matching characters in the translation*, we use the delete option (**-d**).

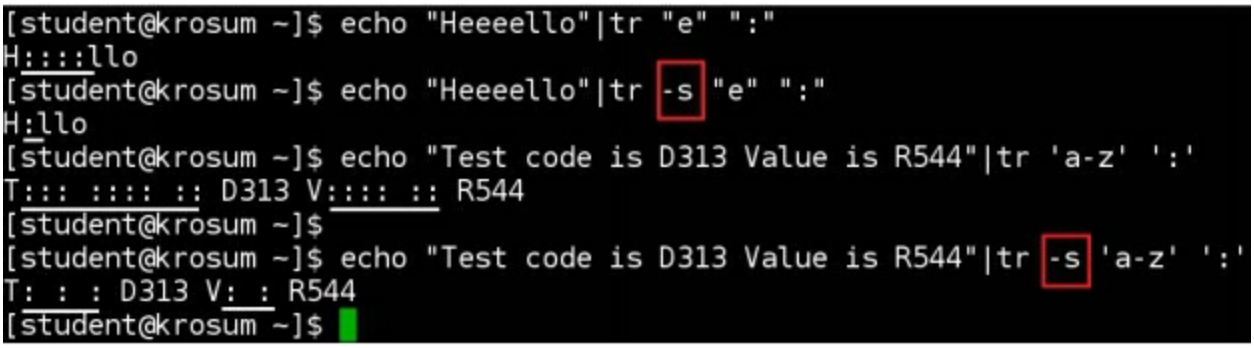
An example to delete all lowercase chars from the given string.



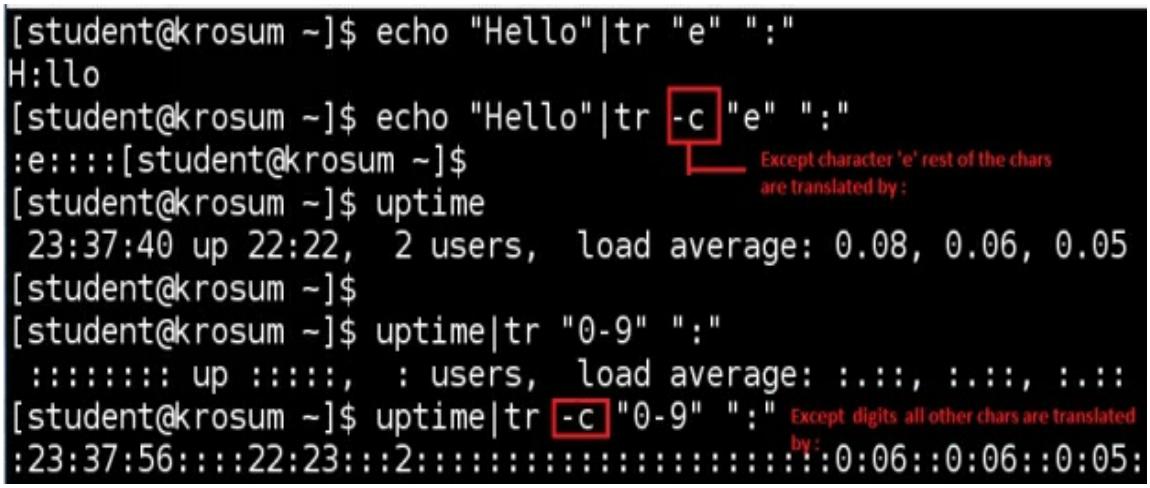
Squeeze output (-s)

The squeeze option deletes consecutive occurrences of the same character in the output.

For example , if after *the translation of 'e' to the letter ':', t*he output contains a string of ':' all but one would be deleted. See the below snap



Complement(-c)

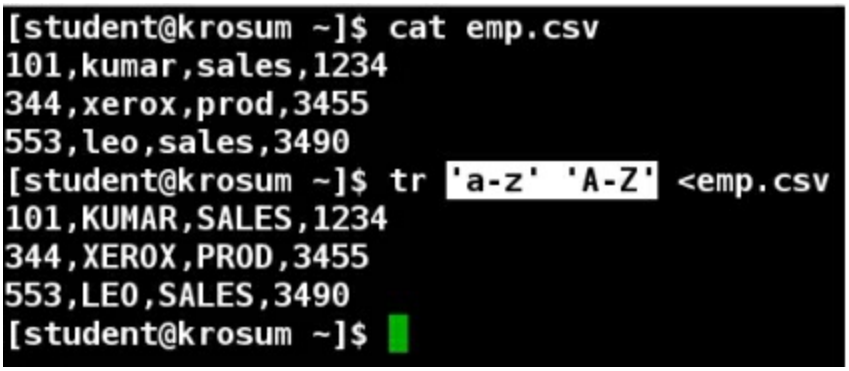


The complement option reverses the meaning of the first string. Rather than specifying what characters are to be changed, it says what characters are not to be changed.

Translate will not accept data from a file. To translate a file,therefore ,we must redirect the file into the translate command.

tr ‘a-z’ ‘A-Z’ < emp.csv

convert from lowercase characters to uppercase characters



Viewing Files

There are several commands that display information about a file in the read-only format.

• The file-viewing commands include the following:

- **cat**
- **more**
- **tail**
- **head**
- **wc**

we have already discussed how to use cat command. recap cat command syntax: ***cat -option inputfile.*** The cat command displays the content of one or more files.

Viewing Files: more Command

The more command displays the content of a text file one screen at a time.

Syntax:-

morefilename

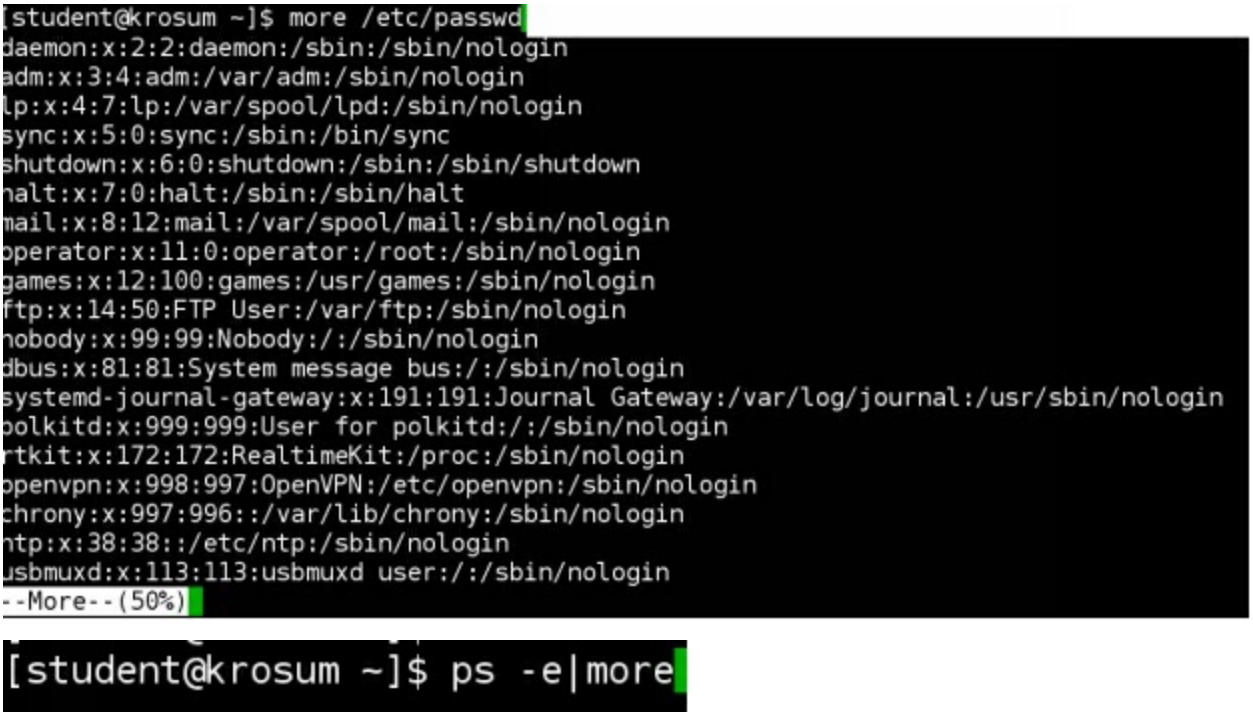
The *--More--(n%)* message appears at the bottom of each screen, where n% is the percentage of the file that has been displayed. When the entire file has been displayed, the shell prompt appears.

When the *--More--(n%)*prompt appears at the bottom of the screen, we can use the keys described in the table to scroll through the file.

Keyboard	Action
Spacebar	Moves forward one screen
Return	Scrolls one line at a time
B	Moves back one screen
H	Displays a help menu of features

/string	**Searches forward for pattern**
N	**Finds the next occurrence of pattern**
Q	**Quits and returns to the shell prompt**

Examples



Press **‘q’** to quit.

Viewing Files: head Command

The head command by default, displays the first 10 lines of a file.

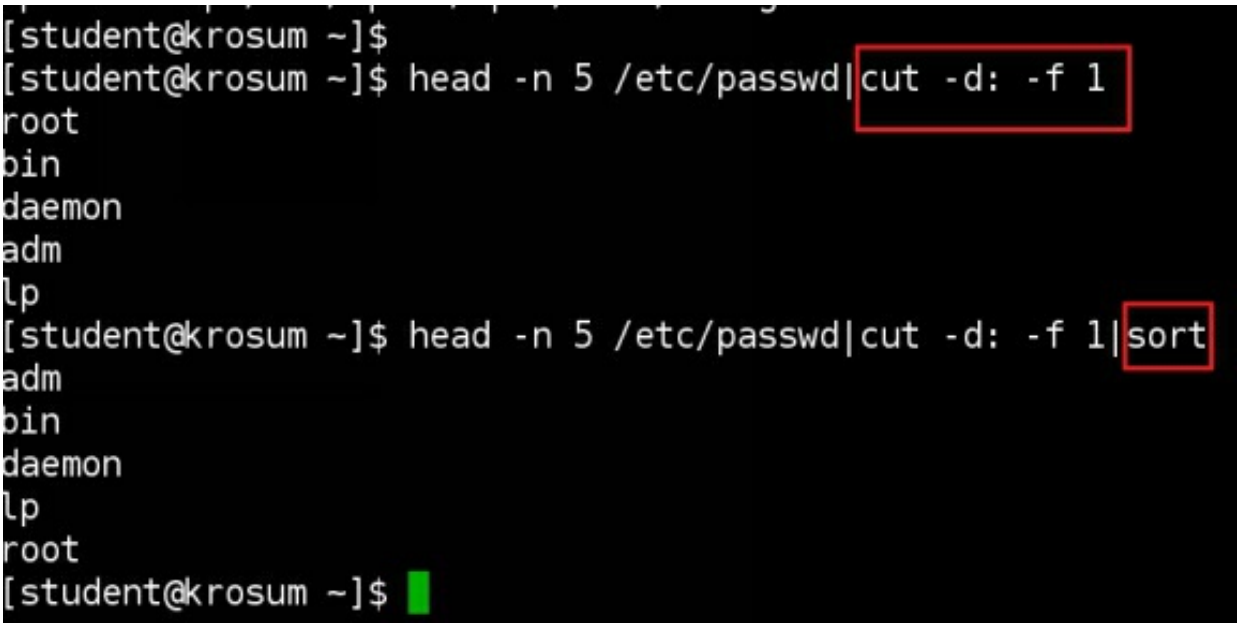
Syntax

head -n filename

We can change the number of lines displayed by using the **-n** option.

For example, to display the first five lines of the **/etc/passwd** file, enter the head command with the **-n** option set to 5.

```
[student@krosum ~]$
[student@krosum ~]$ head -n 5 /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
[student@krosum ~]$
```



Task: *How to filter first 5 line contents from /etc/passwd file and from that filter login name (1st field) and display all the login names in sorted order ?*

Viewing Files: tail Command

The tail command by default, displays the last 10 lines of a file

Syntax

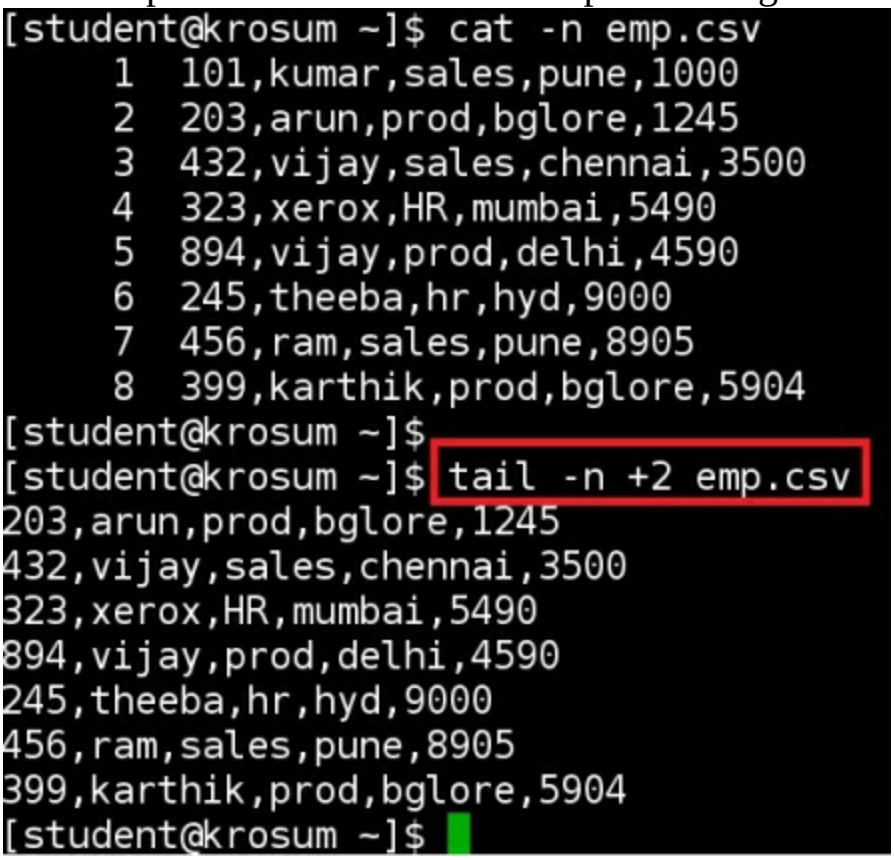
tail –n/+n filename

We can change the number of lines displayed by using the -n or +n options.

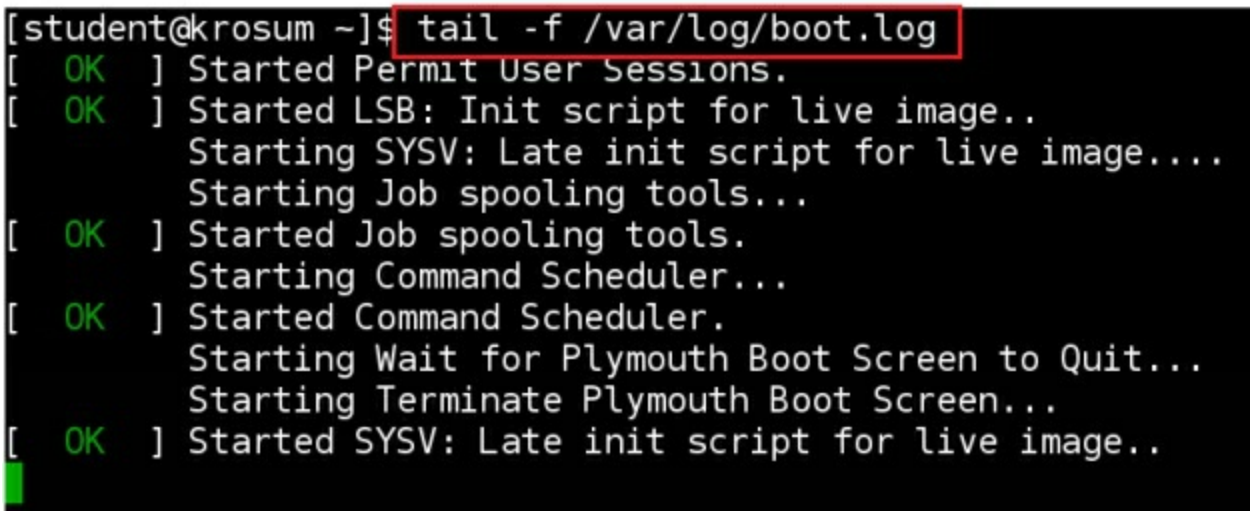
- The **-n** option displays n lines from the end of the file.
- The **+n** option displays the file from line n to the end of the file.

tail–n +2 emp.csv

this example shows all lines of the report starting from the second line.



There is a special command line option **–f**. Instead of just displaying the last few lines and exiting, *tail command with –foption*, displays the lines and then monitors the file.



We can use either **tail–f**or **tailf**command – this command will monitor log content dynamically.

To interrupt tail while it is monitoring, break-in with **Ctrl+C**.

wc command

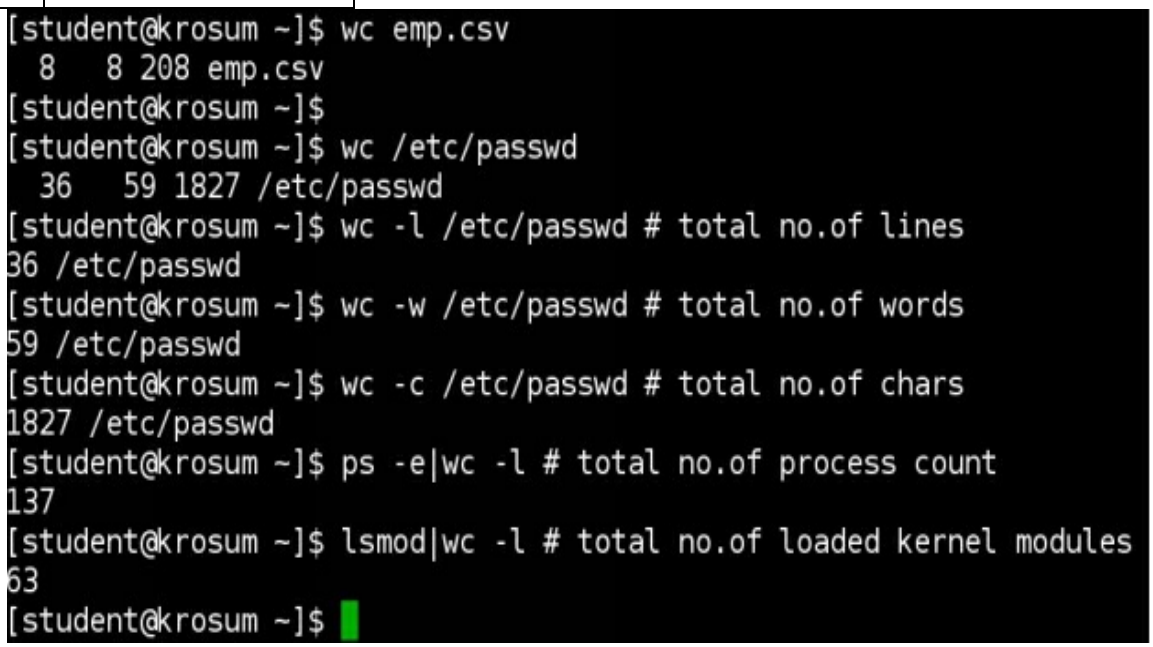
The **wc** command *displays the number of lines, words, and characters* contained in a file.

Syntax:

wc -options filename

When we use the wc command without options, the output displays the number of lines, words, and characters contained in the file.

Symbol	PathName
-l	**Line count**
-w	**Word count**
-c	**Byte count**
-m	**Character count**



For example, to display the number of lines, words, and characters in the emp.csv file, use the wc command.

grep command

- grep stands for **global regular expression print**.
  - grep is used to *search the input file for all lines that match a specified regular expression* and write them to the standard output file (monitor).

Syntax

grep–option pattern inputfile(s)

grep performs the following operations:

- step 1 : greputility *read the input data*, line by line from file (or) pipe into pattern space. The pattern space is a buffer that can hold only one text line.
- step 2: *search the pattern* to the pattern space
- step3: If there is a match, *the line is copied from the pattern space to the standard output.*

The **grep** utilities repeat these three operations on each line in the input.

```
[student@krosum ~]$ cat -n emp.csv
     1  101,kumar,sales,pune,1000
     2  203,arun,prod,bglore,1245
     3  432,vijay,sales,chennai,3500
     4  323,xerox,HR,mumbai,5490
     5  894,vijay,prod,delhi,4590
     6  245,theeba,hr,hyd,9000
     7  456,ram,sales,pune,8905
     8  399,karthik,prod,bglore,5904
[student@krosum ~]$
[student@krosum ~]$ # grep pattern inputfile
[student@krosum ~]$ grep sales emp.csv
101,kumar,sales,pune,1000
432,vijay,sales,chennai,3500
456,ram,sales,pune,8905
[student@krosum ~]$
[student@krosum ~]$ grep admin emp.csv
[student@krosum ~]$
```

Examples :

See the above examples. Here initially we are searching keyword *sales*from *emp.csv* file, recap the above 3 steps on how grep is working.

- Step 1 :grepwill read input (emp.csv) data line by line , placed to pattern space
- Step 2: search the **sales** keyword on the pattern space
- Step 3: if sales keyword is matched, display the matched pattern lines to monitor (STDOUT).

If pattern is not matched, grep does not return any result to monitor. In the above example **admin** keyword is not matched in emp.csv file.

As we walk through the flow, look for how **grep**handles the following situations:

1. grep is a search utility; it can **search only for the existence** of a line that matches a pattern.

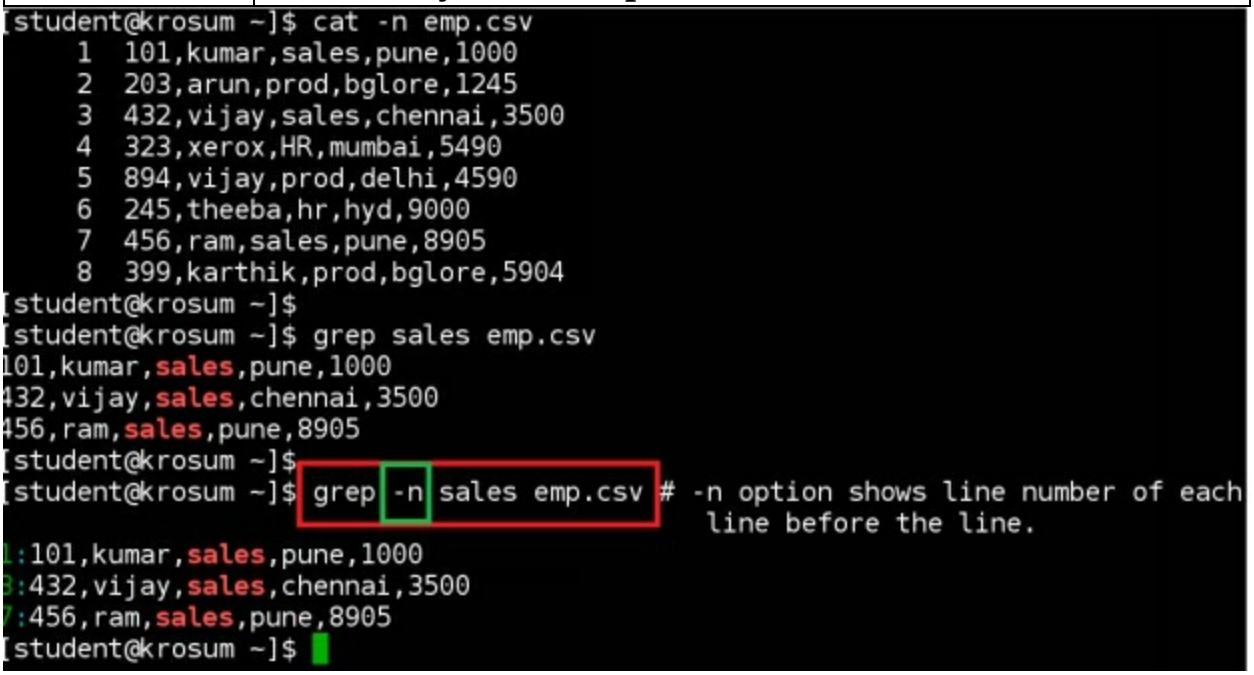
2. The only action that grepcan perform on a line is to send it to standard output. If the line does not match the pattern, it is not printed.
3. The line selection is based only on the pattern.

- grepis a filter , it is used to search keyword (pattern) from input file (or) pipe.
- grepcannot be used to add, delete or change a line.

grep options and explanation

grep–optionpattern inputfile

Option	Explanation
-c	Prints only a count of the number of lines matching the pattern
-i	Ignores upper/lower case in the matching text
-l	Prints a list of files that contains at least one line matching the pattern
-n	Shows line number of each line before the line
-q	Silent mode
-v	Inverse output. Prints lines that do not match pattern
-w	Word based search
-x	Prints only lines that entirely match pattern.
-f file	List of strings to be matched are in file.
-o	Prints only matched pattern.



Examples

```
[student@krosum ~]$ cat -n emp.csv
     1  101,kumar,sales,pune,1000
     2  203,arun,prod,bglore,1245
     3  432,vijay,sales,chennai,3500
     4  323,xerox,HR,mumbai,5490
     5  894,vijay,prod,delhi,4590
     6  245,theeba,hr,hyd,9000
     7  456,ram,sales,pune,8905
     8  399,karthik,prod,bglore,5904
[student@krosum ~]$ grep -n HR emp.csv
4:323,xerox,HR,mumbai,5490
[student@krosum ~]$
[student@krosum ~]$ grep -n hr emp.csv
6:245,theeba,hr,hyd,9000
[student@krosum ~]$ grep -i HR emp.csv # -i ignore case
323,xerox,HR,mumbai,5490
245,theeba,hr,hyd,9000
[student@krosum ~]$ grep -in HR emp.csv # we can combine multiple options
4:323,xerox,HR,mumbai,5490
6:245,theeba,hr,hyd,9000
[student@krosum ~]$
```

```
[student@krosum ~]$ cat -n emp.csv
     1  101,kumar,sales,pune,1000
     2  203,arun,prod,bglore,1245
     3  432,vijay,sales,chennai,3500
     4  323,xerox,HR,mumbai,5490
     5  894,vijay,prod,delhi,4590
     6  245,theeba,hr,hyd,9000
     7  456,ram,sales,pune,8905
     8  399,karthik,prod,bglore,5904
[student@krosum ~]$ grep sales emp.csv |wc -l
3
[student@krosum ~]$ grep -c sales emp.csv # -c count
3
[student@krosum ~]$ grep -ic HR emp.csv  # -i (ignorecase) and count
2
[student@krosum ~]$ ps -e|grep -c bash # count total no.of bash process
1
[student@krosum ~]$
```

```
     1  101,kumar,sales,pune,1000
     2  203,arun,prod,bglore,1245
     3  432,vijay,sales,chennai,3500
     4  323,xerox,HR,mumbai,5490
     5  894,vijay,prod,delhi,4590
     6  245,theeba,hr,hyd,9000
     7  456,ram,sales,pune,8905
     8  399,karthik,prod,bglore,5904
[student@krosum ~]$
[student@krosum ~]$ grep -v sales emp.csv  # -v option inverse output
203,arun,prod,bglore,1245
323,xerox,HR,mumbai,5490
894,vijay,prod,delhi,4590
245,theeba,hr,hyd,9000
399,karthik,prod,bglore,5904
```

```
[student@krosum ~]$ # searching single pattern from more than one file(s)
[student@krosum ~]$ #      pattern
[student@krosum ~]$ grep sales emp.csv F1.txt F2.txt p1.txt
emp.csv:101,kumar,sales,pune,1000
emp.csv:432,vijay,sales,chennai,3500
emp.csv:456,ram,sales,pune,8905
p1.txt:list of sales emp detials
[student@krosum ~]$
[student@krosum ~]$ grep -l sales emp.csv F1.txt F2.txt p1.txt # -l print a list
                                                               of files
emp.csv
p1.txt
[student@krosum ~]$
```

more than one input files

```
[student@krosum ~]$
[student@krosum ~]$ ps -e|wc -l # total no.of process count
139
[student@krosum ~]$ ps -e|grep -c bash # total no.of bash process count
3
[student@krosum ~]$ ps -e|grep -cv bash # except bash process count
136
[student@krosum ~]$
```

```
[student@krosum ~]$ # searching more than one pattern
[student@krosum ~]$ # ==============================
[student@krosum ~]$
[student@krosum ~]$ grep -e sales -e prod emp.csv
101,kumar,sales,pune,1000
203,arun,prod,bglore,1245
432,vijay,sales,chennai,3500
894,vijay,prod,delhi,4590
456,ram,sales,pune,8905
399,karthik,prod,bglore,5904
[student@krosum ~]$ grep -nie sales -e prod emp.csv
1:101,kumar,sales,pune,1000
2:203,arun,prod,bglore,1245
3:432,vijay,sales,chennai,3500
5:894,vijay,prod,delhi,4590
7:456,ram,sales,pune,8905
8:399,karthik,prod,bglore,5904
[student@krosum ~]$
```

combine multiple options

We can search more than one pattern in the following ways .

Syntax:

grep*-*e *“pattern1”* ***-e*** *“pattern2”* ***inputfile***

egrep*“pattern1|pattern2|patternN***” ***inputfile***

```
[student@krosum ~]$ grep -e sales -e prod emp.csv
101,kumar,sales,pune,1000
203,arun,prod,bglore,1245
432,vijay,sales,chennai,3500
894,vijay,prod,delhi,4590
456,ram,sales,pune,8905
399,karthik,prod,bglore,5904
[student@krosum ~]$
[student@krosum ~]$ egrep "sales|prod" emp.csv
101,kumar,sales,pune,1000
203,arun,prod,bglore,1245
432,vijay,sales,chennai,3500
894,vijay,prod,delhi,4590
456,ram,sales,pune,8905
399,karthik,prod,bglore,5904
```

grep-E"Pattern1 \| Pattern2 \|Pattern3 .. \|PatternN" input_file

```
[student@krosum ~]$ grep -E "sales|prod" emp.csv
101,kumar,sales,pune,1000
203,arun,prod,bglore,1245
432,vijay,sales,chennai,3500
894,vijay,prod,delhi,4590
456,ram,sales,pune,8905
399,karthik,prod,bglore,5904
```

```
[student@krosum ~]$ ps -e|grep -e "bash" -e "python"
 6780 pts/0    00:00:02 bash
 7314 pts/1    00:00:00 bash
 7335 pts/2    00:00:00 bash
 7397 pts/2    00:00:00 python
[student@krosum ~]$
[student@krosum ~]$ ps -e|egrep "bash|python"
 6780 pts/0    00:00:02 bash
 7314 pts/1    00:00:00 bash
 7335 pts/2    00:00:00 bash
 7397 pts/2    00:00:00 python
[student@krosum ~]$
[student@krosum ~]$ ps -e|grep -E "bash|python"
 6780 pts/0    00:00:02 bash
 7314 pts/1    00:00:00 bash
 7335 pts/2    00:00:00 bash
 7397 pts/2    00:00:00 python
[student@krosum ~]$
```

```
[student@krosum ~]$ cat ptr.txt
sales
prod
pune
[student@krosum ~]$ grep -f ptr.txt emp.csv # -f attaching external pattern file
101,kumar,sales,pune,1000
203,arun,prod,bglore,1245
432,vijay,sales,chennai,3500
894,vijay,prod,delhi,4590
456,ram,sales,pune,8905
399,karthik,prod,bglore,5904
[student@krosum ~]$
```

```
[student@krosum ~]$ grep sales emp.csv
101,kumar,sales,pune,1000
432,vijay,sales,chennai,3500
456,ram,sales,pune,8905
[student@krosum ~]$
[student@krosum ~]$ grep -o sales emp.csv
sales
sales
sales
[student@krosum ~]$ ps -e|grep bash
 6780 pts/0    00:00:02 bash
 7314 pts/1    00:00:00 bash
 7335 pts/2    00:00:00 bash
[student@krosum ~]$ ps -e|grep -o bash
bash
bash
bash
[student@krosum ~]$
```

```
[student@krosum ~]$ cat process
bash
python
netns
systemd
[student@krosum ~]$ ps -e|grep -f process
    1 ?        00:00:06 systemd
   14 ?        00:00:00 netns
  158 ?        00:00:13 systemd-journal
  160 ?        00:00:01 systemd-udevd
  289 ?        00:00:00 systemd-logind
 6780 pts/0    00:00:02 bash
 7314 pts/1    00:00:00 bash
 7335 pts/2    00:00:00 bash
 7397 pts/2    00:00:00 python
[student@krosum ~]$
```

search patterns are writtern to external file

```
[student@krosum ~]$ cat -n IP
     1  this code number 2345
     2  sample test
     3  is a relationship
     4  line number is3455
     5  total no.of line code 1455
[student@krosum ~]$ grep -n is IP
1:this code number 2345
3:is a relationship
4:line number is3455
[student@krosum ~]$ grep -nw is IP
3:is a relationship
[student@krosum ~]$ ps -e|grep cron
29157 ?        00:00:01 crond
[student@krosum ~]$
[student@krosum ~]$ ps -e|grep -w cron
[student@krosum ~]$
```

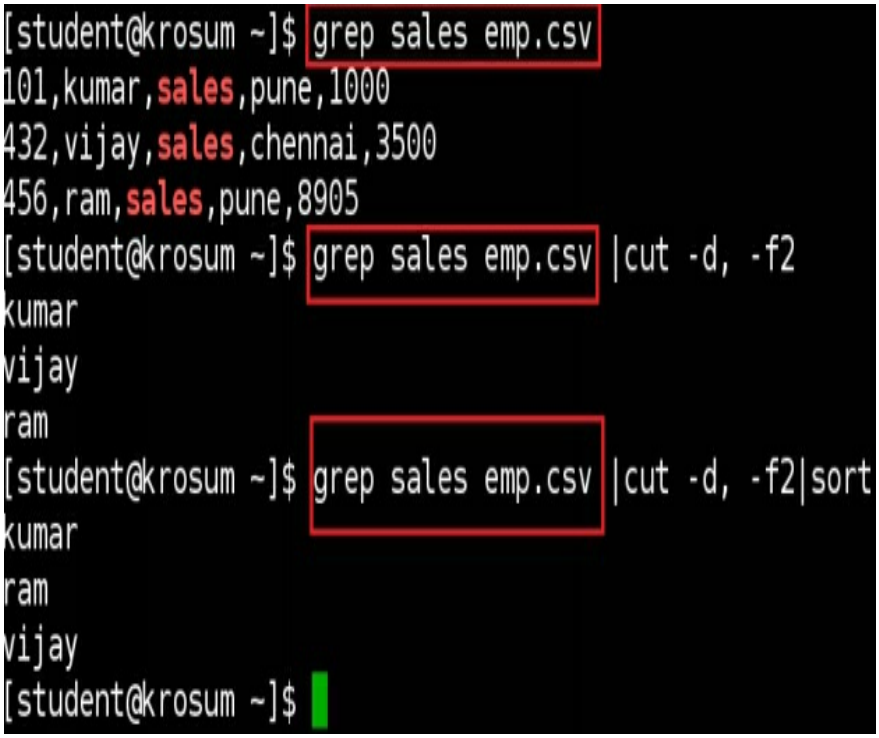
```
[student@krosum ~]$ cat -n IP1
     1  sales details
     2  sales
     3  vijay,sales
     4  paul,sales,pune
[student@krosum ~]$ grep -n sales IP1
1:sales details
2:sales
3:vijay,sales
4:paul,sales,pune
[student@krosum ~]$ grep -x sales IP1
sales
[student@krosum ~]$ grep -nx sales IP1
2:sales
[student@krosum ~]$
```

-x option prints only lines that entirely match pattern**.**

```
[student@krosum ~]$ # filter mounted xfs type file system
[student@krosum ~]$ # ------ ============     ------------
[student@krosum ~]$ df -Th|grep xfs
/dev/sda2      xfs       9.8G  2.6G  7.2G  27% /
/dev/sda5      xfs       2.0G   33M  2.0G   2% /home
[student@krosum ~]$
```

```
[student@krosum ~]$ # search list of loaded bluetooth module
[student@krosum ~]$ # ------ ===============---------
[student@krosum ~]$
[student@krosum ~]$ lsmod|grep bluetooth
bluetooth             312602  22 bnep,btusb,rfcomm
rfkill                 20721  3 bluetooth
[student@krosum ~]$
[student@krosum ~]$ lsmod|grep -n bluetooth
43:bluetooth             312602  22 bnep,btusb,rfcomm
45:rfkill                 20721  3 bluetooth
[student@krosum ~]$
```

```
[student@krosum ~]$ free -m
             total       used       free     shared    buffers     cached
Mem:          1003        981         21          0          0        720
-/+ buffers/cache:        260        743
Swap:         7965          0       7965
[student@krosum ~]$ free -m |grep -i swap # filter swapspace usage
Swap:         7965          0       7965
[student@krosum ~]$
```



grep Family

- Fast **grep**(**fgrep**: supports only string patterns – no regular expression)
- **grep**: supports only a limited number of regular expression
- Extend grep(**egrep**: supports most regular expressions)

```
[student@krosum ~]$ grep -n sales emp.csv # grep family
1:101,kumar,sales,pune,1000
3:432,vijay,sales,chennai,3500
7:456,ram,sales,pune,8905
[student@krosum ~]$
[student@krosum ~]$ grep -ne "sales" emp.csv # egrep family
1:101,kumar,sales,pune,1000
3:432,vijay,sales,chennai,3500
7:456,ram,sales,pune,8905
[student@krosum ~]$
[student@krosum ~]$ fgrep -n "sales" emp.csv # fgrep family
1:101,kumar,sales,pune,1000
3:432,vijay,sales,chennai,3500
7:456,ram,sales,pune,8905
[student@krosum ~]$
```

- We will have a detailed look on regular expressions in upcoming topics.

Find command

- Find command used to *search and locate* list of files and directories.

Syntax:

find<searching from path>-name search file

```
[student@krosum ~]$
[student@krosum ~]$ find -name emp.csv
./Demo/L1/L2/emp.csv
./Demo/emp.csv
./emp.csv
./D1/emp.csv
[student@krosum ~]$
```

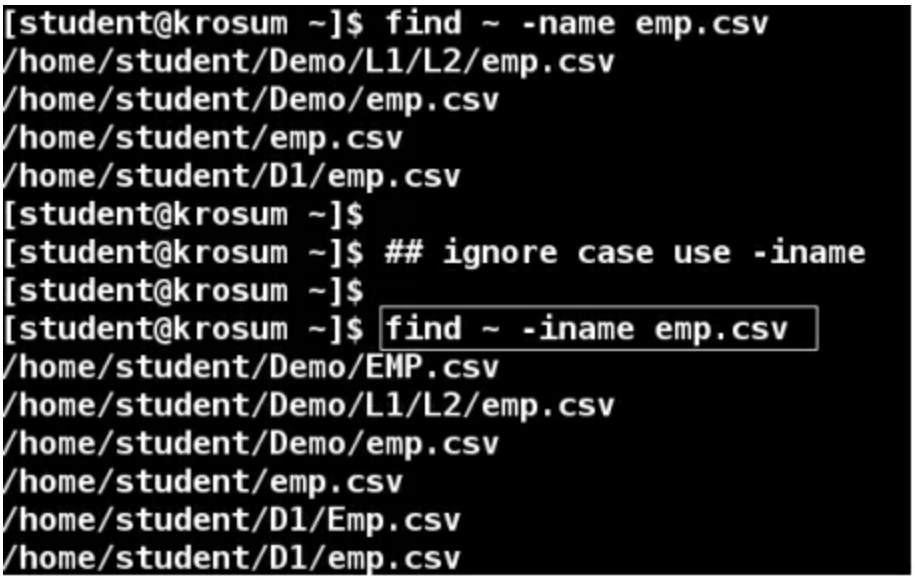
Task : Find all ***the files whose name is emp.csv*** in a current working directory.

- *find command searches the input files recursively*

```
[student@krosum ~]$ find ~ -name t1.txt
/home/student/Demo/t1.txt
/home/student/t1.txt
/home/student/D1/t1.txt
[student@krosum ~]$
```

Task: find all ***the files whose name is t1.txt in a login*** directory.

Task: Find Files Using Name and ***Ignoring Case***


[student@krosum ~]$ find ~ -name emp.csv
/home/student/Demo/L1/L2/emp.csv
/home/student/Demo/emp.csv
/home/student/emp.csv
/home/student/D1/emp.csv
[student@krosum ~]$
[student@krosum ~]$ ## ignore case use -iname
[student@krosum ~]$
[student@krosum ~]$ find ~ -iname emp.csv
/home/student/Demo/EMP.csv
/home/student/Demo/L1/L2/emp.csv
/home/student/Demo/emp.csv
/home/student/emp.csv
/home/student/D1/Emp.csv
/home/student/D1/emp.csv

Search a file with pattern

Filter list of log files

```
[student@krosum ~]$ find -name "*.log"
./.local/share/gvfs-metadata/home-d8db22ac.log
./r1.log
[student@krosum ~]$
```

Task : Filter ***list of log file under /var*** directory

```
[root@krosum ~]# find /var -name "*.log"
/var/log/lightdm/lightdm.log
/var/log/lightdm/x-0-greeter.log
/var/log/lightdm/x-0.log
/var/log/dracut.log
/var/log/Xorg.9.log
/var/log/prelink/prelink.log
/var/log/pm-powersave.log
/var/log/Xorg.0.log
/var/log/anaconda/anaconda.storage.log
/var/log/anaconda/anaconda.log
/var/log/anaconda/ks-script-jWdTaS.log
/var/log/anaconda/anaconda.ifcfg.log
/var/log/anaconda/anaconda.packaging.log
/var/log/anaconda/anaconda.program.log
/var/log/yum.log
/var/log/boot.log
/var/log/spice-vdagent.log
/var/log/audit/audit.log
[root@krosum ~]#
```

Task: Find ***list of regular files*** in a current directory.

```
[student@krosum ~]$
[student@krosum ~]$ find -type f
```

Task : Find ***list of directory files*** in a current directory.

```
[student@krosum ~]$ find -type d
./Videos
./.liferea_1.8
./.liferea_1.8/cache
./.liferea_1.8/cache/favicons
./.liferea_1.8/cache/plugins
./.liferea_1.8/cache/feeds
./.liferea_1.8/cache/scripts
./.gnupg
./.gnupg/private-keys-v1.d
./Music
./.local
./.local/share
./.local/share/gvfs-metadata
./.local/share/webkit
./.local/share/webkit/icondatabase
./.local/share/webkit/databases
./Demo
```

Task : Find ***list of character type device files in a /dev*** directory

```
[student@krosum ~]$
[student@krosum ~]$ find /dev -type c
/dev/tty6
/dev/tty5
/dev/tty4
/dev/tty3
/dev/tty2
/dev/tty1
/dev/vcsa1
/dev/vcs1
/dev/vcsa
/dev/vcs
/dev/tty0
/dev/console
/dev/tty
/dev/oldmem
/dev/kmsg
/dev/urandom
/dev/random
/dev/full
/dev/zero
```

Task : Find ***list of block type device files*** in a /dev directory

```
[student@krosum ~]$
[student@krosum ~]$ find /dev -type b  # List of block type device files
/dev/sda5
/dev/sda4
/dev/sda3
/dev/sda2
/dev/sda1
/dev/sda
/dev/sr0
/dev/loop7
/dev/loop6
/dev/loop5
/dev/loop4
/dev/loop3
/dev/loop2
/dev/loop1
/dev/loop0
```

Task : Find ***Files Based on their Permissions***

Find all the files *whose permissions are user can do read/write/execute*

```
[student@krosum ~]$
[student@krosum ~]$ find -perm -u=rwx
.
./Pictures
./.config
./.config/Thunar
./.config/ibus
./.config/ibus/bus
./.config/xfce4
./.config/xfce4/terminal
./.config/xfce4/xfwm4
./.config/xfce4/desktop
./.config/xfce4/xfconf
./.config/xfce4/xfconf/xfce-perchannel-xml
./.config/midori
./.config/yumex
```

mindepth and maxdepth

- using **mindepth** and **maxdepth** we ***can limit the search*** to a specific directory.
- maxdepth levels : Descend at most levels

Examples required **–maxdepth1**

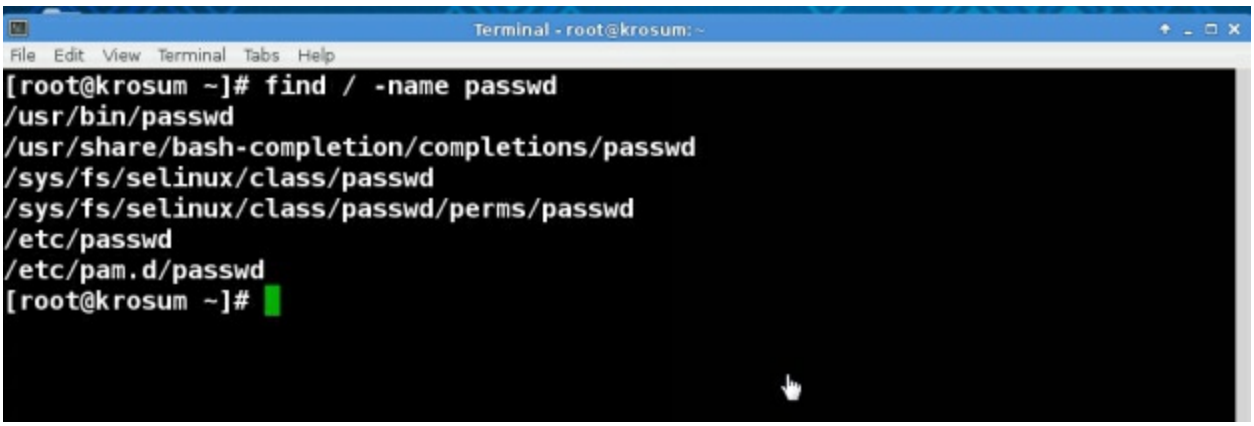
- -maxdepth0 means only apply the tests and actions to the starting-points themselves.

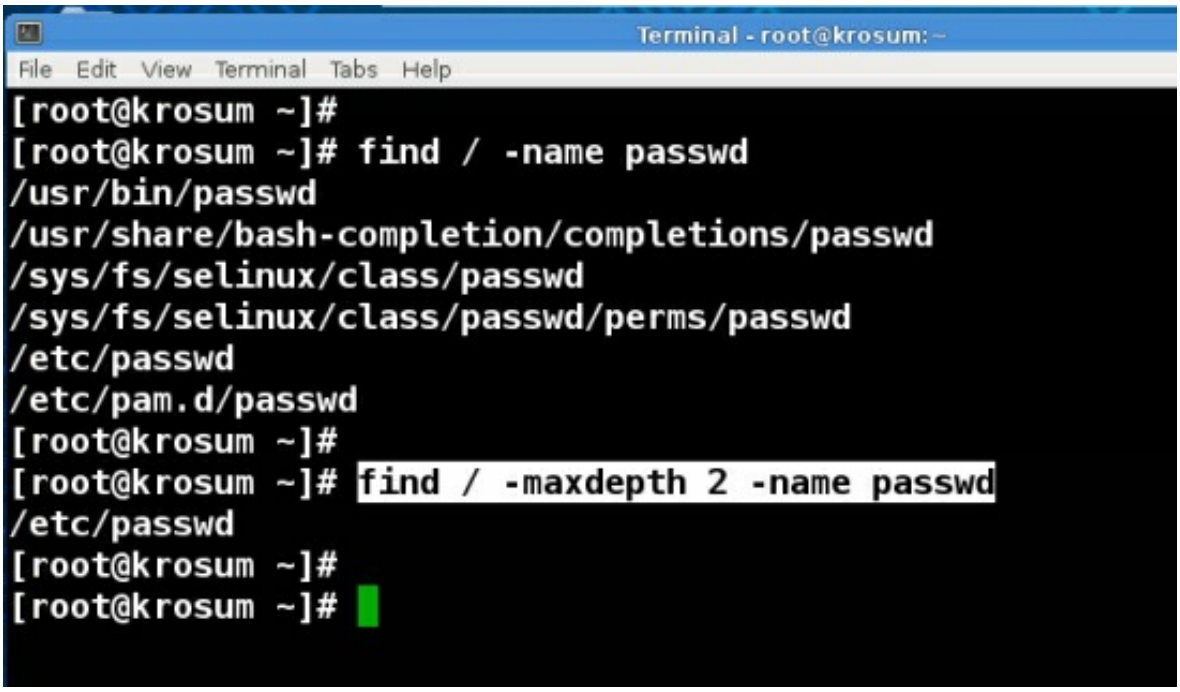
```
[root@krosum ~]# find / -name passwd
/usr/bin/passwd
/usr/share/bash-completion/completions/passwd
/sys/fs/selinux/class/passwd
/sys/fs/selinux/class/passwd/perms/passwd
/etc/passwd
/etc/pam.d/passwd
[root@krosum ~]# find / -maxdepth 2 -name passwd
/etc/passwd
[root@krosum ~]# find / -maxdepth 3 -name passwd
/usr/bin/passwd
/etc/passwd
/etc/pam.d/passwd
```

- -mindepthlevels : Do not apply any tests or actions at levels less than mindepth levels
- -mindepth**1** means process all files except the starting-points.

```
[student@krosum ~]$ find ~ -name "emp.csv"
/home/student/Demo/L1/L2/emp.csv
/home/student/Demo/emp.csv
/home/student/emp.csv
/home/student/D1/emp.csv
[student@krosum ~]$
[student@krosum ~]$ find ~ -maxdepth 1 -name "emp.csv"
/home/student/emp.csv
[student@krosum ~]$
[student@krosum ~]$ find ~ -maxdepth 2 -name "emp.csv"
/home/student/Demo/emp.csv
/home/student/emp.csv
/home/student/D1/emp.csv
```

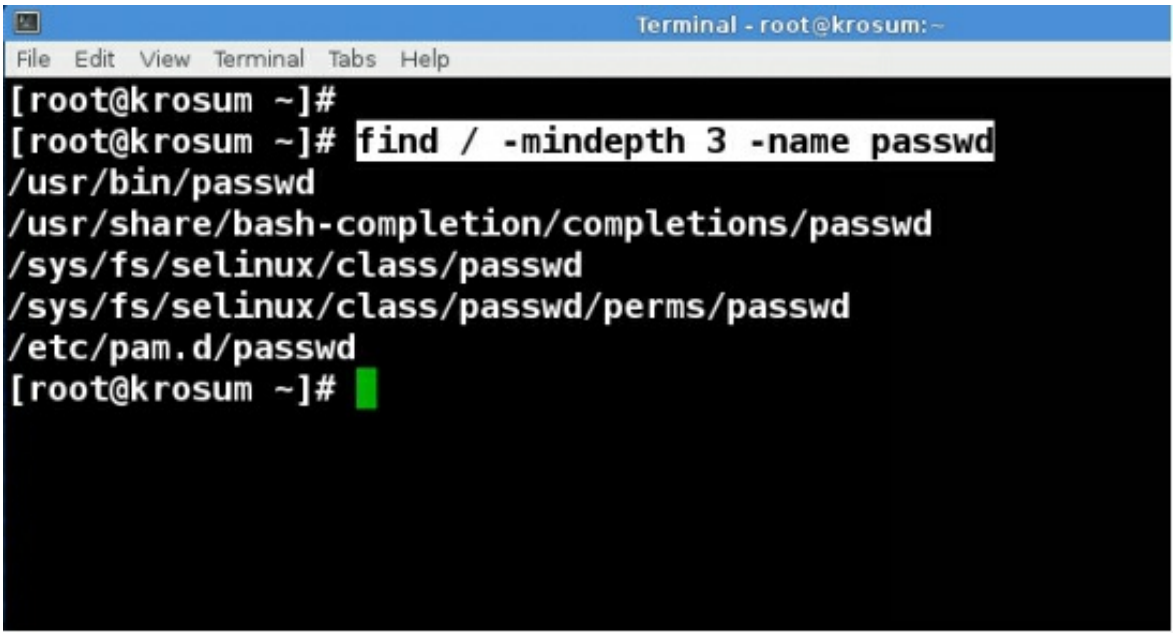
Task : Find ***the passwd file under all sub-directories*** starting from root directory.





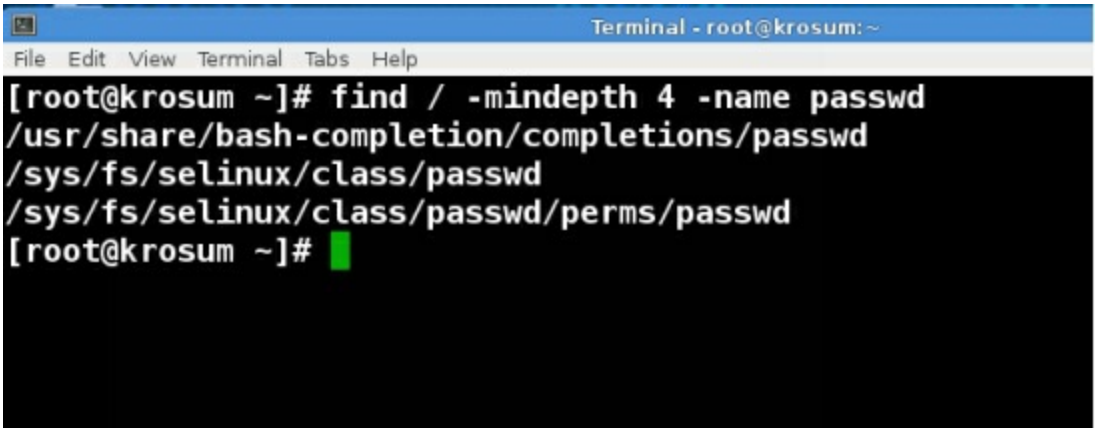
Task : Find ***the passwd file under / directory*** and ***one level down***

(i.e root - level 1, and one sub-directory - level 2)



Task : Find ***the passwd file under / directory***

(search from level 3)



Task : Find ***the passwd file under /directory(***search from level 4)

Find Files and Directories Based on Date and Time

As units we can use:

b– for 512-byte blocks

c- for bytes

w- for two-byte words

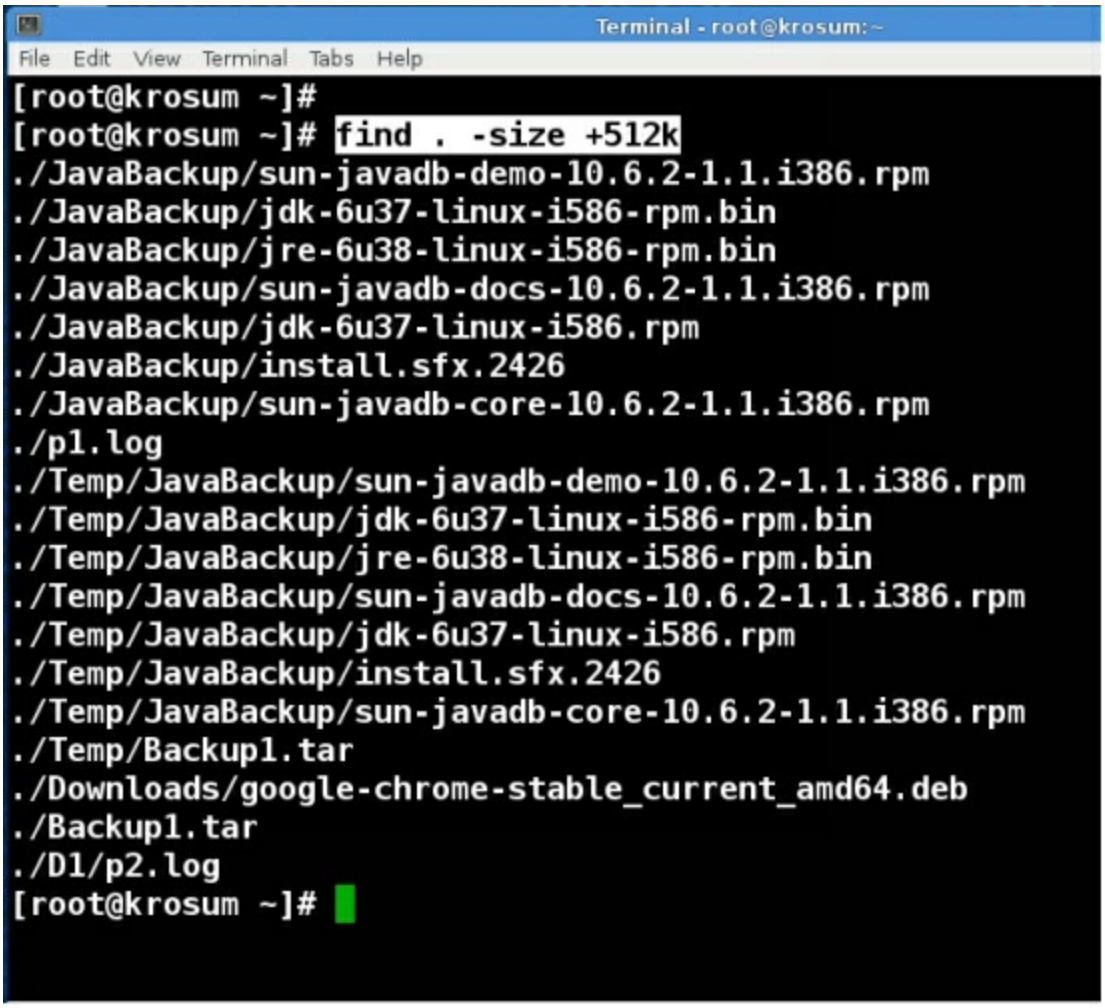
k- for kilobytes (units of 1024 bytes)

M- for Megabytes (units of 1048576 bytes)

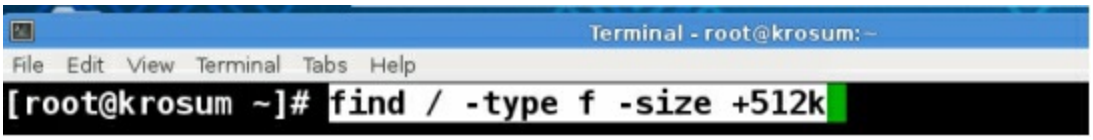
G- for Gigabytes (units of 1073741824 bytes)

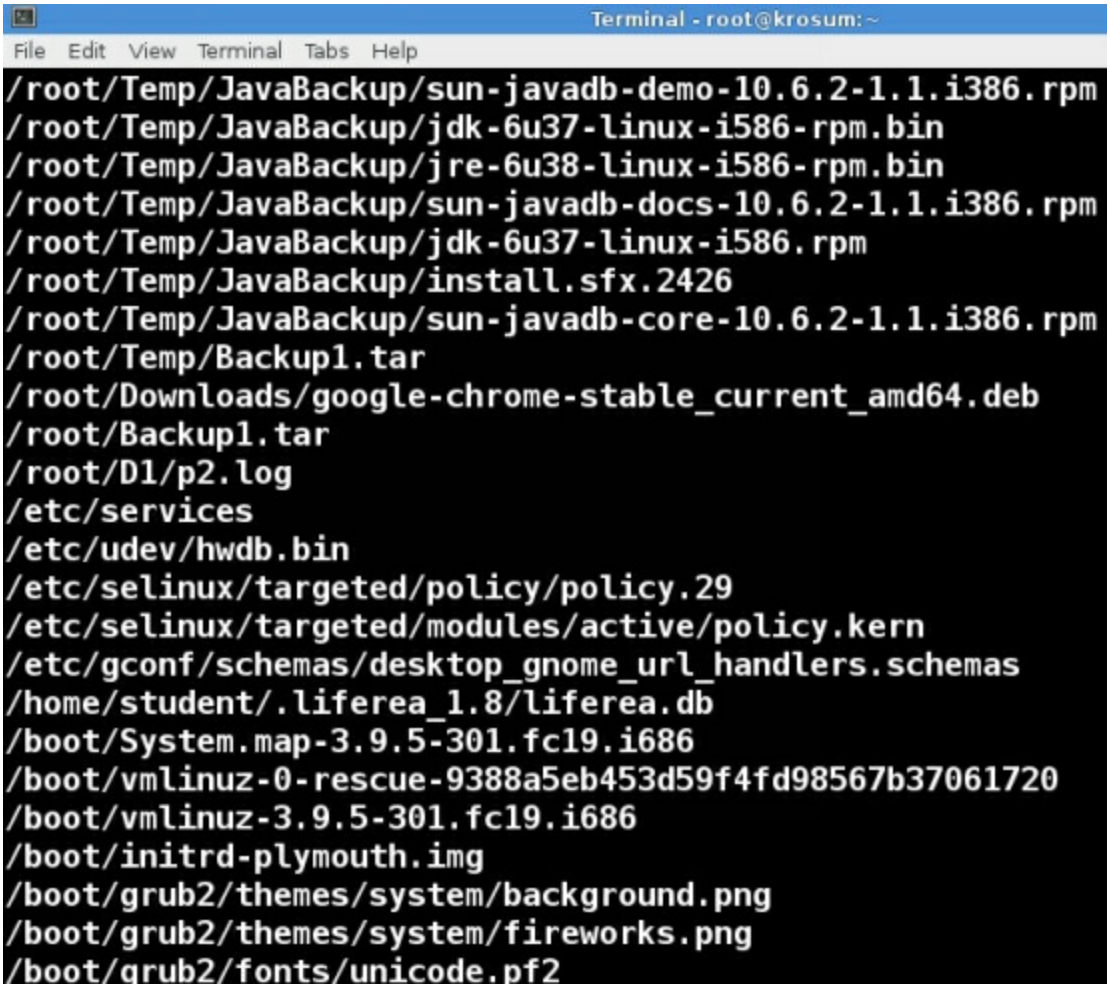
We can search for exact file size or just for bigger(+) or smaller (-) files

Task : Find ***files all bigger than 512k*** files

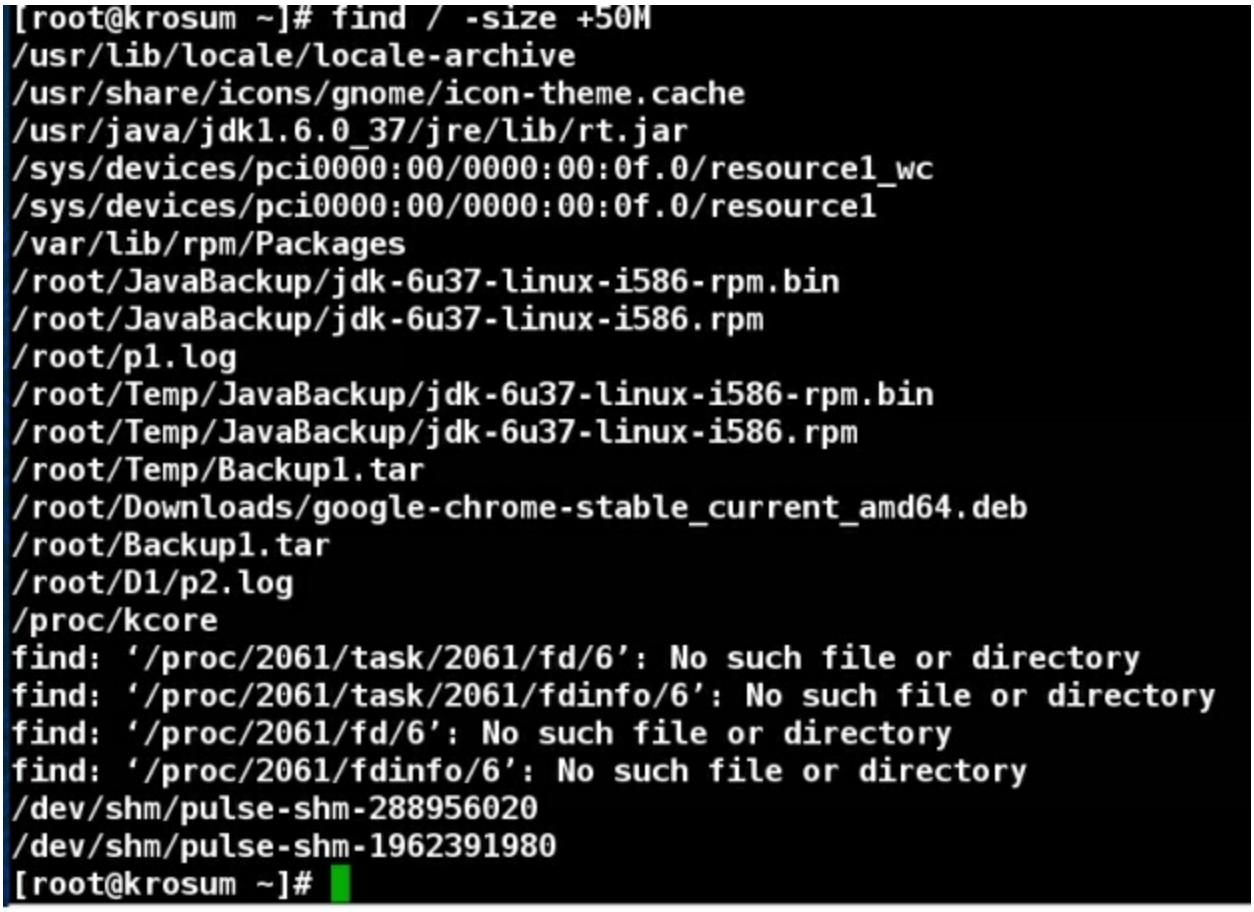


Task: search only **regularfiles** only

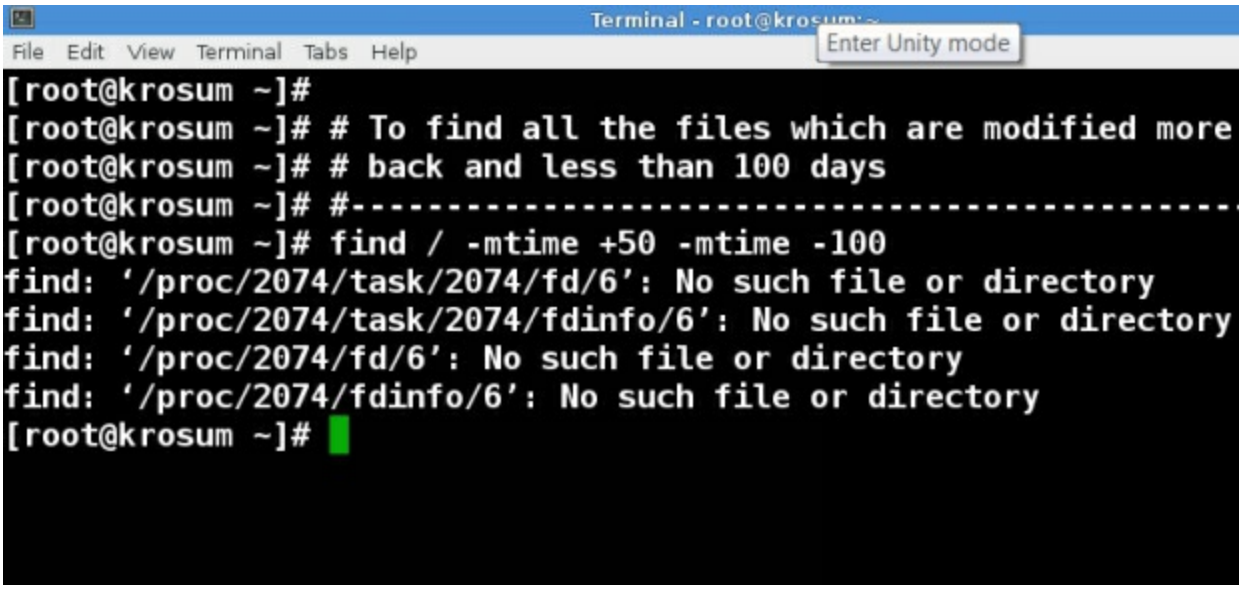


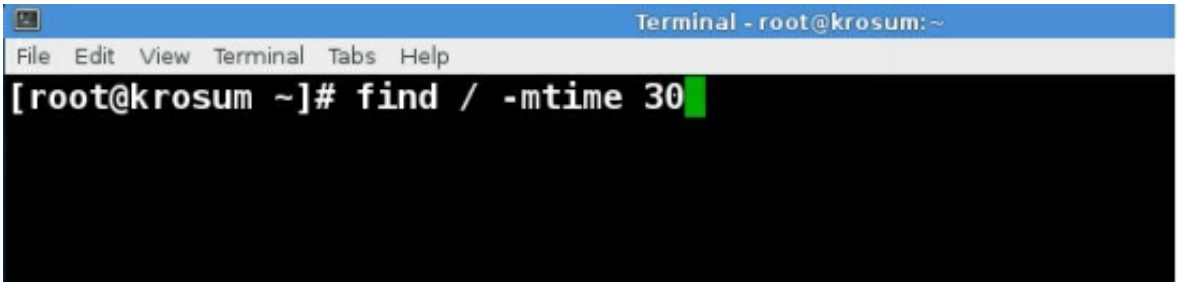


Task : To ***find all 50MB*** files.

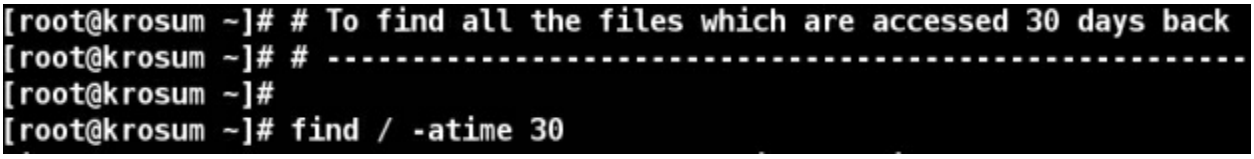


Task : To ***find all the files which are greater than 50MB*** and***less than 100MB***.



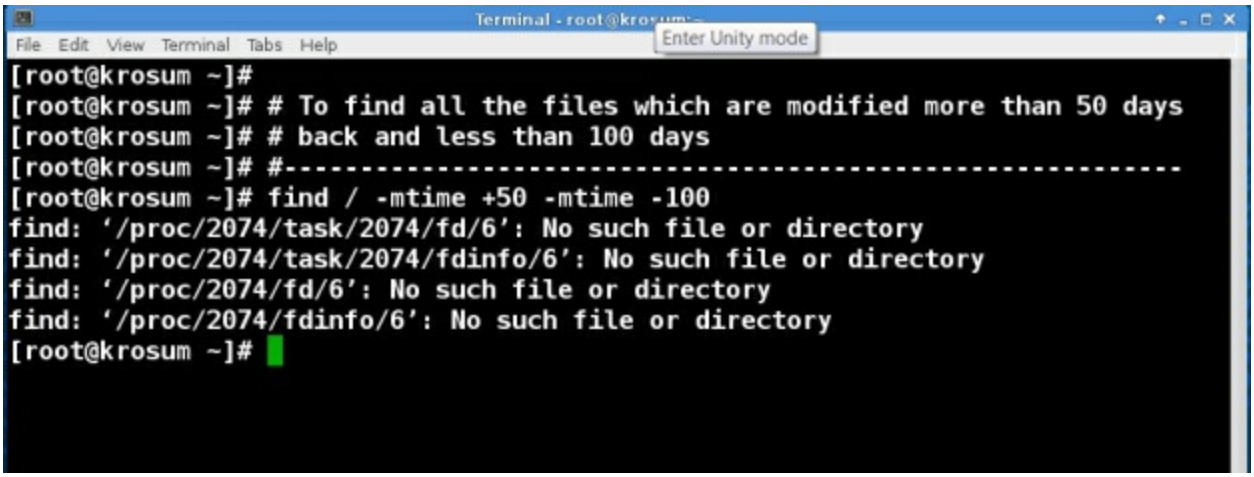


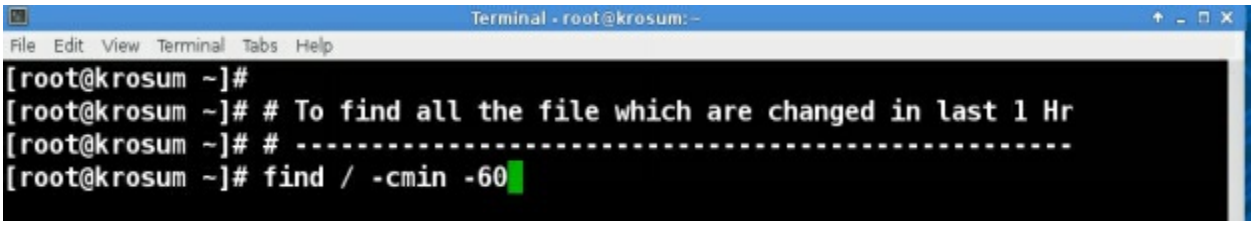
Task : To ***find all the files which are modified 30 days*** back.



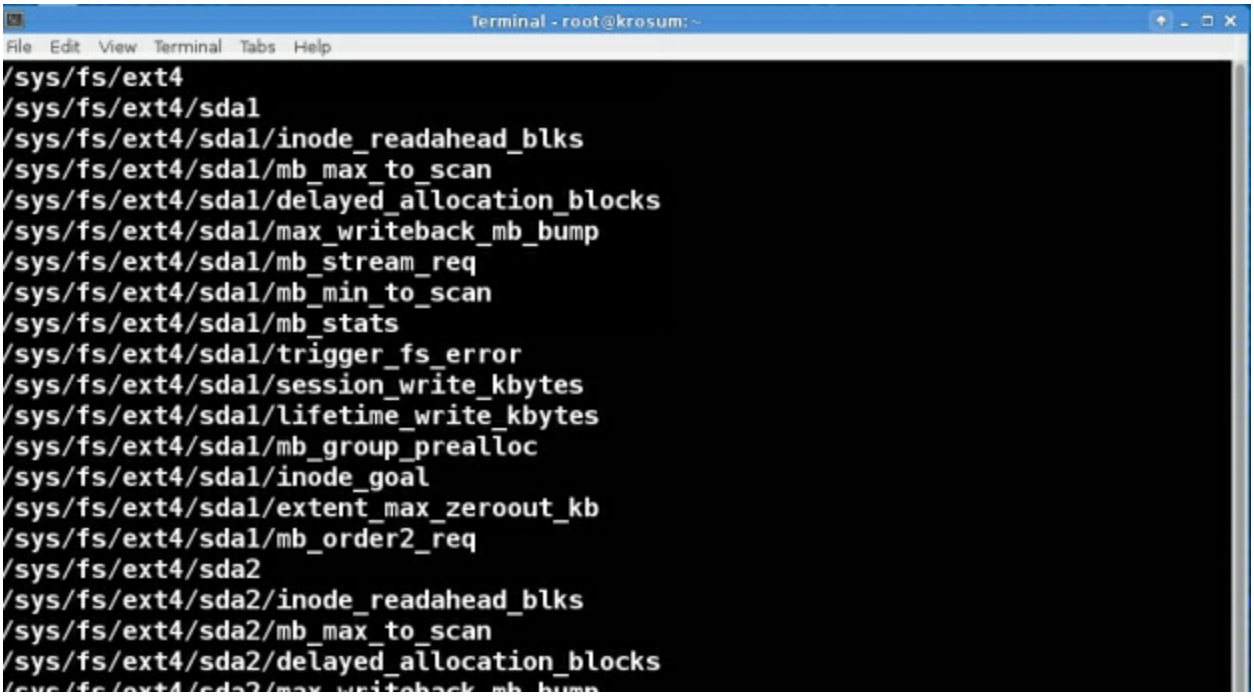
Task : To ***find all the files which are accessed 30 days*** back.

Task : To ***find all the files which are modified more than 50 days back and less than 100 days***.



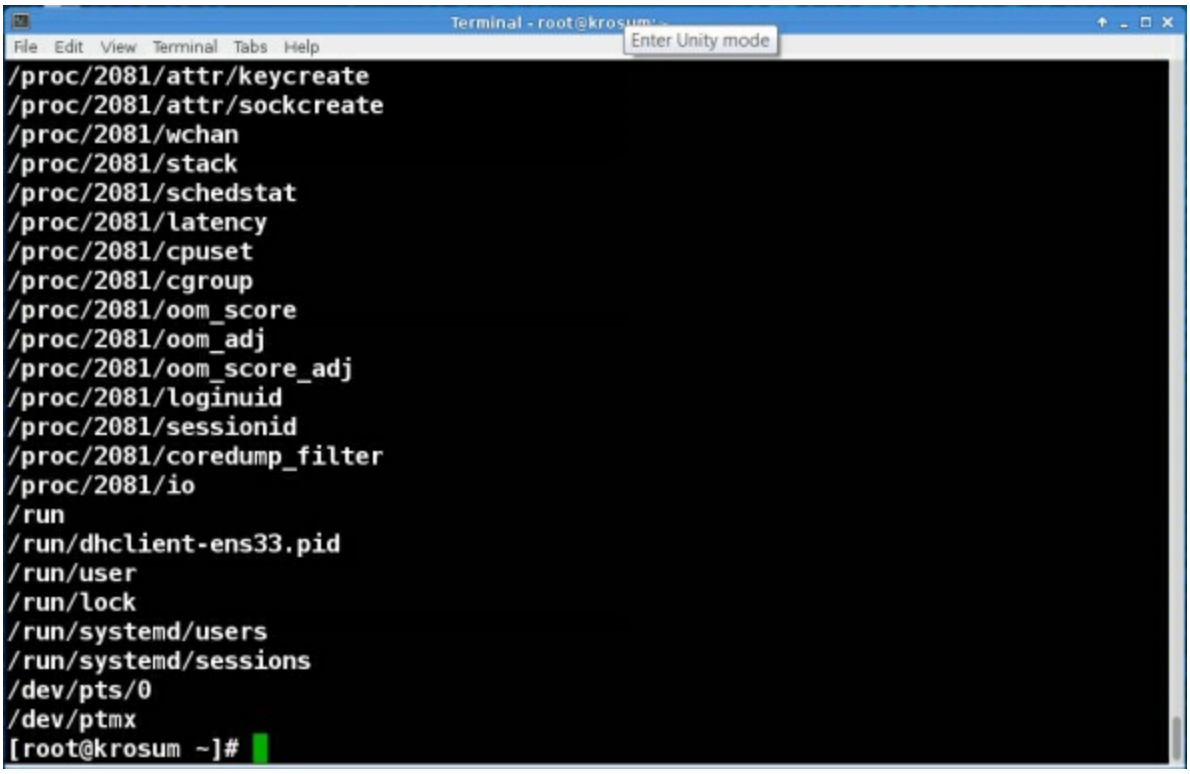


Task : To ***find all the files which are changed in last 1 hour***.



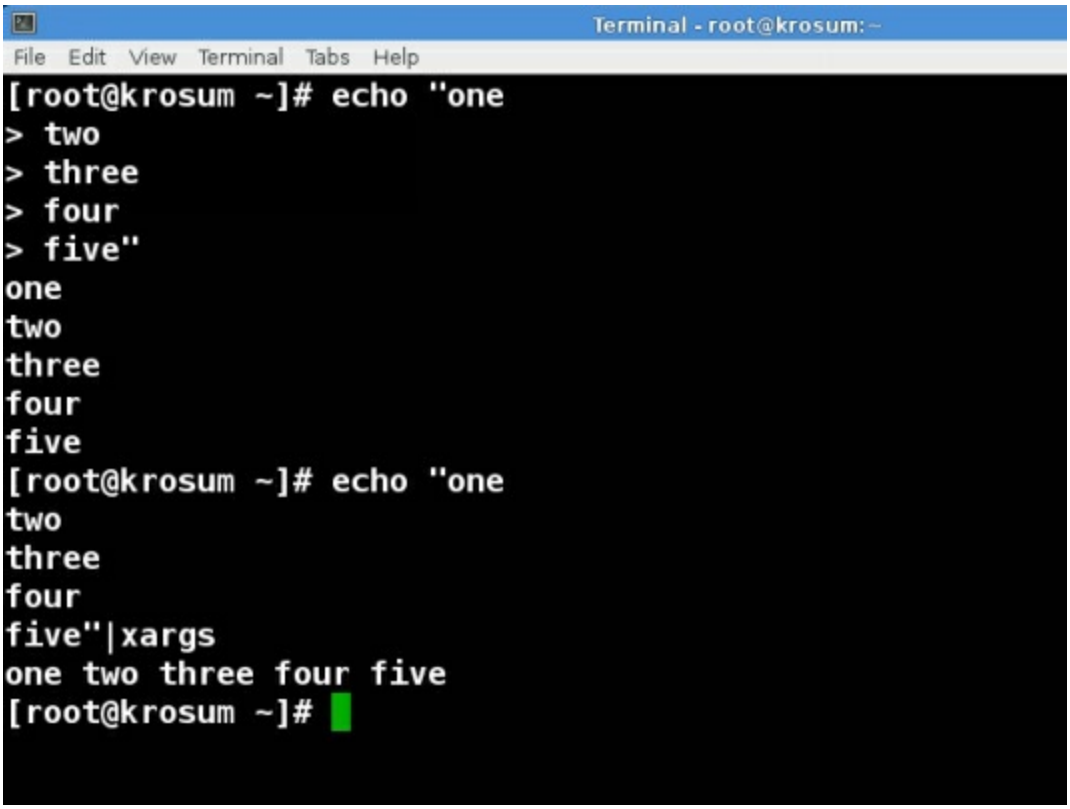
Task : To ***find all the files which are modified in last 1 hour***.

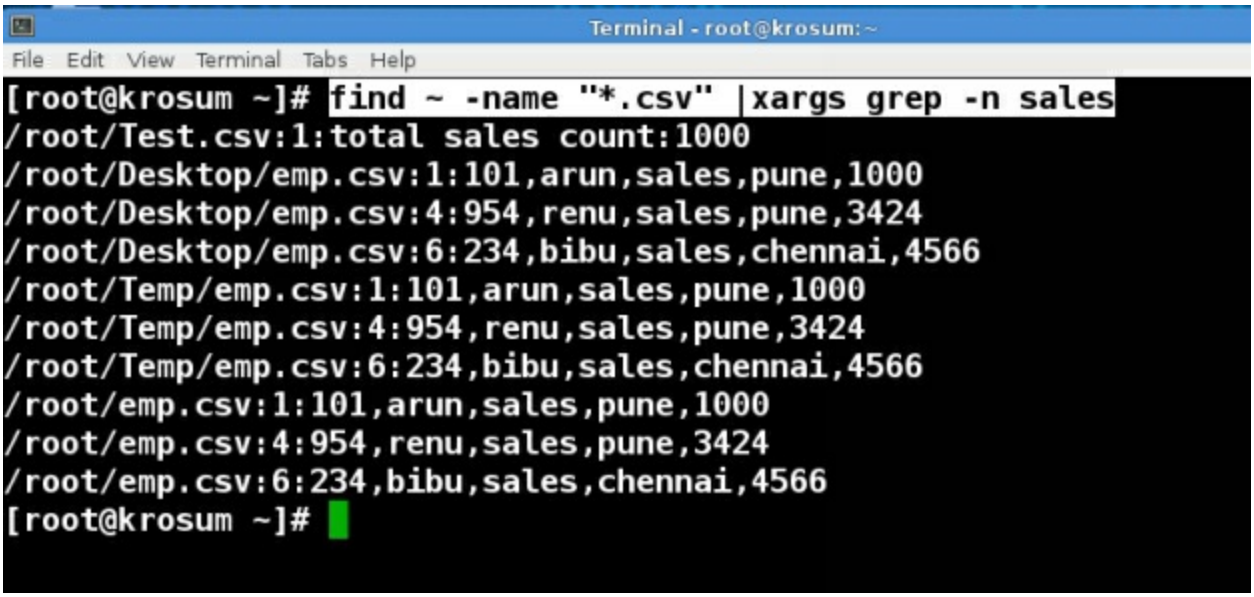

Terminal - root@krosum: ~
File Edit View Terminal Tabs Help
[root@krosum ~]# find / -mmin -60

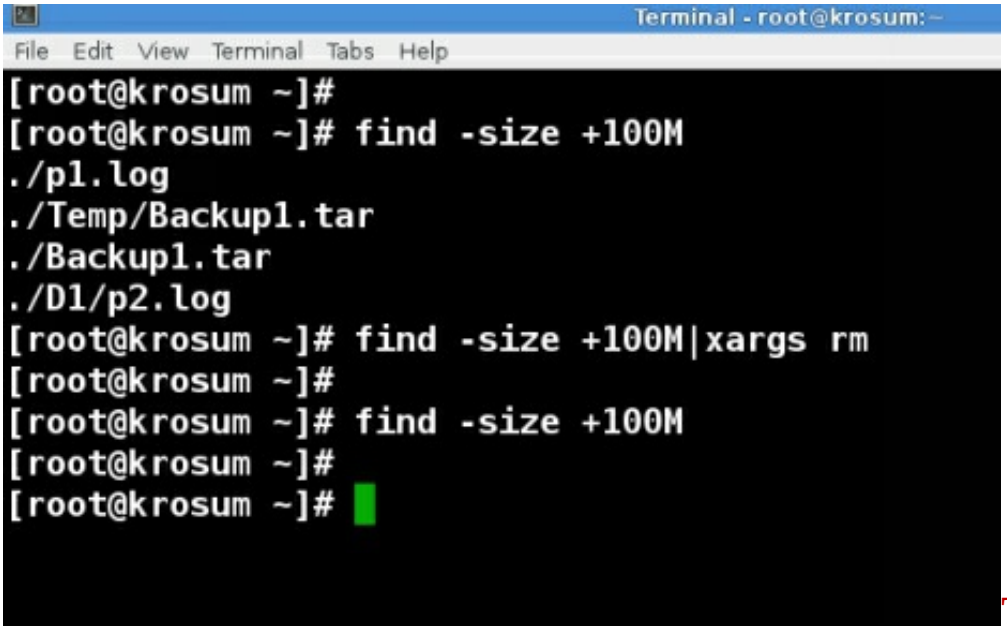

Terminal - root@krosum: ~
Enter Unity mode
File Edit View Terminal Tabs Help
/proc/2081/attr/keycreate
/proc/2081/attr/sockcreate
/proc/2081/wchan
/proc/2081/stack
/proc/2081/schedstat
/proc/2081/latency
/proc/2081/cpuset
/proc/2081/cgroup
/proc/2081/oom_score
/proc/2081/oom_adj
/proc/2081/oom_score_adj
/proc/2081/loginuid
/proc/2081/sessionid
/proc/2081/coredump_filter
/proc/2081/io
/run
/run/dhclient-ens33.pid
/run/user
/run/lock
/run/systemd/users
/run/systemd/sessions
/dev/pts/0
/dev/ptmx
[root@krosum ~]#

xargs

- xargs converts ***input from standard input into arguments*** to a command.
- By default **xargs**displays whatever comes to its stdin as shown below.







Task: search sales keyword from filtered files

find /root -name "*.csv" |xargsgrep -n sales

exec

-execcommand ;*Execute command; true if 0 status is returned*.

All following arguments to find are taken to be arguments to the command until an argument consisting of `**;**' is encountered.

Syntax

find-exec command {} \;

exec

Task: search list of .csv files under /root directory ,from that search pattern is called sales , display matched result to monitor.

• find/root-name "*.csv" -exec grep-n sales {} \;

Task : search all files with size more than 100MB and delete them.

• find / -size +100M -exec /bin/rm {} \;

ABOUT THE AUTHOR

Mr. Palani Karthikeyan is well known for his role as a corporate trainer.

He is also working as a Technical Consultant for krosum labs[i], an Online Technical institution.

He pursued his Bachelor degree in Computer Engineering from The University of Madras and Post Graduate degree M.S. from The Manipal University.

He has more than 16 years of professional experience in corporate world. Until now, He has conducted 350+ corporate trainings. He has acquired profound knowledge in the following areas.

- *High-performance UNIX & LINUX system programming*
- *Oracle Linux Administration & Ansible*
- *Efficient code handling in C / C++ Program.*
- *Unix Shell Script (BASH, KSH, CSH, expect)*
- *Perl Script*
- *Python - flask, django.*
- *TCL script*
- *Ruby Script*

He has worked as a Consultant in various projects like Linux kernel

by-pass technique, System Performance monitoring tools, Education Automation System (EASY). In addition he has developed various ERP Business modules for enterprises.

---

[i]https://www.krosum.com/ email Id- abpalanikarthik@gmail.com
Connect me by LinkedIn – Palani karthikeyan