

# ***LINUX***

SANDEEP KUMAR

# Linux Fundamentals

# WHAT IS LINUX?



Linux is an Operating System which sits in the middle of your hardware and users



# History of Unix



- 1965 Bell Laboratories joins with MIT and General Electric in the development effort for the new operating system, Multics, which would provide multi-user, multi-processor, and multi-level (hierarchical) file system, among its many forward-looking features.
- 1969 AT&T was unhappy with the progress and drops out of the Multics project. Some of the Bell Labs programmers who had worked on this project, Ken Thompson, Dennis Ritchie, Rudd Canaday, and Doug McIlroy designed and implemented the first version of the Unix File System on a PDP-7 along with a few utilities. It was given the name UNIX by Brian Kernighan as a pun on Multics.
- 1971 The system now runs on a PDP-11, with 16Kbytes of memory, including 8Kbytes for user programs and a 512Kbyte disk. Its first real use is as a text processing tool for the patent department at Bell Labs. That utilization justified further research and development by the programming group. UNIX caught on among programmers because it was designed with these features:
  - programmers environment
  - simple user interface
  - simple utilities that can be combined to perform powerful functions
  - hierarchical file system • simple interface to devices consistent with file format • multi-user, multi-process system • architecture independent and transparent to the user.
- \* 1973 Unix is re-written mostly in C, a new language developed by Dennis Ritchie. Being written in this high-level language greatly decreased the effort needed to port it to new machines.

# Linux Adoption timeline



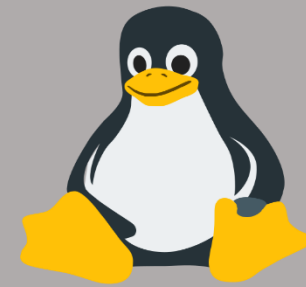
- 1974 Thompson and Ritchie publish a paper in the Communications of the ACM describing the new Unix OS. This generates enthusiasm in the Academic community which sees a potentially great teaching tool for studying programming systems development. Since AT&T is prevented from marketing the product due to the 1956 Consent Decree they license it to Universities for educational purposes and to commercial entities.
- 1977 There are now about 500 Unix sites world-wide.
- 1980 BSD 4.1 (Berkeley Software Development)
- 1983 SunOS, BSD 4.2, SysV
- 1984 There are now about 100,000 Unix sites running on many different hardware platforms, of vastly different capabilities.
- 1988 AT&T and Sun Microsystems jointly develop System V Release 4 (SVR4). This would later be developed into UnixWare and Solaris 2.
- Because of their access to the source code, many programmers helped Linus Torvalds retool and refine the software, and by 1994 Linux kernel (original code) version 1.0 was released.
- 1995 Santa Cruz Operations buys UnixWare from Novell. Santa Cruz Operations and Hewlett-Packard announce that they will jointly develop a 64-bit version of Unix.
- 1996 International Data Corporation forecasts that in 1997 there will be 3 million Unix systems shipped world-wide

# Linux Adoption timeline



- 1998: Many major companies such as IBM, [Compaq](#) and [Oracle](#) announce their support for Linux. [The Cathedral and the Bazaar](#) is first published as an essay (later as a book), resulting in [Netscape](#) publicly releasing the source code to its [Netscape Communicator](#) web browser suite. Netscape's actions and crediting of the essay<sup>[55]</sup> brings Linux's open source development model to the attention of the popular technical press. In addition a group of programmers begins developing the graphical user interface [KDE](#).
- 1999: A group of developers begin work on the graphical environment [GNOME](#), destined to become a free replacement for KDE, which at the time, depends on the, then proprietary, [Qt toolkit](#). During the year IBM announces an extensive project for the support of Linux.
- 2000: Dell announces that it is now the No. 2 provider of Linux-based systems worldwide and the first major manufacturer to offer Linux across its full product line.
- 2002: The media reports that "Microsoft killed Dell Linux"
- 2004: The XFree86 team splits up and joins with the existing X standards body to form the [X.Org Foundation](#), which results in a substantially faster development of the [X server](#) for Linux.
- 2005: The project [openSUSE](#) begins a free distribution from Novell's community. Also the project [OpenOffice.org](#) introduces version 2.0 that then started supporting [OASIS OpenDocument](#) standards.

# Linux Adoption timeline



- 2006: Oracle releases its [own distribution of Red Hat Enterprise Linux](#). Novell and [Microsoft](#) announce cooperation for a better interoperability and mutual patent protection.
- 2007: Dell starts distributing laptops with [Ubuntu](#) pre-installed on them.
- 2009: Red Hat's [market capitalization](#) equals Sun's, interpreted as a symbolic moment for the "Linux-based economy."
- 2011: Version 3.0 of the Linux kernel is released.
- 2012: The aggregate Linux server market revenue exceeds that of the rest of the Unix market.
- 2013: Google's Linux-based [Android](#) claims 75% of the [smartphone](#) market share, in terms of the number of phones shipped.
- 2014: Ubuntu claims 22,000,000 users.
- 2015: Version 4.0 of the Linux kernel is released.
- 2019: Version 5.0 of the Linux kernel is released.<sup>[63]</sup> New features such as AMD FreeSync display support, Raspberry Pi Touchscreen support, Btrfs swap file support and Adiantum data encryption make into the kernel.

# Linux Distributions



- A Linux distribution is a collection of (usually open source) software on top of a Linux kernel. A distribution can bundle server software, system management tools, documentation and many desktop applications in a central secure software repository. A distribution aims to provide a common look and feel, secure and easy software management and often a specific operational purpose.

## Red Hat

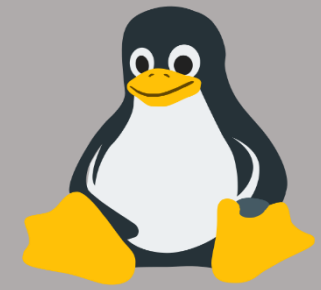
Red Hat is a billion dollar commercial Linux company that puts a lot of effort in developing Linux. They have hundreds of Linux specialists and are known for their excellent support. They give their products (Red Hat Enterprise Linux and Fedora) away for free. While Red Hat Enterprise Linux (RHEL) is well tested before release and supported for up to seven years after release, Fedora is a distro with faster updates but without support.

## Ubuntu

Canonical started sending out free compact discs with Ubuntu Linux in 2004 and quickly became popular for home users (many switching from Microsoft Windows). Canonical wants Ubuntu to be an easy to use graphical Linux desktop without need to ever see a command line. Of course they also want to make a profit by selling support for Ubuntu.



# Linux Distributions



## **Debian**

There is no company behind Debian. Instead there are thousands of well organised developers that elect a Debian Project Leader every two years. Debian is seen as one of the most stable Linux distributions. It is also the basis of every release of Ubuntu. Debian comes in three versions: stable, testing and unstable. Every Debian release is named after a character in the movie Toy Story.

## **Other Distributions**

There are others like CentOS, Oracle Enterprise Linux and Scientific Linux are based on Red Hat Enterprise Linux and share many of the same principles, directories and system administration techniques. Linux Mint, Edubuntu and many other \*buntu named distributions are based on Ubuntu and thus share a lot with Debian. There are hundreds of other Linux distributions

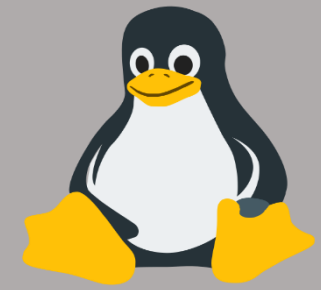
# Choosing A Distribution



Choosing a Linux distribution will always be based on a company or individual choice but here are some name reason(s) which can help decide

Distribution Name	Reason
Red Hat Enterprise (RHEL)	Good Support Contract
Centos	You want Red Hat without the support contract from Red Hat.
Fedora	Need it on Laptop/Desktop
Linux Mint	Graphical desktop to play music, watch movies
Debian/Ubuntu	Servers, laptops
Kali	For Ethical Hacking

# LAB : Installation ..

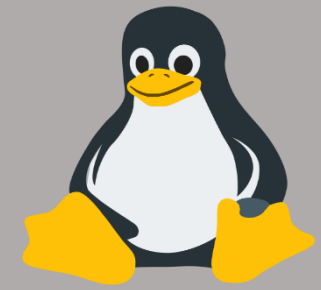


- This module is a step by step demonstration of an actual installation of CentOS 7. We start by downloading an image from the internet and install CentOS 7 as a virtual machine in Virtualbox. We will also do some basic configuration of this new machine like setting an ip address and fixing a hostname. This procedure should be very similar for other versions of CentOS, and also for distributions like RHEL (Red Hat Enterprise Linux) or Fedora. This procedure can also be helpful if you are using another virtualization solution.

## Download Links

- Centos iso : [http://isoredirect.centos.org/centos/7/isos/x86\\_64/](http://isoredirect.centos.org/centos/7/isos/x86_64/)
- Virtualbox: <https://www.virtualbox.org/wiki/Downloads>
- Open Virtualbox post installation
- Click on New
- Create a New Virtual Machine

# Installation




?×

← Create Virtual Machine

## Name and operating system

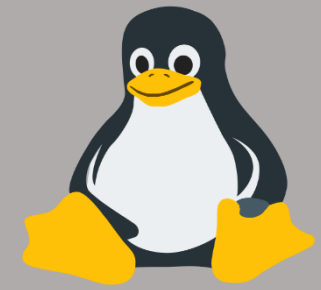
Please choose a descriptive name for the new virtual machine and select the type of operating system you intend to install on it. The name you choose will be used throughout VirtualBox to identify this machine.

Name:

Type:  

Version:

# Installation



← Create Virtual Machine

## Memory size

Select the amount of memory (RAM) in megabytes to be allocated to the virtual machine.

The recommended memory size is **1024 MB**.

4 MB 4096 MB

1024 MB

Next Cancel

← Create Virtual Machine

## Hard disk

If you wish you can add a virtual hard disk to the new machine. You can either create a new hard disk file or select one from the list or from another location using the folder icon.

If you need a more complex storage set-up you can skip this step and make the changes to the machine settings once the machine is created.

The recommended size of the hard disk is **8.00 GB**.

☐ Do not add a virtual hard disk

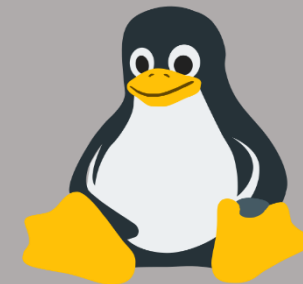
☒ Create a virtual hard disk now

☐ Use an existing virtual hard disk file

Empty

Create Cancel

# Installation



? ×

← Create Virtual Hard Disk

### Hard disk file type

Please choose the type of file that you would like to use for the new virtual hard disk. If you do not need to use it with other virtualization software you can leave this setting unchanged.

☒ VDI (VirtualBox Disk Image)

☐ VHD (Virtual Hard Disk)

☐ VMDK (Virtual Machine Disk)

Expert Mode

Next

Cancel

? ×

← Create Virtual Hard Disk

### Storage on physical hard disk

Please choose whether the new virtual hard disk file should grow as it is used (dynamically allocated) or if it should be created at its maximum size (fixed size).

A **dynamically allocated** hard disk file will only use space on your physical hard disk as it fills up (up to a maximum **fixed size**), although it will not shrink again automatically when space on it is freed.

A **fixed size** hard disk file may take longer to create on some systems but is often faster to use.

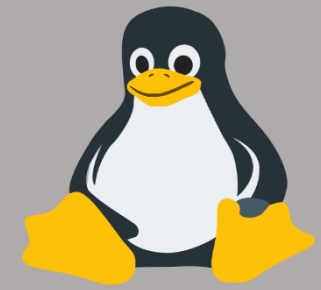
☒ Dynamically allocated

☐ Fixed size

Next

Cancel


# Installation




← Create Virtual Hard Disk

### File location and size

Please type the name of the new virtual hard disk file into the box below or click on the folder icon to select a different folder to create the file in.

Centos 

Select the size of the virtual hard disk in megabytes. This size is the limit on the amount of file data that a virtual machine will be able to store on the hard disk.

4.00 MB  2.00 TB

Create Cancel

## LAB : VM snapshot ..

- Snapshot is used to restore the state of your VM if in case something gets corrupted in your VM or you are not able to boot your VM in near future.



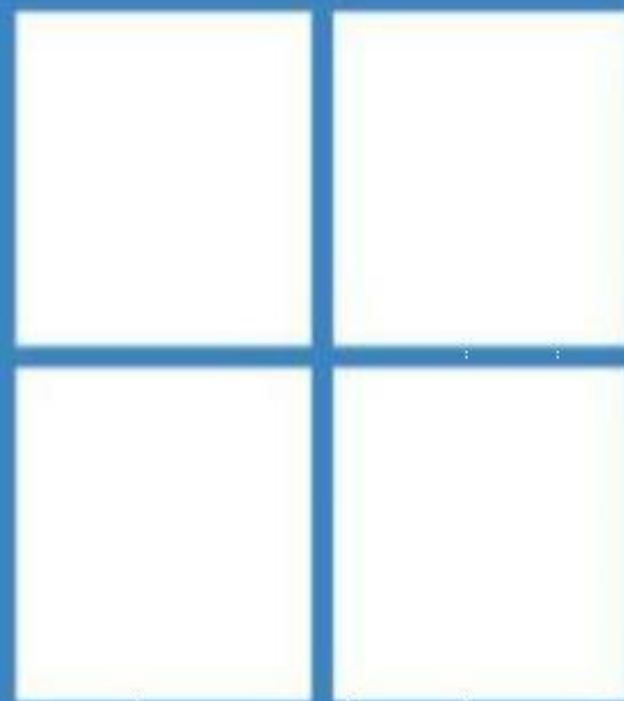
# VIRTUAL MACHINE MGMT



# Linux



# Windows





## Comparison



- Linux

- Open Source
- Free
- Free Software
- Live CD Distribution
- Secure
- NO
- Low Hardware Cost
- Customizable add features

- Windows

Closed Source  
Cost 150\$-320\$  
Cost Software  
NO  
Insecure  
Virus, Malware  
High Hardware Cost  
Not Customizable



# Companies That Use Linux



WIKIPEDIA



Google

TESLA

NETFLIX

IBM



amazon

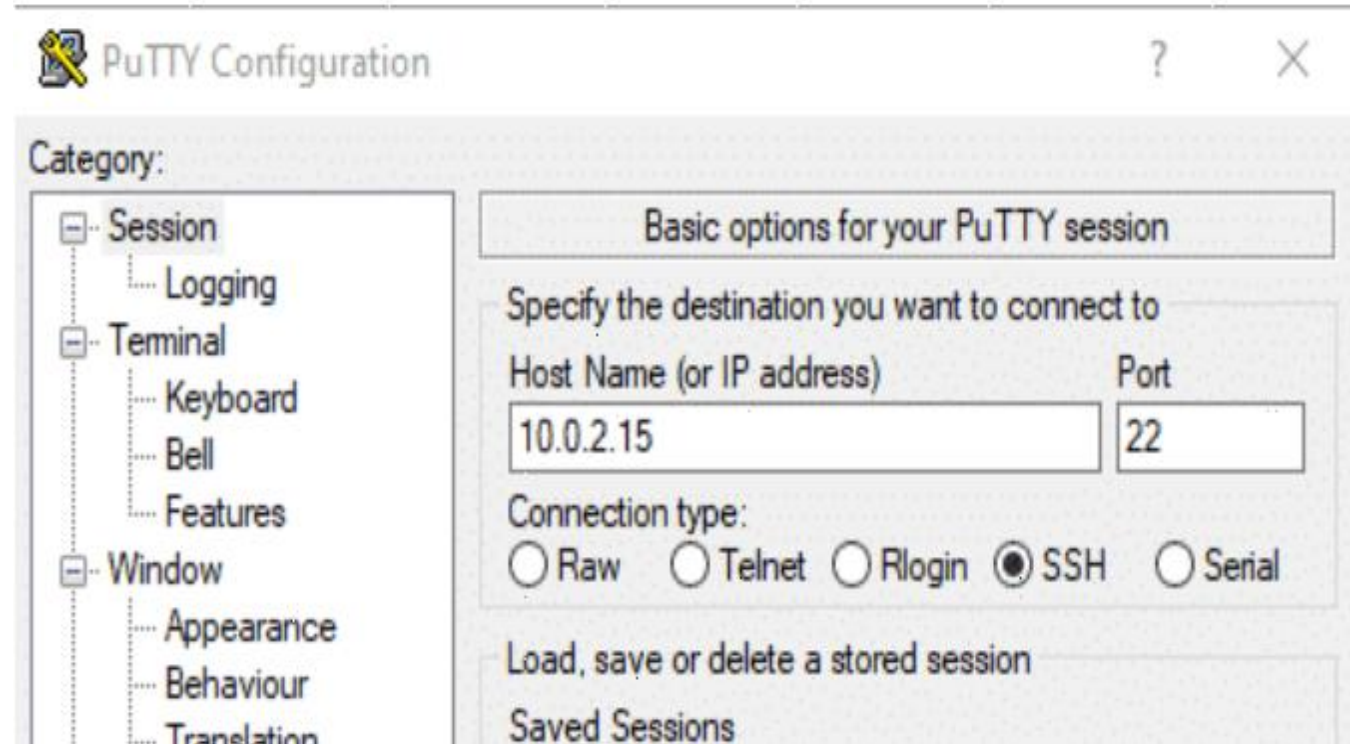


SPACEX

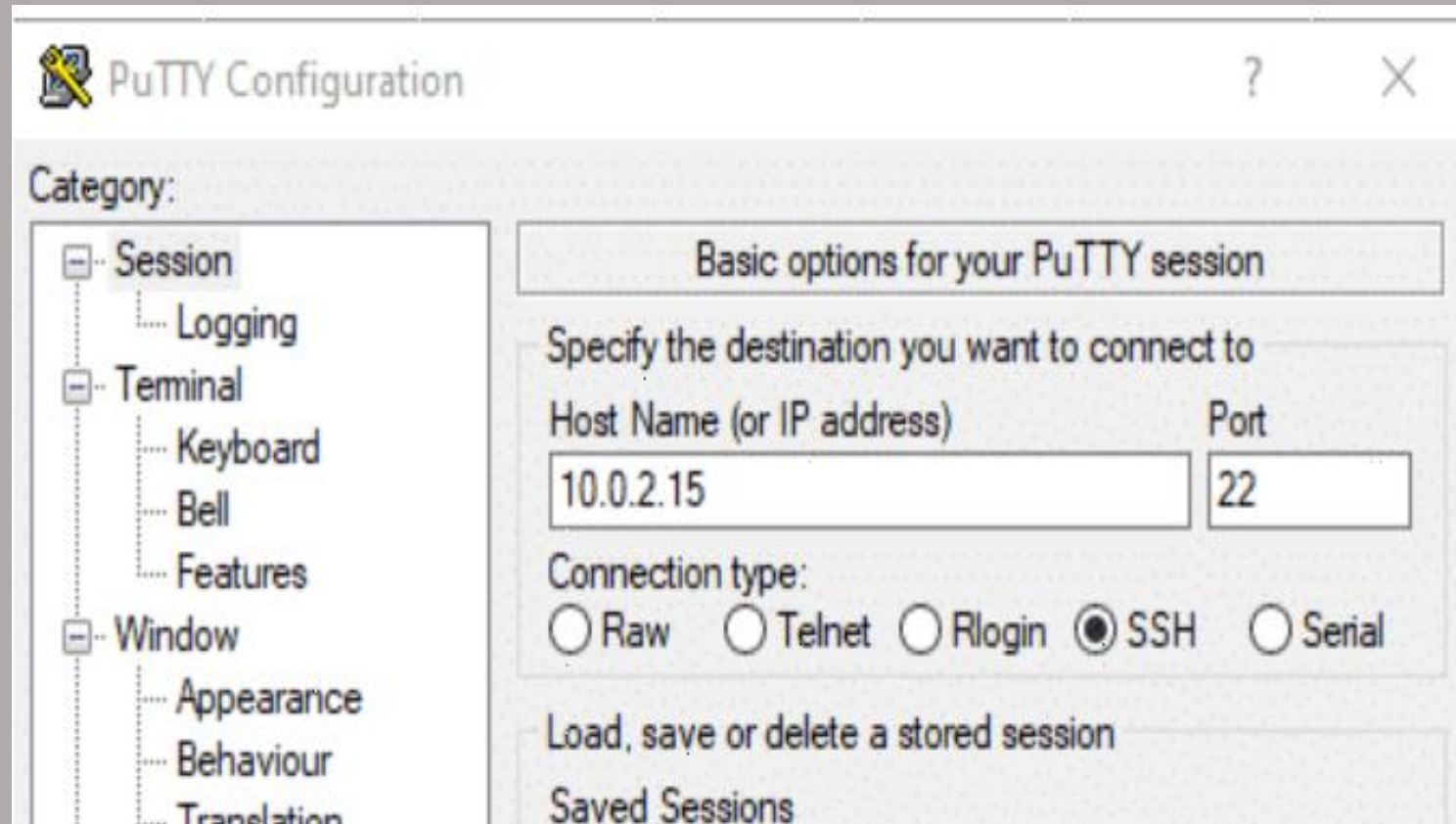
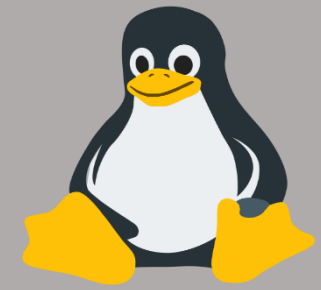
# Command Prompts and Getting Prompts Back

- While using terminal , type ctrl+c whenever you are stuck on any command.

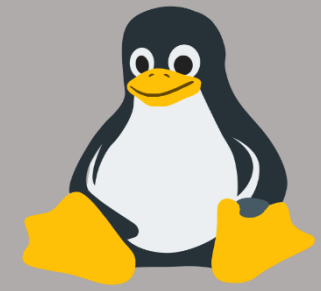
# Accessing systems from windows



# LAB: INSTALLATION OF PUTTY TO ACCESS LINUX FROM WINDOWS ..



# Network Command TO CHECK IP OF MACHINE



- ifconfig
- Ip addr
- Ip a



## LAB : CONNECT LINUX VIA PUTTY

- Add the ip to the hostname tab in putty
- Enter the username of the user created.
- Enter the password

# First steps



- Now that we have Linux Installed I am sure we all be interested in exploring but as with everything we all need some help .The help in Linux or Unix apart from a google search but what if you do not have access to internet or company policy does not allow search engines .

The help in linux is provided by Man

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

# Brief history of Unix

- In 1960's MIT, Bell Labs and GE were working on a timesharing operating system called **multics** but that could not be successful.
- Unix was originally developed at AT&T Bell Laboratories by Ken Thompson and Dennis Ritchie in 1969-1970. **Was called AT&T Unix.**
- In 1973, Unix was migrated from assembly language to C. This made possible for Unix to be installed and run on many different types of computers which had a C compiler.

# Brief history of Unix

- AT&T gave Unix to the Universities for research. The first one was University of California at Berkeley, that developed "Berkeley Unix" or BSD (Berkeley Standard Distribution).
- The influence of Unix in academic circles led to large-scale adoption of Unix by commercial startups, including [HP-UX](#), [Solaris](#), [AIX](#), and [Xenix](#)

# Popular Unix versions

Version	Developped By
UNIX SYSTEM	AT & T
UNIX BSD 4.x.x	BERKELEY SW DIST
XENIX	MICROSOFT
ULTRIX	DEC
AIX	IBM
HP-UX	HP
SOLARIS	SUN MICROSYSTEM

# GNU/GPL

- In earlier days, programmers used to share code and software among themselves but this trend changed when software started to be used under a business model.
- Companies started restricting sharing of source code and started selling software under copyrights.

# GNU/GPL

- Frustrated by this, Richard Stallman founded GNU project under which he planned to develop and distribute software as opensource and free.
- He formed GPL (General Public License) which laid rules as to how the free software can be used and distributed.
- Most commonly known as the GPL.

# History of Linux

- In the 1985, a professor by name Andy Tanenbaum wrote a Unix like Operating system from scratch for the Intel i386 platform. He named it Minix. It was used for teaching students about operating systems.
- In 1990, Linus Torvalds came to know about Minix and wanted to upgrade it by putting in more features and improvements but he was prohibited by Tanenbaum to do so.
- Then Linus decided to write his own kernel and released it under GPL. This is now popularly known as Linux.



# RHEL

- As Linux kernel was free and opensource, so anyone was free to modify and distribute it under GPL.
- So, lots of “Linux distributions”, both completely free and commercialized started appearing. All used the same Linux kernel.
- One of those was Red Hat Linux which was the product of a U.S. company called Red Hat in 1993.
- Red Hat Enterprise Linux 7 is the latest commercial offering from Red Hat.

# Popular Linux Distributions

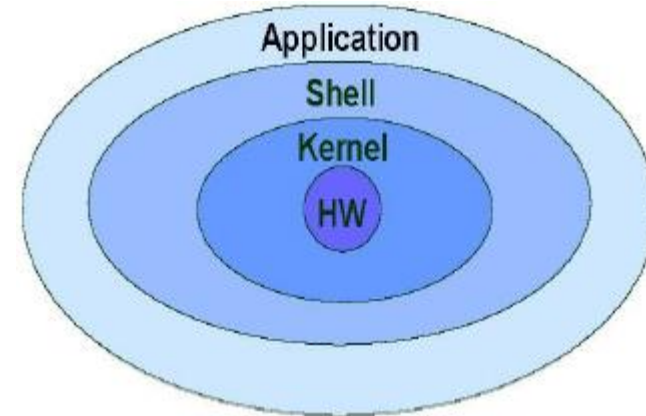
- CentOS
- Ubuntu
- Debian
- Fedora
- Linux Mint
- SuSE
- Slackware
- RHEL

# Installation of Linux

- To install CentOS Linux OS, you will need to download the ISO images (CD Images) of the installation CD-ROM from <https://www.centos.org/download/>
- Lets begin with the Installation and then understand about the major Linux concepts.

# Linux Architecture

- **Kernel:** the heart of the UNIX Operating System. Keeps track of the disks, tapes, printers, terminals, communication lines and any other devices attached to the computer. It also interfaces between the computer's hardware and the users.
- **Shell:** a program that interfaces between the user and the UNIX Operating System. It listens to the user's terminal and translates the actions requested by the user. There are a number of different Shells that may be used.
- **Application:** application programs, such as Word Processors, spreadsheets and Database Management Systems, may be installed alongside the UNIX Commands. A user may run an utility or application through the shell.



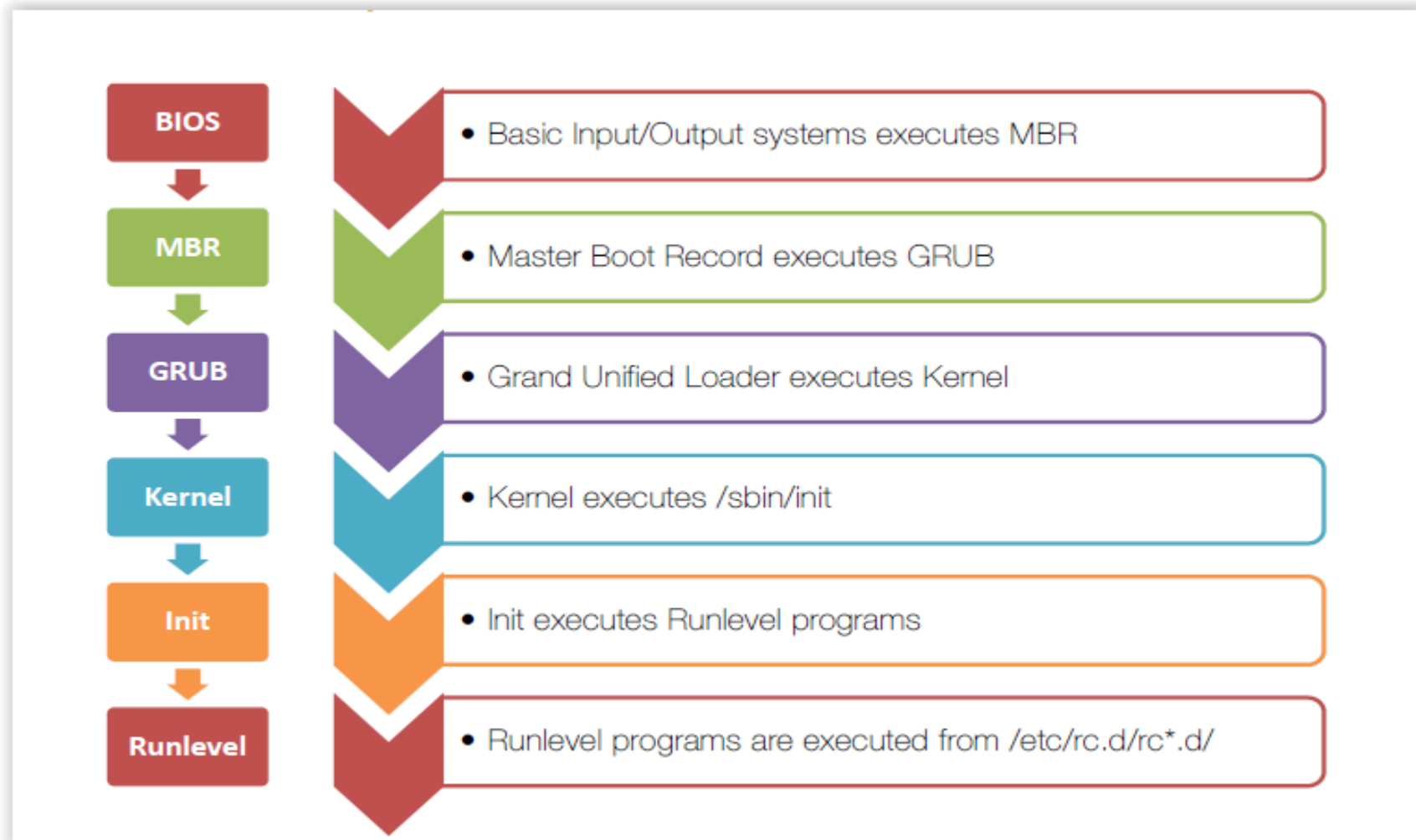
# What is Kernel

- Kernel is the program that controls the resources of the computer and allocates them to the user.
- Kernel basically consists of
  - code for process and memory management
  - code for I/O from hardware
  - code for handling filesystem
- Unix kernel is set of essential routines and data structures that are always stored in main memory when the system is running.
- Rest of the OS is stored on disk, to be paged in as necessary.

# Functions of Kernel

- Process Management
  - Allocation of resources to a process
  - Synchronization among processes
- Memory Management
  - Allocating and deallocating memory to programs
- Device Management
  - Controlling several devices attached to the system
- Storage Management
  - Disk Management
  - Disk scheduling

# System Startup Sequence



# System runlevels

- Runlevels act as a method to define what processes are started and stopped. RHEL (upto ver 6) uses 7 runlevels (0-6).
- In RHEL 7 runlevels are replaced by targets.

0 – Halt

1 – Single user text mode

2 – Not used (user – definable)

3 – Full multi-user text mode

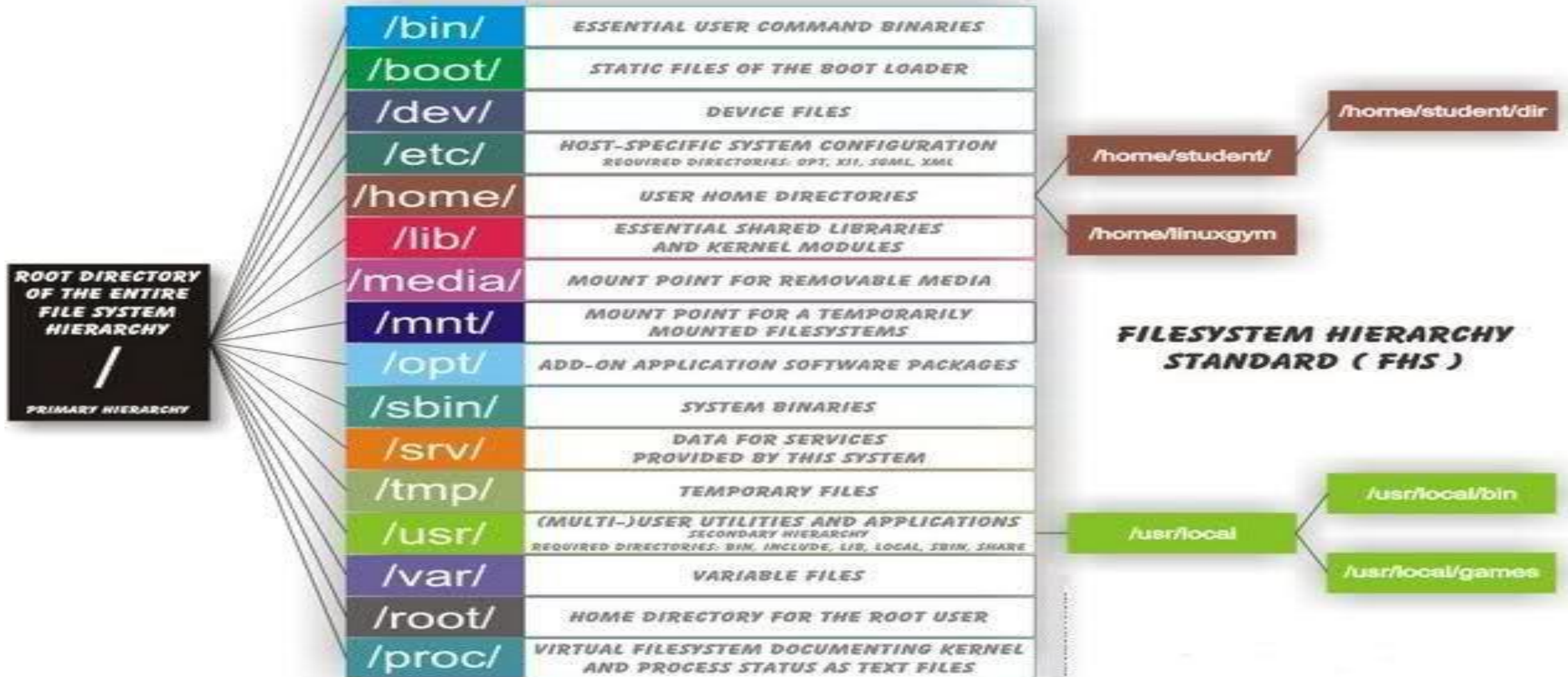
4 – Not used – ( user – definable )

5 -- Full multi-user graphical mode (with an X-based login screen)

6 -- Reboot



# Directory Hierarchy



`/etc/fstab`

- All mounted filesystems can be found here and new can be added.

`/etc/resolve.conf`

- To add DNS server entry here for resolving your hostname with it.

`/etc/ssh/sshd_config`

- For enabling remote access via SSH port 22

`/etc/passwd @ /etc/group`

- having password and shell entries
- and having group entries

`/etc/inittab`

- Having details of all run levels

`/etc/init.d`

- Contains links to the run-level scripts. These scripts are linked from files in the `/etc/rc?.d` directories to have each service associated with a script started or stopped for the particular run level. The `?` is replaced by the run-level number (0 through 6).

`/etc/hosts`

- Contains IP addresses and hostnames that you can reach from your computer. (Usually this file is used just to store names of computers on your LAN or small private network.)

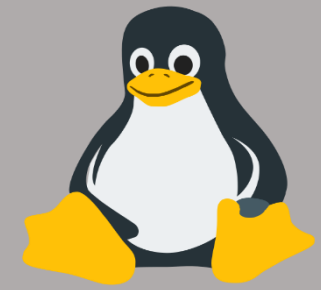
`/etc/hosts.allow`

- Lists host computers that are allowed to use certain TCP/IP services from the local computer.

`/etc/hosts.deny`

- Lists host computers that are not allowed to use certain TCP/IP services from the local computer (doesn't exist by default).

# First steps



- **Man \$command**

Type man followed by a command (for which you want help) and start reading. Press q to quit the manpage. Some man pages contain examples (near the end).

man ls

man ls

LS(1)

User Commands

LS(1)

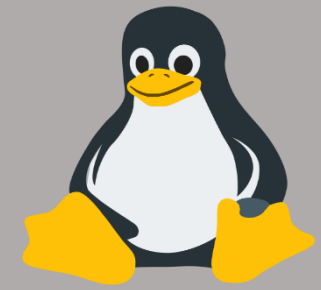
NAME

ls - list directory contents

SYNOPSIS

ls [OPTION]... [FILE]...

# First steps



Most configuration files have their own manual.

`man $configfile`

E.g

```
[cloud_user@ravisingh1c ~]$ man sysctl.conf
```

```
SYSCTL.CONF(5)           File Formats           SYSCTL.CONF(5)
```

## NAME

`sysctl.conf` - `sysctl` preload/configuration file

## DESCRIPTION

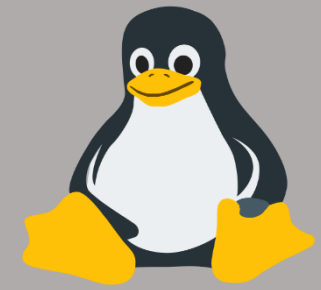
`sysctl.conf` is a simple file containing `sysctl` values to be read in and set by `sysctl`. The syntax is simply as follows:

```
# comment
```

```
; comment
```

```
token = value
```

# First steps



```
man $daemon
```

This is also true for most daemons (background programs) on your system.

```
[cloud_user@ravisingh1c ~]$ man rsyslogd
```

```
RSYSLOGD(8)          Linux System Administration          RSYSLOGD(8)
```

## NAME

rsyslogd - reliable and extended syslogd

## SYNOPSIS

```
rsyslogd [ -d ] [ -D ] [ -f config file ] [ -i pid file ] [ -n ] [ -N  
level ] [ -C ] [ -v ]
```

# Working With Directories



- This module is a brief overview of the most common commands to work with directories: `pwd`, `cd`, `ls`, `mkdir` and `rmdir`. These commands are available on any Linux (or Unix) system. This module also discusses absolute and relative paths and path completion in the bash shell.

- `pwd`

The you are here sign can be displayed with the `pwd` command (Print Working Directory). Go ahead, try it: Open a command line interface (also called a terminal, console or xterm) and type `pwd`. The tool displays your current directory.

```
[cloud_user@ravisingh1c ~]$ pwd
/home/cloud_user
```

```
. cd
```

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

```
[cloud_user@ravisingh1c ~]$ cd /etc
[cloud_user@ravisingh1c etc]$ pwd
/etc
```

# Working With Directories



```
[cloud_user@ravisingh1c ~]$ cd /bin
```

```
[cloud_user@ravisingh1c bin]$ pwd
```

```
/bin
```

```
[cloud_user@ravisingh1c bin]$ cd /home/cloud_user
```

```
[cloud_user@ravisingh1c ~]$ pwd
```

```
/home/cloud_user
```

Pro Tip : The cd is also a shortcut to get back into your home directory. Just typing cd without a target directory, will put you in your home directory.

```
[cloud_user@ravisingh1c ~]$ cd /etc
```

```
[cloud_user@ravisingh1c etc]$ cd
```

```
[cloud_user@ravisingh1c ~]$ pwd
```

```
/home/cloud_user
```

```
[cloud_user@ravisingh1c ~]$
```

Typing cd ~ has the same effect.

# Working With Directories



```
[cloud_user@ravisingh1c ~]$ cd /bin
[cloud_user@ravisingh1c bin]$ pwd
/bin
[cloud_user@ravisingh1c bin]$ cd ~
[cloud_user@ravisingh1c ~]$ pwd
/home/cloud_user
[cloud_user@ravisingh1c ~]$
```

**cd ..**

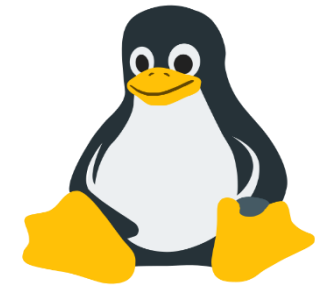
```
[cloud_user@ravisingh1c ~]$ pwd
/home/cloud_user
[cloud_user@ravisingh1c ~]$ cd ..
[cloud_user@ravisingh1c home]$ pwd
/home
[cloud_user@ravisingh1c home]$
```

To stay in the current directory, type `cd .`

;-) We will see useful use of the `.` character representing the current directory later.



# ls, cd, pwd, whoami



- **ls**

ls command is used to list the contents of a directory with ls.

```
[cloud_user@ravisingh1c ~]$ ls
Desktop my-login.pp
[cloud_user@ravisingh1c ~]$
```

- **ls -a**

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

```
[cloud_user@ravisingh1c ~]$ ls -a
.      .bash_logout .cache Desktop .mozilla .viminfo
..     .bash_profile .config .esd_auth my-login.pp .vnc
```

# ls



- **ls -l**

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

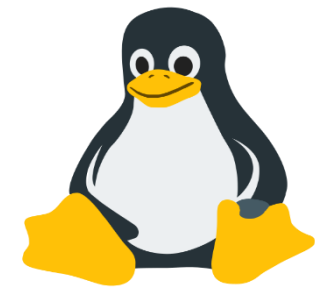
```
[cloud_user@ravisingh1c ~]$ ls -l
total 4
drwxr-xr-x. 2 cloud_user cloud_user  6 May  9 15:58 Desktop
-rw-r--r--. 1 cloud_user cloud_user 1297 Aug 27 21:12 my-login.pp
[cloud_user@ravisingh1c ~]$
```

- **ls -lh**

Very frequently used ls option is -h. It shows the numbers (file sizes) in a more human readable format.

```
[cloud_user@ravisingh1c ~]$ ls -lh
total 4.0K
drwxr-xr-x. 2 cloud_user cloud_user  6 May  9 15:58 Desktop
-rw-r--r--. 1 cloud_user cloud_user 1.3K Aug 27 21:12 my-login.pp
[cloud_user@ravisingh1c ~]$
```

# ls



**Pro Tip: The same command `ls -lh` can be typed in different variations all providing same output**

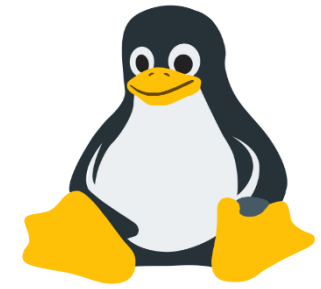
```
[cloud_user@ravisingh1c ~]$ ls -l -h
total 4.0K
drwxr-xr-x. 2 cloud_user cloud_user  6 May  9 15:58 Desktop
-rw-r--r--. 1 cloud_user cloud_user 1.3K Aug 27 21:12 my-login.pp
[cloud_user@ravisingh1c ~]$ ls -hl
total 4.0K
drwxr-xr-x. 2 cloud_user cloud_user  6 May  9 15:58 Desktop
-rw-r--r--. 1 cloud_user cloud_user 1.3K Aug 27 21:12 my-login.pp
[cloud_user@ravisingh1c ~]$ ls -h -l
total 4.0K
drwxr-xr-x. 2 cloud_user cloud_user  6 May  9 15:58 Desktop
-rw-r--r--. 1 cloud_user cloud_user 1.3K Aug 27 21:12 my-login.pp
[cloud_user@ravisingh1c ~]$
```

# Lab : Navigating file system

- NAVIGATE THROUGH EACH DIRECTORY UNDER / AND USE THE COMMANDS

`ls, ls -a , ls -l , ls -lh ,ls -ltr cd , pwd` TO BROWSE THE FILES.

# Linux file tree



- This Module takes a look at the most common directories in the Linux file tree. It also shows that on Unix everything is a file. The module is very important to understanding how a Linux system distributes contents to make it easier to navigate through when most needed in every day use .

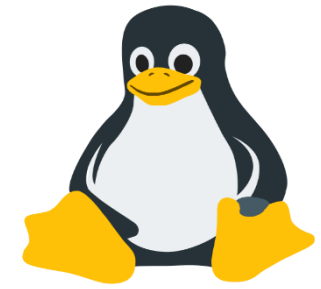
## File system hierarchy standard

Many Linux distributions partially follow the Filesystem Hierarchy Standard. The FHS may help make more Unix/Linux file system trees conform better in the future.

The FHS is available online at <http://www.pathname.com/fhs/> where we read: "The filesystem hierarchy standard has been designed to be used by Unix distribution developers, package developers, and system implementers. However, it is primarily intended to be a reference and is not a tutorial on how to manage a Unix filesystem or directory hierarchy."

**Pro tip:** Run below man command

# Linux file tree



- **Man hier**

```
[cloud_user@ravisingh1c ~]$ man hier
HIER(7)          Linux Programmer's Manual
HIER(7)

NAME
  hier - description of the file system hierarchy
```

## DESCRIPTION

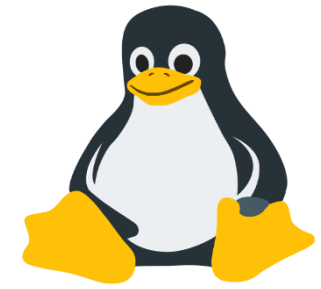
A typical Linux system has, among others, the following directories:

/ This is the root directory. This is where the whole tree starts.

/bin This directory contains executable programs which are needed in single user mode and to bring the system up or repair it.

/boot Contains static files for the boot loader. This directory holds only the files which are needed during the boot process. The map installer and configuration files should go to /sbin and /etc

# Linux file tree

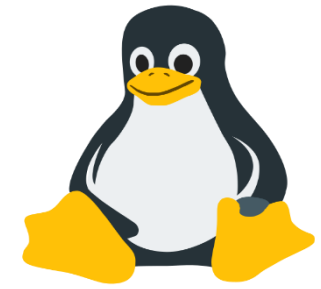


- **The root directory /**

All Linux systems have a directory structure that starts at the root directory. The root directory is represented by a forward slash, like this: /. Everything that exists on your Linux system can be found below this root directory. Let's take a brief look at the contents of the root directory.

```
[cloud_user@ravisingh1c ~]$ ls /  
bin  dev  home  lib64  mnt  proc  run  srv  tmp  var  
boot  etc  lib  media  opt  root  sbin  sys  usr  
[cloud_user@ravisingh1c ~]$
```

# Linux file tree



- **Binary directories**

Binaries are files that contain compiled source code (or machine code). Binaries are also called executables.

- **/bin**

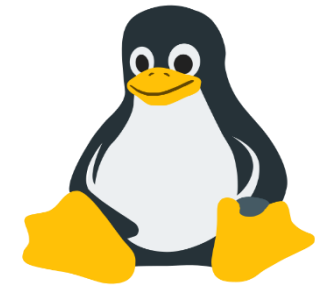
The /bin directory contains binaries for use by all users.

In the screenshot below you see common Unix/Linux commands like cat, ls, grep and so on. We will cover few of them as we go during the course .

```
[cloud_user@ravisingh1c ~]$ ls /bin
[                nano
a2p              nautilus
aclocal          nautilus-autorun-software
aclocal-1.13     nautilus-desktop
addr2line        ncurses5-config
alias            ncursesw5-config
amazon-ssm-agent ndptool
animate          needs-restarting
appstream-compose neqn
```



# Linux file tree



- **Other /bin directories**

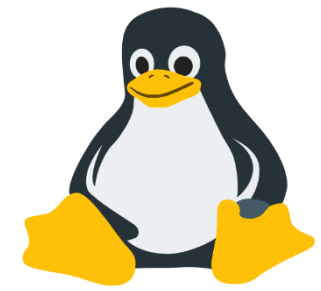
You can find a /bin subdirectory in many other directories. Some applications, often when installed directly from source will put themselves in /opt. A apache server installation can use /opt/httpd/bin to store its binaries.

- **/sbin**

/sbin contains binaries to configure the operating system. Many of the system binaries require root privilege to perform certain tasks. Below a screenshot containing system binaries to manage httpd, partition a disk and create an ext4 file system.

```
[cloud_user@ravisingh1c ~]$ ls -l /sbin/fdisk /sbin/mkfs  
/sbin/apachectl  
-rwxr-xr-x. 1 root root  4290 Jul 29 17:18 /sbin/apachectl  
-rwxr-xr-x. 1 root root 200504 Mar 14 10:37 /sbin/fdisk  
-rwxr-xr-x. 1 root root  11592 Mar 14 10:37 /sbin/mkfs  
[cloud_user@ravisingh1c ~]$
```

# Linux file tree

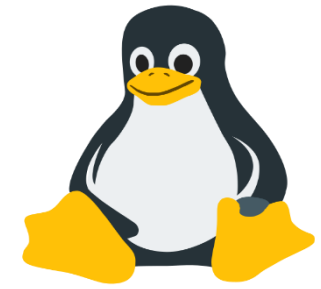


- **/lib**

Binaries found in /bin and /sbin often use shared libraries located in /lib. Below is the partial contents of /lib.

```
[cloud_user@ravisingh1c ~]$ ls /lib
binfmt.d  gems  modprobe.d  rpm  tmpfiles.d
debug     grub  modules     sendmail  tuned
dracut    kbd   modules-load.d  sendmail.postfix  udev
firmware  kdump NetworkManager sse2      yum-plugins
fontconfig kernel polkit-1     sysctl.d
games     locale python2.7    systemd
[cloud_user@ravisingh1c ~]$
```

# Linux file tree



- **/lib/modules**

The Linux kernel loads kernel modules from `/lib/modules/$kernel-version/`.

- **/lib32 and /lib64**

We currently are in a transition between 32-bit and 64-bit systems.

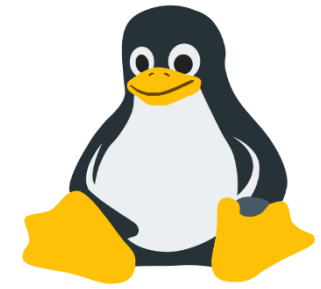
Therefore, you may encounter directories named `/lib32` and `/lib64` which clarify the register size used during compilation time of the libraries. A 64-bit computer may have some 32-bit binaries and libraries for compatibility with legacy applications. This screenshot uses the file utility to demonstrate the difference.

- **File /lib64/libc-2.17.so**

`/lib64/libc-2.17.so`: ELF 64-bit LSB shared object, x86-64, version 1 (GNU/Linux), dynamically linked (uses shared libs), BuildID[sha1]=426a04647352308628f2091a30d347edeeded787, for GNU/Linux 2.6.32, not stripped

The ELF (Executable and Linkable Format) is used in almost every Unix-like operating system since System V.

# Linux file tree



- **/opt**

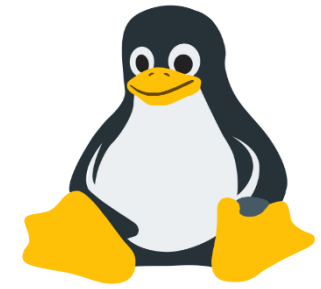
The /opt directory mostly is to store optional software. These are for third party software installs. You may find an empty /opt directory on many systems.

```
[cloud_user@ravisingh1c opt]$ ls -ltr
total 0
drwxr-xr-x. 7 root root 102 Aug 18 2014 openoffice4
drwxr-xr-x. 3 root root 16 Apr 15 2016 amazon
drwxr-xr-x. 5 root root 91 Apr 10 15:24 websh
[cloud_user@ravisingh1c opt]$
```

A large package can install all its files in /bin, /lib, /etc subdirectories within /opt/ \$packagename/.

If for example the package is called welogicb, then it installs in /opt/welogicb, putting binaries in /opt/welogicb/bin and manpages in /opt/welogicb/man.

# Configuration directories



- **/boot**

The `/boot` directory contains all files needed to boot the computer. These files are static in nature and don't change much unless there are kernel or module updates .

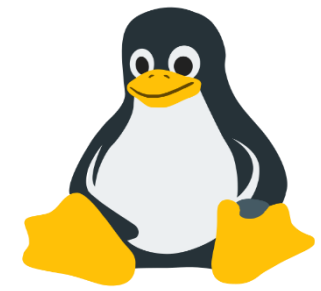
On Linux systems one can typically find the `/boot/grub` directory here. `/boot/grub` contains `/boot/grub/grub.cfg` (older versions of linux used to have `/boot/grub/grub.conf`) which defines the boot menu that is displayed before the kernel starts.

- **/etc**

All of the machine-specific configuration files should be located in `/etc`.

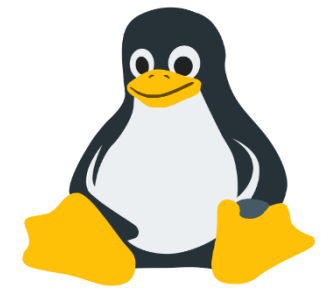
Historically `/etc` stood for etcetera, today people often use the Editable Text Configuration backronym. Most of the times the name of a configuration files is the same as the name of application, daemon, or protocol with `.conf` added as the extension for one to recognise the type/category .

# Configuration directories



```
[cloud_user@ravisingh1c opt]$ ls /etc/*.conf
/etc/asound.conf /etc/ld.so.conf /etc/resolv.conf
/etc/chrony.conf /etc/libaudit.conf /etc/rsyncd.conf
/etc/dnsmasq.conf /etc/libuser.conf /etc/rsyslog.conf
/etc/dracut.conf /etc/locale.conf /etc/sestatus.conf
/etc/e2fsck.conf /etc/logrotate.conf /etc/sudo.conf
/etc/fuse.conf /etc/man_db.conf /etc/sudo-ldap.conf
/etc/GeoIP.conf /etc/mke2fs.conf /etc/sysctl.conf
/etc/grub.conf /etc/nfs.conf /etc/tcsd.conf
/etc/host.conf /etc/nfsmount.conf /etc/updatedb.conf
/etc/idmapd.conf /etc/nsswitch.conf /etc/vconsole.conf
/etc/kdump.conf /etc/oddjobd.conf /etc/yum.conf
/etc/krb5.conf /etc/request-key.conf
```

# Configuration directories



- **/etc/init.d/**

Most of Unix Distro's have /etc/init.d directory that contains scripts to start and stop daemons. In newer versions of OS it has been replaced with those under /usr/lib/systemd/ system

```
[cloud_user@ravisingh1c opt]$ ls -l /etc/init.d/  
total 44  
-rwxr-xr-x. 1 root root 2608 Sep 23 2016 awslogs  
-rw-r--r--. 1 root root 18281 Aug 24 2018 functions  
-rwxr-xr-x. 1 root root 4569 Aug 24 2018 netconsole  
-rwxr-xr-x. 1 root root 7923 Aug 24 2018 network  
-rw-r--r--. 1 root root 1160 Jul 31 08:15 README
```

- **/etc/X11/**

The graphical display (aka X Window System or just X) is driven by software from the X.org foundation. The configuration file for your graphical display is /etc/X11/xorg.conf.

# Configuration directories



- **/etc/skel/**

The skeleton directory /etc/skel is copied to the home directory of a newly created user. It usually contains hidden files like a .bashrc script.

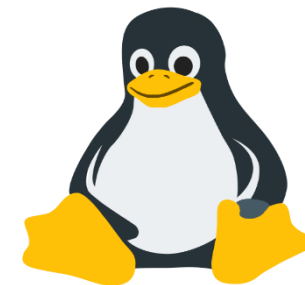
- **/etc/sysconfig/**

This directory contains most of Red Hat Enterprise Linux configuration files.

```
[cloud_user@ravisingh1c ~]$ ls /etc/sysconfig/  
authconfig  htcache-clean  man-db        readonly-root  sshd  
cbq         httpd          modules       rpcbind       vmimport.network  
chronyd     init           netconsole    rpc-rquotad    vncservers  
console     ip6tables-config  network       rsyncd         wpa_supplicant  
cpupower    iptables-config  network-scripts  rsyslog  
crond       irqbalance      nfs           run-parts  
firstboot   kdump           raid-check     samba  
grub        kernel          rdisc         selinux  
[cloud_user@ravisingh1c ~]$
```



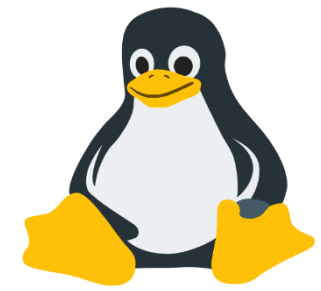
# Configuration directories



```
[cloud_user@ravisingh1c ~]$ cat /etc/sysconfig/network
# Created by cloud-init on instance boot automatically, do not edit.
#
NETWORKING=yes
NETWORKING_IPV6=yes
IPV6_AUTOCONF=no
HOSTNAME=ravisingh1c.mylabserver.com
[cloud_user@ravisingh1c ~]$
```

**The above states to enable networking on boot .**

# Data Directories



- /home

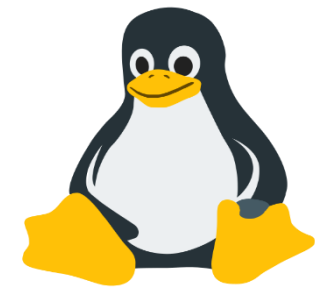
To enable users to store personal or some data linux provides a default under /home. It is common (but not mandatory) practice to name the users home directory after the user name in the format /home/ \$USERNAME.

For example:

```
[cloud_user@ravisingh1c ~]$ ls /home
cloud_user ssm-user user
[cloud_user@ravisingh1c ~]$ ls -d /home/ssm-user
/home/ssm-user
[cloud_user@ravisingh1c ~]$
```

The user directory in most organisations will not be local but stored on a central NFS share .

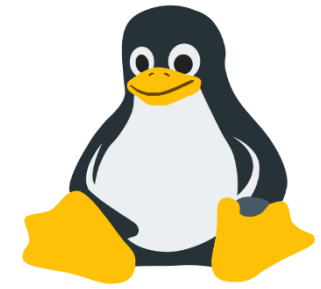
# Data Directories



Besides giving every user (or every project or group) a location to store personal files, the home directory of a user also serves as a location to store their user profile. A typical Unix user profile contains many hidden files (files whose file name starts with a dot). The hidden files of the Unix user profiles contain settings specific for that user.:

```
[cloud_user@ravisingh1c ~]$ ls -d /home/cloud_user/.*
/home/cloud_user/.          /home/cloud_user/.dbus
/home/cloud_user/..         /home/cloud_user/.esd_auth
/home/cloud_user/.bash_history /home/cloud_user/.local
/home/cloud_user/.bash_logout /home/cloud_user/.mozilla
/home/cloud_user/.bash_profile /home/cloud_user/.ssh
/home/cloud_user/.bashrc     /home/cloud_user/.viminfo
/home/cloud_user/.cache      /home/cloud_user/.vnc
/home/cloud_user/.config     /home/cloud_user/.Xauthority
[cloud_user@ravisingh1c ~]$
```

# Data Directories



- **/root**

On RHEL systems /root is the default location for personal data and profile of the root user.  
/media

The /media directory serves as a mount point for removable media devices such as CDROM's and various usb-attached devices.

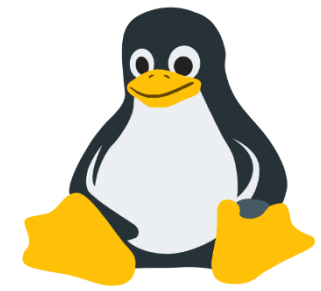
- **/mnt**

The /mnt directory should be empty and is used for temporary mount points.  
/tmp

/tmp is used by many applications and users to store temporary data when needed. Data stored in /tmp may use either disk space or RAM. Never use /tmp to store data that is important and will be lost on reboot .

**Pro Tip : Storing large files under /tmp can lead to performance issues**

# Data Directories

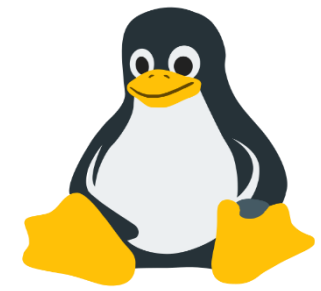


- **/proc**

Most of the files in /proc are read only, some require root privileges, some files are writable, and many files in /proc/sys are writable.

```
[cloud_user@ravisingh1c ~]$ ls /proc
1    1279 1701 1937 228 408 684    execdomains  sched_debug
10   1280 1706 1943 229 409 686    fb           schedstat
1073 1290 1710 1944 23  410 687    filesystems  scsi
1074 13   1728 1947 24  411 688    fs          self
1077 14   1750 1950 25  412 689    interrupts  slabinfo
11   1406 1759 1951 26  413 691    iomem       softirqs
1177 1417 1763 1952 27  414 7      ioports     stat
1178 1428 1766 1959 2744 415 704    irq         swaps
1179 1429 1780 1965 28  416 705    kallsyms    sys
12   1434 1787 1988 2857 417 731    kcore       sysrq-trigger
1230 1448 1794 1993 29  418 789    keys        sysvipc
1234 1453 18   2    3    46 8      key-users   timer_list
```

# Data Directories

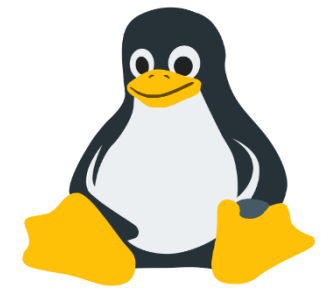


```
[cloud_user@ravisingh1c ~]$ file /proc/cpuinfo
/proc/cpuinfo: empty
[cloud_user@ravisingh1c ~]$ cat /proc/cpuinfo
processor      : 0
vendor_id     : GenuineIntel
cpu family    : 6
model         : 85
model name    : Intel(R) Xeon(R) Platinum 8175M CPU @ 2.50GHz
stepping      : 4
microcode     : 0x200005e
cpu MHz       : 2500.000
cache size    : 33792 KB
physical id   : 0
```

- **/sys**

The /sys directory was created for the Linux 2.6 kernel. Since 2.6, Linux uses sysfs to support usb and IEEE 1394 (FireWire) hot plug devices. Basically the /sys directory contains kernel information about hardware.

# Unix System Resources



- **/usr**

Although /usr is pronounced like user, it stands for Unix System Resources. The /usr hierarchy mostly contains shareable, read only data.

- **/usr/bin**

The /usr/bin directory contains all commands that requires no elevated privileges like root and can be run by normal users .

```
[cloud_user@ravisingh1c ~]$ ls /usr/bin | wc -l
1118
[cloud_user@ravisingh1c ~]$
[cloud_user@ravisingh1c ~]$ ls -ld /bin
lrwxrwxrwx. 1 root root 7 May 10 2018 /bin -> usr/bin
[cloud_user@ravisingh1c ~]$
/bin is also linked to /usr/bin
```

# Unix System Resources



- **/usr/lib**

The /usr/lib directory contains libraries that are not directly executed by users or scripts.

- **/usr/local**

The /usr/local directory can be used by an administrator to install software locally.

- **/usr/share**

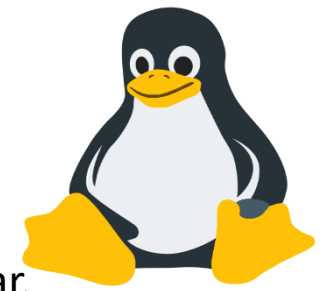
The /usr/share directory contains architecture independent data. As you can see, this is a fairly large directory.

```
[root@ravisingh1c ~]# du -sh  
/usr/share  
699M  /usr/share  
[root@ravisingh1c ~]#
```

**Pro tip : Manual page ( man ) are stored under /usr/share/man**



# Unix System Resources



- **/var variable data**

Files that are unpredictable in size, such as log, cache and spool files, should be located in /var.

- **/var/log**

The /var/log directory serves as a central place and typically contains all log files.

```
[cloud_user@ravisingh1c ~]$ ls /var/log
amazon          btmp-20180529  dpkg.log      maillog        pm-powersave.log  spooler        wtmp}
anaconda        chrony        firewallld    maillog-20150806  ppp                spooler-20150806  xrdp.log-
20180529.gz
audit           cloud-init.log  gdm           maillog-20160416  samba              spooler-20160416  xrdp-sesman.log-
20180312.gz
{auth.log,      cron          grubby        maillog-20180312  secure             spooler-20180312  xrdp-sesman.log-
20180529.gz
auth.log        cron-20150806  grubby_prune_debug  maillog-20180529  secure,            spooler-20180529
yum.log
awslogs-agent-setup.log  cron-20160416  httpd         messages        secure}            syslog           yum.log-
20150107
awslogs.log      cron-20180312  lastlog       messages-20150806  secure-20150806  tallylog          yum.log-
20160416
```

# Unix System Resources



- **/var/log/messages**

A typical first file to check when troubleshooting on Red Hat (and derivatives) is the `/var/log/messages` file. By default this file will contain information on what just happened to the system. The file is called `/var/log/syslog` on Debian and Ubuntu

- **/var/cache**

The `/var/cache` directory can contain cache data for several applications.

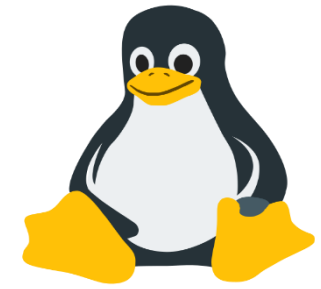
- **/var/spool**

The `/var/spool` directory typically contains spool directories for mail and cron, but also serves as a parent directory for other spool files (for example print spool files).

- **/var/lib**

The `/var/lib` directory contains application state information.

# Unix System Resources



- **/var/run**

/var also contains Process ID files in /var/run .

- **/var/tmp**

Temporary files that survive a reboot are stored under /var/

# ROOT ?

- ROOT ACCOUNT
- ROOT AS “ / ”
- ROOT HOME DIRECTORY “ /root ”

# Absolute and Relative paths



- One of the core understanding one should be aware of absolute and relative paths in the file tree. When you type a path starting with a slash (/), then the root of the file tree is assumed. If you don't start your path with a slash, then the current directory is the assumed starting point.
- The screenshot below first shows the current directory //home/cloud\_user. From within this directory, you have to type cd /home instead of cd home to go to the /home directory.
- While you are under a path to go to the next level you need to type cd followed by the directory instead of cd /

```
[cloud_user@ravisingh1c ~]$ pwd
/home/cloud_user
[cloud_user@ravisingh1c ~]$ cd home
-bash: cd: home: No such file or
directory
[cloud_user@ravisingh1c ~]$ cd /home
[cloud_user@ravisingh1c home]$ pwd
/home
[cloud_user@ravisingh1c home]$
```

```
[cloud_user@ravisingh1c home]$ pwd
/home
[cloud_user@ravisingh1c home]$ cd
/cloud_user
-bash: cd: /cloud_user: No such file or
directory
[cloud_user@ravisingh1c home]$ cd
cloud_user
[cloud_user@ravisingh1c ~]$
```

# Path Completion



- The tab key can help you in auto completion of commands without the need for typing the whole command or path without errors.

Typing `cd /ho` followed by the tab key will expand the command line to `cd /home/`.

When typing `cd /Ho` followed by the tab key, nothing will happen because you typed the wrong path (upper case H).

**Pro Tip :** You will need fewer key strokes when using the tab key, and you will be sure your typed path is correct!

LAB : PRACTISE RELATIVE AND  
ABSOLUTE PATHS ..

# FILE SYSTEM ATTRIBUTES

permission modes	# links	owner	group	size (bytes)	date (modified)		file name
↓	↓	↓	↓	↓	↓	↓	↓
drwxr-xr-x	2	root	root	4096	Mar 21	2002	bin
drwxr-xr-x	17	root	root	77824	Aug 11	14:40	dev
drwxr-xr-x	69	root	root	8192	Sep 25	18:15	etc
drwxr-xr-x	66	root	root	4096	Sep 25	18:15	home
dr-xr-xr-x	46	root	root	0	Aug 11	10:39	proc
drwxr-x---	12	root	root	4096	Aug 7	2002	root
drwxr-xr-x	2	root	root	8192	Mar 21	2002	sbin
drwxrwxrwx	6	root	root	4096	Sep 29	04:02	tmp
drwxr-xr-x	16	root	root	4096	Mar 21	2002	usr
-rw-r--r--	1	root	root	802068	Sep 6	2001	vmlinuz



# Mkdir



```
[cloud_user@ravisingh1c test]$ mkdir test1
[cloud_user@ravisingh1c test]$ mkdir test2
[cloud_user@ravisingh1c test]$ ls -l
total 0
drwxrwxr-x. 2 cloud_user cloud_user 6 Aug 31
20:03 test1
drwxrwxr-x. 2 cloud_user cloud_user 6 Aug 31
20:03 test2
[cloud_user@ravisingh1c test]$
```

- When we need to create a directory structure where the parent point does not exist mkdir -p comes in handy

Eg

The below command will fail

# Mkdir



```
$ mkdir test3/test4/test5
mkdir: cannot create directory 'test3/test4/test5': No such file or directory
[cloud_user@ravisingh1c test]$
Now if we use -p
[cloud_user@ravisingh1c test]$ mkdir -p test3/test4/test5
[cloud_user@ravisingh1c test]$ ls -ltr
total 0
drwxrwxr-x. 2 cloud_user cloud_user 6 Aug 31 20:03 test1
drwxrwxr-x. 2 cloud_user cloud_user 6 Aug 31 20:03 test2
drwxrwxr-x. 3 cloud_user cloud_user 18 Aug 31 20:07 test3
[cloud_user@ravisingh1c test]$ ls -ltr test3
total 0
drwxrwxr-x. 3 cloud_user cloud_user 18 Aug 31 20:07 test4
[cloud_user@ravisingh1c test]$ ls -ltr test3/test4
total 0
drwxrwxr-x. 2 cloud_user cloud_user 6 Aug 31 20:07 test5
[cloud_user@ravisingh1c test]$
```

# rmdir



- Rmdir

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

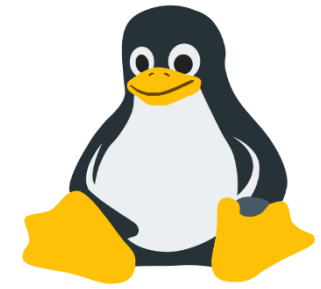
```
[cloud_user@ravisingh1c test]$ rmdir test3
rmdir: failed to remove 'test3': Directory not empty
[cloud_user@ravisingh1c test]$
[cloud_user@ravisingh1c test]$ rmdir test2
[cloud_user@ravisingh1c test]$
```

- rmdir -p

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

```
[cloud_user@ravisingh1c test]$ rmdir -p test3/test4/test5
[cloud_user@ravisingh1c test]$ ls -l
total 0
drwxrwxr-x. 2 cloud_user cloud_user 6 Aug 31 20:03 test1
[cloud_user@ravisingh1c test]$
```

# touch



```
[root@ravisingh1c ~]# file /proc/cpuinfo
/proc/cpuinfo: empty
[root@ravisingh1c ~]# file -s /proc/cpuinfo
/proc/cpuinfo: ASCII text, with very long lines
[root@ravisingh1c ~]#
```

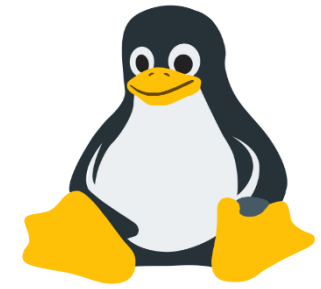
- **Touch**

To create an empty file

One easy way to create an empty file is with touch.

```
[cloud_user@ravisingh1c test]$ touch a
[cloud_user@ravisingh1c test]$ touch b
[cloud_user@ravisingh1c test]$ ls -l
total 0
-rw-rw-r--. 1 cloud_user cloud_user 0 Aug 31 20:25 a
-rw-rw-r--. 1 cloud_user cloud_user 0 Aug 31 20:25 b
drwxrwxr-x. 2 cloud_user cloud_user 6 Aug 31 20:03 test1
[cloud_user@ravisingh1c test]$
```

# CP



## Copy one file

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

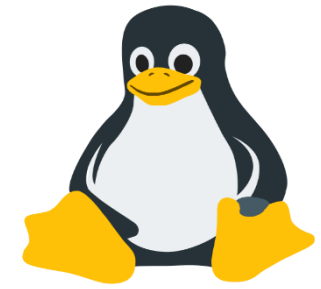
```
[cloud_user@ravisingh1c ~]$ touch a
[cloud_user@ravisingh1c ~]$ cp a b
[cloud_user@ravisingh1c ~]$ ls -ltr
total 4
drwxr-xr-x. 2 cloud_user cloud_user  6 May  9 15:58 Desktop
-rw-r--r--. 1 cloud_user cloud_user 1297 Aug 27 21:12 my-login.pp
-rw-rw-r--. 1 cloud_user cloud_user   0 Aug 31 20:32 a
-rw-rw-r--. 1 cloud_user cloud_user   0 Aug 31 20:32 b
[cloud_user@ravisingh1c ~]$
```

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

**cp -R**

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

# CP

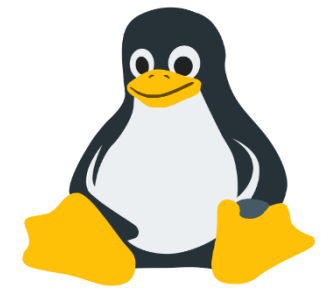


Copy to another directory

```
[cloud_user@ravisingh1c ~]$ cp a c
[cloud_user@ravisingh1c ~]$ ls -ltr
total 4
drwxr-xr-x. 2 cloud_user cloud_user  6 May  9 15:58 Desktop
-rw-r--r--. 1 cloud_user cloud_user 1297 Aug 27 21:12 my-login.pp
-rw-rw-r--. 1 cloud_user cloud_user   0 Aug 31 20:32 a
-rw-rw-r--. 1 cloud_user cloud_user   0 Aug 31 20:32 b
drwxrwxr-x. 2 cloud_user cloud_user 14 Aug 31 20:33 c
[cloud_user@ravisingh1c ~]$ cd c
[cloud_user@ravisingh1c c]$ ls -l
total 0
-rw-rw-r--. 1 cloud_user cloud_user 0 Aug 31 20:33 a
[cloud_user@ravisingh1c c]$
```

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

# CP



```
[cloud_user@ravisingh1c ~]$ cp c d
cp: omitting directory 'c'
[cloud_user@ravisingh1c ~]$ cp -r c d
[cloud_user@ravisingh1c ~]$
```

- **Copy multiple files to directory**

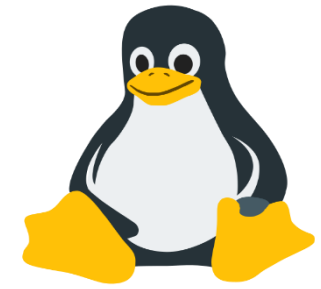
You can also use cp to copy multiple files into a directory. In this case, the last argument (a.k.a. the target) must be a directory.

```
[cloud_user@ravisingh1c ~]$ cp a b d
[cloud_user@ravisingh1c ~]$ ls -ltr d
total 0
-rw-rw-r--. 1 cloud_user cloud_user 0 Aug 31 20:37 a
-rw-rw-r--. 1 cloud_user cloud_user 0 Aug 31 20:37 b
[cloud_user@ravisingh1c ~]$
```

- **Cp -i**

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

```
[cloud_user@ravisingh1c ~]$ cp -i a b
cp: overwrite 'b'? n
[cloud_user@ravisingh1c ~]$
```



- mv

rename files with mv

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

```
[cloud_user@ravisingh1c ~]$ mv a test
[cloud_user@ravisingh1c ~]$ ls -l
total 4
-rw-rw-r--. 1 cloud_user cloud_user  0 Aug 31 20:32 b
drwxrwxr-x. 2 cloud_user cloud_user 14 Aug 31 20:33 c
drwxrwxr-x. 2 cloud_user cloud_user 22 Aug 31 20:37 d
drwxr-xr-x. 2 cloud_user cloud_user  6 May  9 15:58 Desktop
-rw-r--r--. 1 cloud_user cloud_user 1297 Aug 27 21:12 my-login.pp
-rw-rw-r--. 1 cloud_user cloud_user  0 Aug 31 20:32 test
[cloud_user@ravisingh1c ~]$
```

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

rename directories with mv

The same mv command can be used to rename directories



Lab : ..

to practise mkdir , cp ,  
touch , vi

# LAB : Test Your Knowledge



- **Practice**

1. Display your current directory.
2. Change to the /etc directory.
3. Now change to your home directory using only three key presses.
4. Change to the /boot/grub directory using only one key presses.
5. Go to the parent directory of the current directory.
6. Go to the root directory.
7. List the contents of the root directory.
8. List a long listing of the root directory.
9. Stay where you are, and list the contents of /etc
10. Stay where you are , and list the contents of /bin and /sbin.
11. Stay where you are, and list the contents of ~.
12. List all the files (including hidden files) in your home directory.
13. List the files in /boot in a human readable format.
14. Create a directory testdir in your home directory.
15. Change to the /etc directory, stay here and create a directory newdir in your home directory.
16. Create in one command the directories ~/dir1/dir2/dir3 (dir3 is a subdirectory from dir2, and dir2 is a subdirectory from dir1 ).

# Files in Linux



Linux stores data and programs in files. These are organised on directory. In simple terms a directory is also a file that contains other files ( or directories ) . Files in Linux ( or for that matter any unix ) are case sensitive which would mean uppercase and lowercase unlike windows matters

```
[cloud_user@ravisingh1c test]$ ls -ld test1
drwxrwxr-x. 2 cloud_user cloud_user 6 Aug 31
20:03 test1
[cloud_user@ravisingh1c test]$ ls -ld Test1
ls: cannot access Test1: No such file or
directory
[cloud_user@ravisingh1c test]$
```

Everything is a file

A directory is a special kind of file, but it is still a (case sensitive!) file. It will become clear throughout this course that everything on Linux is a file.

# Files in Linux



- **File**

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

```
[cloud_user@ravisingh1c test]$ file /etc/hosts
/etc/hosts: ASCII text
[cloud_user@ravisingh1c test]$

[cloud_user@ravisingh1c test]$ file /bin/ls
/bin/ls: ELF 64-bit LSB executable, x86-64, version 1 (SYSV),
dynamically linked (uses shared libs), for GNU/Linux 2.6.32,
BuildID[sha1]=ceaf496f3aec08afced234f4f36330d3d13a657b,
stripped
[cloud_user@ravisingh1c test]$
```

**Pro Tip :** The file command uses a magic file that contains patterns to recognise file types. The magic file is located in /usr/share/file/magic. Type man 5 magic for more information. It is interesting to point out file -s for special files like those in /dev and /proc.

# LINUX FILE TYPES

-rw-----

Regular File

**d**rwxr-xr-x.

Directory File

**l**rwxrwxrwx.

Link File

**c**rw-rw----.

Character Device File

**b**rw-rw----.

Block Special File

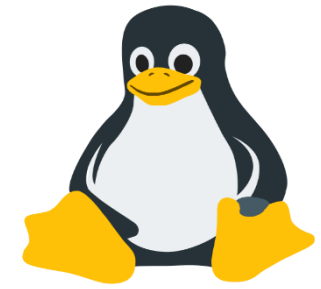
**s**rw-rw-rw-

Socket File

**p**rw-----.

Named Pipe File

# Common Unix tools



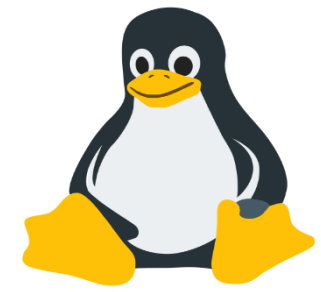
This module introduces few commonly used commands to find or locate files.

- **Find**

The find command can be very useful at the start of a pipe to search for files. Here are some examples. You might want to add 2>/dev/null to the command lines to avoid cluttering your screen with error messages.

```
find / -name "ifcfg-enp0s3"
```

# Common Unix tools

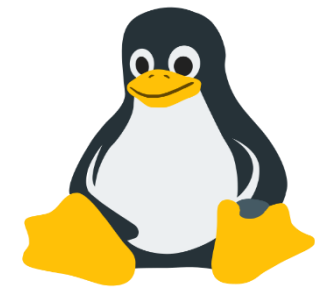


```
locate ifcfg-enp0s3
```

- **Locate**

The locate tool is very different from find in that it uses an index to locate files. This is a lot faster than traversing all the directories, but it also means that it is always outdated. If the index does not exist yet, then you have to create it (as root on Red Hat Enterprise Linux) with the updatedb command. You will need to install mlocate binary if not installed .

# Common Unix tools



```
[root@ravisingh1c ~]# locate
locate: no pattern to search for specified
[root@ravisingh1c ~]# locate httpd ( not indexed hence did not find )
[root@ravisingh1c ~]# updated ( update the database )
```

```
[root@ravisingh1c ~]# locate httpd
/etc/httpd
/etc/httpd/conf
/etc/httpd/conf.d
/etc/httpd/conf.modules.d
/etc/httpd/logs
/etc/httpd/modules
```



# Difference between find and locate

Find	Locate
Searches in whole system and find the specified files or directories.	Searches in database which is already created in system.
Find command has number of options and is very configurable.	Locate has very few options.
We can find files based on name, size, age, permission, user, groups etc	We can find files based on name and permission only.
There are many ways to reduce the depth and breadth of your search and make it more efficient.	While locate cannot be reduced to make it efficient.
Find is more accurate as it search in real time system.	Locate is less accurate as it searches from the existing database because if database is not updated then locate command will find any match. To sync the database, it is must to execute the <b>"updatedb"</b> command.

# LAB : Locate and find

- Create a new file susan in Desktop directory under home directory
- Go to / directory and try to search susan using find command .
- Now try to find susan using locate command
- Did you find it using locate command ?
- Now try to update the database of locate using updatedb
- Now again try to search susan file using locate command
- Did you find it ?

# Rm

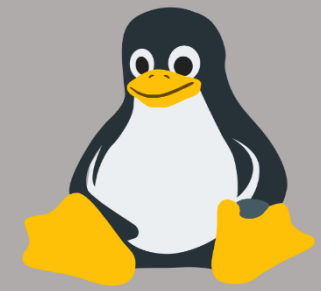


- **Remove forever**

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

```
[cloud_user@ravisingh1c test]$ rm a
[cloud_user@ravisingh1c test]$ ls -ltr
total 0
drwxrwxr-x. 2 cloud_user cloud_user 6 Aug 31 20:03 test1
-rw-rw-r--. 1 cloud_user cloud_user 0 Aug 31 20:25 b
[cloud_user@ravisingh1c test]$
```

# Rm



- Rm -i

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

```
[cloud_user@ravisingh1c test]$ rm -i b
rm: remove regular empty file 'b'? yes
[cloud_user@ravisingh1c test]$ touch b
[cloud_user@ravisingh1c test]$ ls -l
total 0
-rw-rw-r--. 1 cloud_user cloud_user 0 Aug 31 20:29 b
drwxrwxr-x. 2 cloud_user cloud_user 6 Aug 31 20:03 test1
[cloud_user@ravisingh1c test]$ rm -i b
rm: remove regular empty file 'b'? no
[cloud_user@ravisingh1c test]$ ls -l
total 0
-rw-rw-r--. 1 cloud_user cloud_user 0 Aug 31 20:29 b
drwxrwxr-x. 2 cloud_user cloud_user 6 Aug 31 20:03 test1
[cloud_user@ravisingh1c test]$
```

# Rm



- **Rm -rf**

By default, `rm -r` will not remove non-empty directories. However `rm` accepts several options that will allow you to remove any directory. The `rm -rf` statement is famous because it will erase anything (providing that you have the permissions to do so).

```
[cloud_user@ravisingh1c ~]$ rm test
rm: cannot remove 'test': Is a directory
[cloud_user@ravisingh1c ~]$ rm -rf test
[cloud_user@ravisingh1c ~]$ ls test
ls: cannot access test: No such file or directory
[cloud_user@ravisingh1c ~]$
```

**Pro Tip :** When you are logged on as root, be very careful with `rm -rf` (the `f` means force and the `r` means recursive) since being root implies that permissions don't apply to you. You can literally erase your entire file system by accident.

# WILDCARDS

- Wildcards replace characters
  - \* replaces zero, one or more characters
  - ? replaces one character
  - [ ] match one of the choices inside brackets
  - { } match each of the choices inside braces

# LAB : WILDCARDS

- Create 10 files using { } wildcard of format abcd1-xyz ,abcd2-xyz ,,, and so on
- Use ? Wildcard to list the file starting with any character and ending with bcd5-xyz
- Use \* to list the files starting with abc and ending with anything
- Use \*[ ]\* wildcard to list files containing bcd in it



# HARD LINK vs SOFT LINK/SYMBOLIC LINK



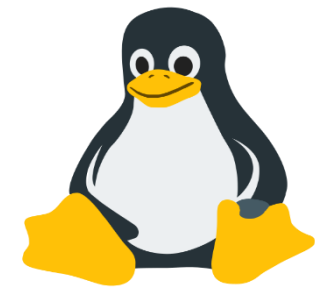


LAB : SOFT AND HARD LINKS ..

# PRACTISE LAB

- Change your password
- Create 10 files under your home directory (File names = jerry, kramer, george, lex, clark, lois, homer, bart, lisa, and marge)
- Create 3 directories under your home directory (Dir name = seinfeld, superman and simpsons)
- Create a new file jupiter and write to it as "Jupiter is a planet". Then create a soft link in /tmp directory
- Also create a hard link of jupiter in /tmp directory
- Check the inodes of both links

# File Permissions



- **Standard file ownership**

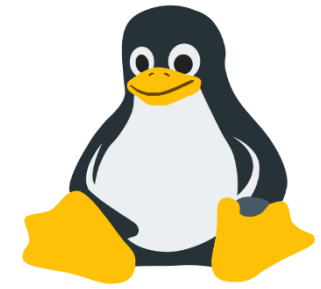
The users and groups of a system can be locally managed in `/etc/passwd` and `/etc/group`. These users and groups can own files. Actually, every file has a user owner and a group owner, as can be seen in the following screenshot.

```
[cloud_user@ravisingh1c ~]$ ls -lh
-rw-rw-r--. 1 cloud_user cloud_user  11 Sep  3 07:47 test1
-rw-rw-r--. 1 cloud_user wheel      40 Sep  3 08:33 test.txt
[cloud_user@ravisingh1c ~]$ ls -lh
```

File `test1` is owned by user `cloud_user` and group `cloud_user` and the file `test.txt` is owned by user `cloud_user` and group `wheel`.

**Pro Tip:** Only the owner of the file and root is allowed to change file permissions unless the group owner also has read and write permissions

# File Permissions



First character	File type
-	Normal file
d	Directory
l	Symbolic link
p	Named pipe
b	Block device
c	Character device
s	Socket



User



Read  
Write  
Execute



Group



Read  
Write  
Execute



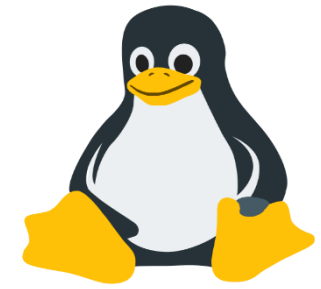
All



Read  
Write  
Execute



# File Permissions



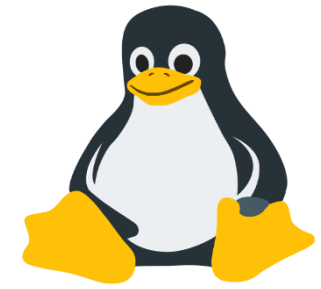
- **Entities**

**Owner** Each file and directory (a special type of file) has an "owner." This is the user account that has primary power over the file, allowing it to do things like change the file's permissions. The owner is expressed as a user account such as root or your personal user account, or even some "user" account automatically created for the use of a piece of software you have installed. Normally, the owner of a file is the account used to create the file, though files can be reassigned to a different owner later on using the `chown` command.

**Group** In addition to the owner, each file has a group account associated with it. This group, like the user account that is the file's owner, has its own set of access permissions to the file. When creating a file, the group is set to the default group of the user account used to create the file though files can be reassigned to a different group later on using the `chgrp` command

**World or Others** The last permissions category covers "everyone else" — any accounts that are not the owner or a member of the account group that is the file's group owner.

# File Permissions



- **File-level security management**

If you use a Unix OS, or Unix-like OS, you should — as a bare minimum — understand this most basic form of file-level security management. Failing to understand file permission management might cause problems one day. The fundamental key to basic file-level security on a Unix system is to keep file permissions as restrictive as possible without stopping the system from doing what it needs to do, and without preventing yourself from accessing the files the way you need to access them.

For the most part, file permissions outside of user account home directories — such as `/usr/home/jon/` in the case of the above hypothetical jon account or `/root/` in the case of the root account on a FreeBSD system, or `/home/jon/` and `/root/` respectively on a typical Linux-based system — should be left to defaults unless you know exactly what you are doing.

Most data files in a user account's home directory, such as text files, word processor files, and so on, should have `110/6/rw-` permissions for that account and `000/0/---` for group and world permissions, while most subdirectories within that home directory should have `111/7/rwx` for the owner and `000/0/---` for the group and world permissions, especially if you are concerned about privacy protection against other accounts or malicious security crackers that may compromise or create other accounts on the system.

# File Permissions



**Pro Tip:** Learn how to use Unix file permissions to best effect, and you will know the first thing to know about Unix security. Without secure use of file permissions, the strong privilege separation of a Unix operating system that provides it such a significant security advantage over many other OSes can be undermined

- **Permissions**

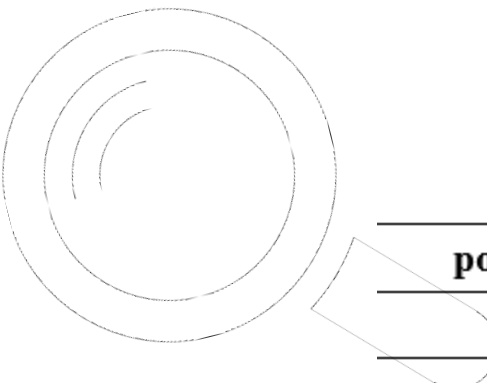
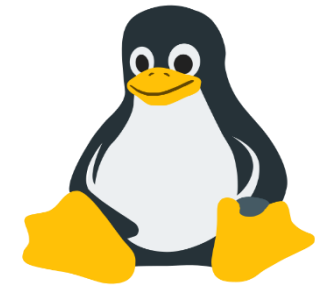
rwX

The nine characters following the file type denote the permissions in three triplets. A permission can be r for read access, w for write access, and x for execute. You need the r permission to list (ls) the contents of a directory. You need the x permission to enter (cd) a directory. You need the w permission to create files in or remove files from a directory.

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

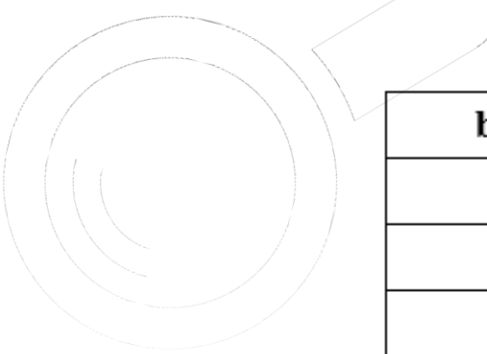


# File Permissions

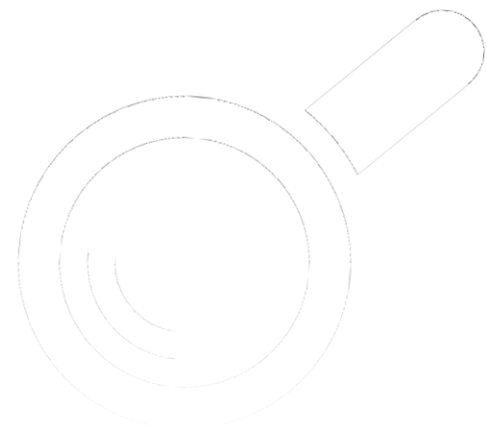
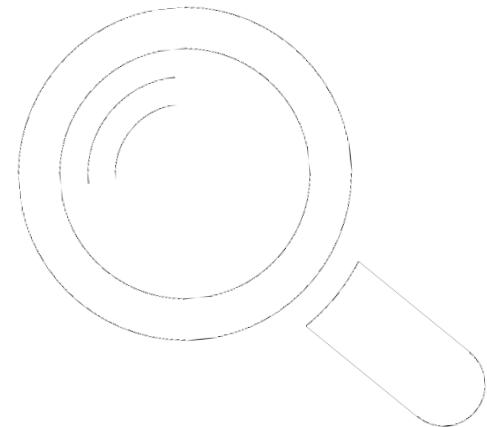


position	characters	function
1	-	this is a regular file
2-4	<u>rwX</u>	permissions for the <b>user owner</b>
5-7	r-X	permissions for the <b>group owner</b>
8-10	r--	permissions for <b>others</b>

- **Octal Permissions**



binary	octal	permission
000	0	---
001	1	--X
010	2	-w-
011	3	- <u>wX</u>
100	4	r--
101	5	r-X
110	6	<u>rw</u> -
111	7	<u>rwX</u>



# File Permissions



- **List of special files**

When you use `ls -l`, for each file you can see ten characters before the user and group owner. The first character tells us the type of file. Regular files get a `-`, directories get a `d`, symbolic links are shown with an `l`, pipes get a `p`, character devices a `c`, block devices a `b`, and sockets an `s`. Table

- **Permissions**

`rw`

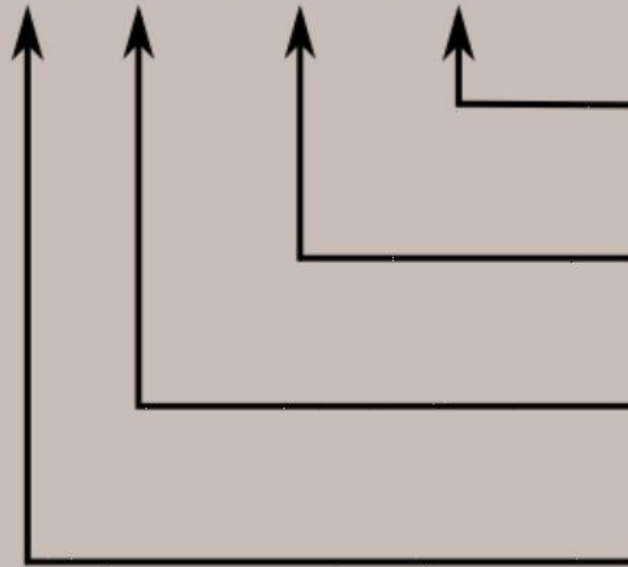
The nine characters following the file type denote the permissions in three triplets. A permission can be `r` for read access, `w` for write access, and `x` for execute. You need the `r` permission to list (`ls`) the contents of a directory. You need the `x` permission to enter (`cd`) a directory. You need the `w` permission to create files in or remove files from a directory.

`Chown` can also be used to change both user and group permissions

```
[root@ravisingh1c cloud_user]# ls -l allfiles.txt
-rw-rw-r--. 1 cloud_user cloud_user 8174638 Sep  4 18:19 allfiles.txt
[root@ravisingh1c cloud_user]# chown user2:user2 allfiles.txt
[root@ravisingh1c cloud_user]# ls -l allfiles.txt
-rw-rw-r--. 1 user2 user2 8174638 Sep  4 18:19 allfiles.txt
[root@ravisingh1c cloud_user]#
```

# Chmod Command :

- rwxrwxrwx



Read, write, and execute permissions for all other users.

Read, write, and execute permissions for the group owner of the file.

Read, write, and execute permissions for the file owner.

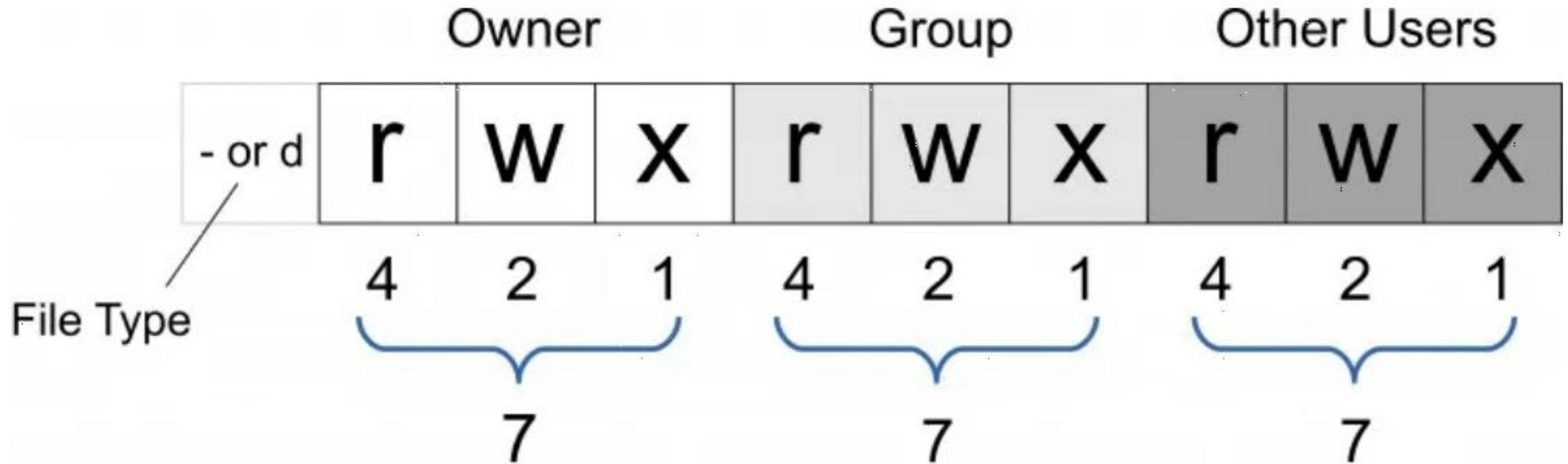
File type:  
- indicates regular file  
d indicates directory

**-rwx rwx rwx**  
u g o  
user group others

# LAB :

- Go to home user directory rkhanha being standard user.
- Create a directory named testdirectory
- List the contents and see the permissions assigned to the directory
- Try to create a file inside testdirectory .
- Change the permissions assigned to the testdirectory using chmod command; exclude read and write access from testdirectory from user owner.  
chmod u-rw testdirectory/
- Now again try to create a file inside testdirectory. Are u able to now ?
- Assign back the permissions and try to create the file now !!

# File Permissions Using Numeric Mode



# LAB : File Permissions Using Numeric Mode

- Create a file test2 in user home directory
- Change the permissions of the file using numerical method
- To change the user permissions to only read and write and execute , group permission to read and write , others to only read :

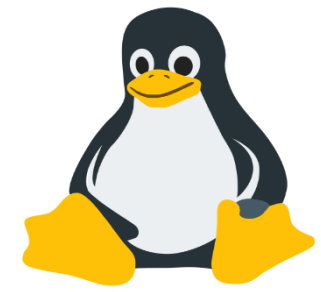
```
chmod 764 test2
```

- Now try changing user, group and others to only read access :

```
chmod 444 test2
```

\*Now try to edit the file test2 . Are u able to ?

# File Permissions



- **Chgrp**

To change group ownership

```
[cloud_user@ravisingh1c ~]$ ls -ld test.txt
-rwxr--r--. 1 cloud_user wheel 40 Sep  3 08:33 test.txt
[cloud_user@ravisingh1c ~]$ chgrp cloud_user test.txt
[cloud_user@ravisingh1c ~]$ ls -ld test.txt
-rwxr--r--. 1 cloud_user cloud_user 40 Sep  3 08:33 test.txt
[cloud_user@ravisingh1c ~]$
```

- **Chown**

To change owner of a file use chown command

```
[root@ravisingh1c cloud_user]# ls -ld d.txt.gz
-rw-rw-r--. 1 cloud_user cloud_user 842 Sep  4 18:31 d.txt.gz
[root@ravisingh1c cloud_user]# chown user2 d.txt.gz
[root@ravisingh1c cloud_user]# ls -ld d.txt.gz
-rw-rw-r--. 1 user2 cloud_user 842 Sep  4 18:31 d.txt.gz
[root@ravisingh1c cloud_user]#
```

# LAB : chown and chgrp

- \* Go to /home/rkhanna directory and create a file testfile
- List the contents and see the owner and group of testfile
- Try to add content in testfile and cat it out.
- Now become root user and change the owner and group of file testfile

chown root testfile

chgrp root testfile

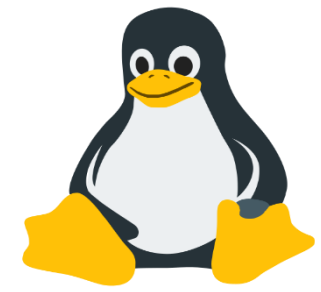
chmod o-r testfile

\*Exit out of root user now and with standard user to go to /home/rkhanna

- Now try to cat or add content and cat it, Are you able to do it ?



# File Permissions



- **Umask**

When creating a file or directory, a set of default permissions are applied. These default permissions are determined by the **umask**. The **umask** specifies permissions that you do not want set on by default. You can display the **umask** with the **umask** command.

# Working with file contents



In this module we will look at the how to view or display contents of text files with head, tail, cat, more, less and strings. We will also get to know the possibilities of tools like cat on the command line.

- head

```
head /etc/group
root:x:0:
bin:x:1:
daemon:x:2:
sys:x:3:
adm:x:4:centos
tty:x:5:
disk:x:6:
lp:x:7:
mem:x:8:
kmem:x:9:
```

You can use head to display the first ten lines of a file.  
The head command can also display the first n lines of a file.

```
head -4 /etc/group
root:x:0:
bin:x:1:
daemon:x:2:
sys:x:3:
```

# Working with file contents



- Tail

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

```
tail /etc/group
input:x:993:
systemd-bus-proxy:x:992:
systemd-network:x:991:
geoclue:x:990:
pulse-rt:x:989:
rvm:x:1002:
printadmin:x:988:
cloud_user:x:1003:
ssm-user:x:1004:
sudo:x:1005:user
```

```
tail -3 /etc/group
cloud_user:x:1003:
ssm-user:x:1004:
sudo:x:1005:user
```

# Working with file contents

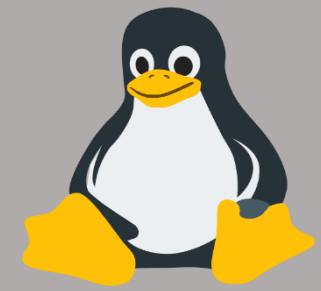


- Cat

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

```
cat /etc/sysconfig/network
# Created by cloud-init on instance boot
automatically, do not edit.
#
NETWORKING=yes
NETWORKING_IPV6=yes
IPV6_AUTOCONF=no
HOSTNAME=xxxxx.mylabserver.com
```

# Working with file contents



## Concatenate

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

```
[cloud_user@ravisingh1c ~]$ echo a >a  
[cloud_user@ravisingh1c ~]$ echo b > b  
[cloud_user@ravisingh1c ~]$ cat a  
a  
[cloud_user@ravisingh1c ~]$ cat b  
b  
[cloud_user@ravisingh1c ~]$ cat a b >e  
[cloud_user@ravisingh1c ~]$ cat e  
a  
b  
[cloud_user@ravisingh1c ~]$
```

# Working with file contents



- **Create files**

One can use cat to create flat text files.

Type the `cat > a.txt` command as shown in the screenshot below. Then type one or more lines, finishing each line with the enter key. After the last line, type and hold the Control (Ctrl) key and press d.

```
[cloud_user@ravisingh1c ~]$ cat > a.txt  
this is a test file  
[cloud_user@ravisingh1c ~]$ cat a.txt  
this is a test file  
[cloud_user@ravisingh1c ~]$
```

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

# Working with file contents



## Copy files

In the below example you will see that cat can be used to copy files instead of the cp command we saw earlier .

```
[cloud_user@ravisingh1c ~]$ cat a.txt
this is a test file
[cloud_user@ravisingh1c ~]$ cat a.txt > c.txt
[cloud_user@ravisingh1c ~]$ cat c.txt
this is a test file
[cloud_user@ravisingh1c ~]$
```

**Pro Tip** : Use the tac command to see backwards of the same file

```
[cloud_user@ravisingh1c ~]$ cat b.txt
this is a new file
this is for a test
[cloud_user@ravisingh1c ~]$ tac b.txt
this is for a test
this is a new file
```

# Working with file contents



- **More and Less**

The more command is useful for displaying files that take up more than one screen. More will allow you to see the contents of the file page by page. Use the space bar to see the next page, or q to quit. Some people prefer the less command to more.

```
[cloud_user@ravisingh1c ~]$ more /etc/group
root:x:0:
bin:x:1:
daemon:x:2:
sys:x:3:
adm:x:4:centos
tty:x:5:
disk:x:6:
lp:x:7:
mem:x:8:
kmem:x:9:
wheel:x:10:centos,cloud_user,user
cdrom:x:11:
```

```
mail:x:12:postfix
man:x:15:
dialout:x:18:
floppy:x:19:
games:x:20:
tape:x:30:
video:x:39:
ftp:x:50:
lock:x:54:
audio:x:63:
nobody:x:99:
--More--(32%)
```



# LAB : PRACTISE WORKING WITH FILE CONTENTS

# Shell expansion



This module introduces you to shell expansion by taking a close look at commands and arguments. Having a good understanding of shell expansion is important because a lot of commands on our Linux system are processed and most likely changed by the shell before they are executed.

The default shell used on most Linux systems is bash, which stands for Bourne again shell. The bash shell incorporates features from sh (the original Bourne shell), csh (the C shell), and ksh (the Korn shell).

```
[cloud_user@ravisingh1c ~]$ ls -ld /bin/sh
lrwxrwxrwx. 1 root root 4 Dec 13 2018 /bin/sh -> bash
[cloud_user@ravisingh1c ~]$
```

**Pro Tip :** **what is the default shell in Linux?? .The answer is bash**

One of the primary features of a shell is to perform a command line scan. When you enter a command at the shell's command prompt and press the enter key, then the shell will start scanning that line, cutting it up in arguments. While scanning the line, the shell may make many changes to the arguments you typed. This process is called shell expansion. When the shell has finished scanning and modifying that line, then it will be executed.

# Shell expansion



- **White space removal**

Parts that are separated by one or more consecutive white spaces (or tabs) are considered separate arguments, any white space is removed. The first argument is the command to be executed, the other arguments are given to the command. The shell effectively cuts your command into one or more arguments. Below explains why the following four different command lines are the same after shell expansion.

```
[cloud_user@ravisingh1c ~]$ echo Lion King
Lion King
[cloud_user@ravisingh1c ~]$ echo Lion  King
Lion King
[cloud_user@ravisingh1c ~]$ echo   Lion king
Lion king
[cloud_user@ravisingh1c ~]$ echo           Lion  king
Lion king
[cloud_user@ravisingh1c ~]$
```

# Shell expansion



- **Single quotes**

When we need to prevent the removal of white spaces by quoting the spaces within a ' '. The contents of the quoted string are considered as one argument. In the example below the echo receives only one argument.

- **Double quotes**

The removal of white spaces can also be achieved by double quoting the spaces. Echo only receives one argument.

```
[cloud_user@ravisingh1c ~]$ echo "Lion    King"  
Lion    King  
[cloud_user@ravisingh1c ~]$
```

# Commands



## External or builtin commands ?

Not all commands are external to the shell, some are builtin. External commands are programs that have their own binary and reside somewhere in the file system. Many external commands are located in /bin or /sbin. Builtin commands are an integral part of the shell program itself.

type

To find out whether a command given to the shell will be executed as an external command or as a builtin command, use the **type command**.

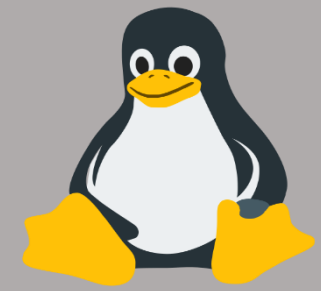
```
[cloud_user@ravisingh1c ~]$ type ls
ls is aliased to `ls --color=auto'
[cloud_user@ravisingh1c ~]$ type echo
echo is a shell builtin
[cloud_user@ravisingh1c ~]$
```

You can see echo is a builtin and cat is an external command

## Which

The which command will search for binaries in the \$PATH environment variable (variables will be explained later). In the screenshot below, it is determined that cd is builtin, and ls, cp, rm, mv, mkdir, pwd, and which are external commands.

# Commands



```
[cloud_user@ravisingh1c ~]$ which ls
alias ls='ls --color=auto'
/usr/bin/ls
[cloud_user@ravisingh1c ~]$ which httpd
/usr/sbin/httpd
[cloud_user@ravisingh1c ~]$

[cloud_user@ravisingh1c ~]$ which dc
/usr/bin/which: no dc in (/usr/local/rvm/gems/ruby-2.4.1/bin:/usr/local/rvm/gems/ruby-2.4.1@global/bin:/usr/local/rvm/rubies/ruby-2.4.1/bin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/usr/local/rvm/bin:/home/cloud_user/.local/bin:/home/cloud_user/bin)
[cloud_user@ravisingh1c ~]$
```

We can see as dc command is not found in current path or not there it display no dc in and then displays the search path .

**Pro Tip : If we know a command exists but does not find it we need to append to PATH variable**

# Aliases



- **Alias**

Aliases are often used to create an easier to remember name for an existing command or to easily supply parameters.

```
[alias raman='ls -ltr'
```

```
unalias raman
```

We see above we created a alias for ls to be ll and if we type ll or ls the output is same .This really becomes more helpful when you have to pass a long path or don't want to remember and can be said to be shortcuts

# Aliases



To display all alias set for user

```
[cloud_user@ravisingh1c ~]$ alias
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l.='ls -d .* --color=auto'
alias ll='ls'
alias rvm-restart='rvm_reload_flag=1 source \"/usr/local/rvm/scripts/rvm\"'
alias vi='vim'
alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot --show-tilde'
[cloud_user@ravisingh1c ~]$
```



# Aliases



- **Unalias**

You can undo an alias with the unalias command.

```
alias raman='ls -ltr'
```

```
unalias raman
```

# Control operators



So far we have seen run of a single command but there might be situations where we need to repeat commands on multiple systems. We can use control operators when we need put more than one command on the command line using control operators.

## **; semicolon**

You can put two or more commands on the same line separated by a semicolon ; . The shell will scan the line until it reaches the semicolon. All the arguments before this semicolon will be considered a separate command from all the arguments after the semicolon. Both series will be executed sequentially with the shell waiting for each command to finish before starting the next one.

```
[cloud_user@ravisingh1c ~]$ echo a
a
[cloud_user@ravisingh1c ~]$ echo b
b
[cloud_user@ravisingh1c ~]$ echo a; echo b
a
b
[cloud_user@ravisingh1c ~]$
```

# Shell Variables



## \$ dollar sign

The shell looks at \$ and will replace the value of the variable if it exists. Common ones are \$USER, \$HOSTNAME, \$SHELL, \$HOME

```
[cloud_user@ravisingh1c ~]$ echo This is the default shell for RHEL $SHELL  
This is the default shell for RHEL /bin/bash  
[cloud_user@ravisingh1c ~]$
```

**Pro Tip: The shell variables are case sensitive**

```
[cloud_user@ravisingh1c ~]$ echo $UID  
1002  
[cloud_user@ravisingh1c ~]$ echo $uid  
  
[cloud_user@ravisingh1c ~]$
```

# Define variables



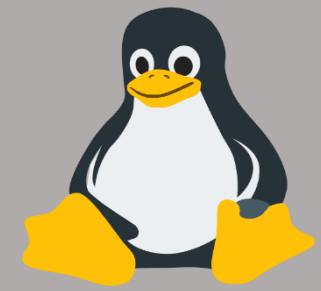
We can define our own variables and use it

```
[cloud_user@ravisingh1c ~]$ testvar=cat
[cloud_user@ravisingh1c ~]$ echo $testvar
cat
[cloud_user@ravisingh1c ~]$ echo Value of var is $testvar
Value of var is cat
[cloud_user@ravisingh1c ~]$
```

## **\$ PATH**

This \$PATH determines as earlier stated locations which are searched by the shell for commands unless they are builtin or has an alias .

# Define variables



To find the current search path

```
[cloud_user@ravisingh1c ~]$ echo $PATH
/usr/local/rvm/gems/ruby-2.4.1/bin:/usr/local/rvm/gems/ruby-
2.4.1@global/bin:/usr/local/rvm/rubies/ruby-
2.4.1/bin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/usr/local/rvm/bin:/home/cloud_user
/.local/bin:/home/cloud_user/bin
[cloud_usebnr@ravisingh1c ~]$
```

# Define variables



**env**

To display all exported variables we can use env

```
[cloud_user@ravisingh1c ~]$ env
XDG_SESSION_ID=2
rvm_bin_path=/usr/local/rvm/bin
HOSTNAME=ravisingh1c.mylabserver.com
SELINUX_ROLE_REQUESTED=
GEM_HOME=/usr/local/rvm/gems/ruby-2.4.1
TERM=xterm
SHELL=/bin/bash
HISTSIZE=100000
IRBRC=/usr/local/rvm/rubies/ruby-2.4.1/.irbrc
SSH_CLIENT=92.233.63.67 52410 22
SELINUX_USE_CURRENT_RANGE=
OLDPWD=/opt
MY_RUBY_HOME=/usr/local/rvm/rubies/ruby-2.4.1
SSH_TTY=/dev/pts/0
USER=cloud_user
```

# Shell history



The shell makes it easy for us to repeat commands, this chapter explains how.

repeating the last command

To repeat the last command in bash, type !!.

```
[cloud_user@ravisingh1c ~]$ echo a  
a  
[cloud_user@ravisingh1c ~]$ !!  
echo a  
a  
[cloud_user@ravisingh1c ~]$
```

repeating other commands

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

# Shell history



```
[cloud_user@ravisingh1c ~]$ echo a
a
[cloud_user@ravisingh1c ~]$ cat a
cat: a: No such file or directory
[cloud_user@ravisingh1c ~]$ !ca
cat a
cat: a: No such file or directory
[cloud_user@ravisingh1c ~]$ !ec
echo a
a
[cloud_user@ravisingh1c ~]$
```

- **History**

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



# Shell history



```
[cloud_user@ravisingh1c ~]$ history 5
309 env
310 echo a
311 cat a
312 echo a
313 history 5
[cloud_user@ravisingh1c ~]$
```

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

```
[cloud_user@ravisingh1c ~]$ !312
echo a
a
[cloud_user@ravisingh1c ~]$
```

Another option is to use ctrl-r to search in the history. In the screenshot below i only typed ctrl-r followed by four characters apti and it finds the last command containing these four consecutive characters.  
(reverse-i-search) `host`: file /etc/hosts

# Access control lists

ACLs allow us to apply a more specific set of permissions to a file or directory without (necessarily) changing the base ownership and permissions. They let us "tack on" access for other users or groups.

## Access Control List using ls

① `rw-rw-r-- file`

The permission does not contain a "+" character, no ACLs are defined for the file

② `rw-rw-r-- + file`

The permission contain a "+" character, ACLs are defined for this file

# COMMANDS USED :

\*To see the permissions assigned to file.

`getfacl testfile` (testfile is the file already created)

\*To set permissions for a particular user apart from the one selected as owner using `chmod`

`setfacl -m u:rkhanna:rwX /home/rkhanna/testfile`

\*To set permissions for a particular group apart from the one selected as group using `chmod`

`setfacl -m g:rkhanna:rwX /home/rkhanna/testfile`

\*To remove all ACLs attached with a file.

`setfacl -b /home/rkhanna/testfile`

# LAB :

- Practise all the commands used for ACL in the testfile created in / as root user to add standard user to its permissions.

# Pipes and commands



- **I/O redirection**

One of the powers of the Unix command line is the use of input/output redirection and pipes. This chapter explains redirection of input, output and error streams

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

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

# Pipes and commands



## output redirection

- stdout

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

```
[cloud_user@ravisingh1c ~]$ echo test  
test  
[cloud_user@ravisingh1c ~]$  
  
[cloud_user@ravisingh1c ~]$ echo test >test1  
[cloud_user@ravisingh1c ~]$ cat test1  
test
```

The > notation is in fact the abbreviation of 1>

# Pipes and commands



- >> append

Use >> to append output to a file

```
[cloud_user@ravisingh1c ~]$ cat test1
[cloud_user@ravisingh1c ~]$ echo test >test1
[cloud_user@ravisingh1c ~]$ cat test1
test
[cloud_user@ravisingh1c ~]$ echo test1 >test1
[cloud_user@ravisingh1c ~]$ cat test1
test1
[cloud_user@ravisingh1c ~]$ echo test >>test1
[cloud_user@ravisingh1c ~]$ cat test1
test1
test
```



# Error Redirection



- **2> stderr**

Redirection of stderr is done with 2> .Useful when you want to prevent messages from filling your screen

Eg

```
[rkhanha@MyFirstLinuxVM /]$ find network 2>/dev/null
[rkhanha@MyFirstLinuxVM /]$ find / -name "network"
2>/dev/null
/etc/sysconfig/network
/etc/rc.d/init.d/network
/etc/vmware-tools/scripts/vmware/network
[rkhanha@MyFirstLinuxVM /]$
```

# Error Redirection



**Without redirection**

```
[rkhanna@MyFirstLinuxVM /]$ find /etc -name "network"
find: '/etc/pki/CA/private': Permission denied
find: '/etc/pki/rsyslog': Permission denied
find: '/etc/grub.d': Permission denied
find: '/etc/selinux/targeted/active': Permission denied
find: '/etc/selinux/final': Permission denied
/etc/sysconfig/network
find: '/etc/dhcp': Permission denied
find: '/etc/lvm/archive': Permission denied
find: '/etc/lvm/backup': Permission denied
find: '/etc/lvm/cache': Permission denied
```

# Error Redirection



- **Quick File Clear**

To clear a file or null the contents

```
> testfile
```

**Pro Tip:** The above is a great way to null a file of its contents and is useful than removing a file if an active process is using it .

# Filters



- **Grep**

The grep filter is one of the widely used filters by Unix users. The most common use of grep is to filter lines of text containing (or not containing) a certain string.

```
[root@MyFirstLinuxVM home]# cat test.txt |  
grep live  
myname is ramann i live in India  
[root@MyFirstLinuxVM home]#
```

# Filters



One of the most useful options of grep is grep -i which filters in a case insensitive way.

```
[cloud_user@ravisingh1c ~]$ grep my test.txt  
[cloud_user@ravisingh1c ~]$ grep -i my test.txt  
My name is xyz  
[root@MyFirstLinuxVM etc]# grep -i net group  
systemd-network:x:192:
```

# Filters



Commands that are created to be used with a pipe are often called filters. These filters are very small programs that do one specific thing very efficiently. They can be used as building blocks.

- **Tee**

A lot of times when we execute commands or long programs we may want intermediate results. This is where tee comes in handy. The tee filter puts stdin on stdout and also into a file.

```
[cloud_user@ravisingh1c ~]$ cat test.txt |  
grep My  
My name is xyz  
My friends name is yzx  
[cloud_user@ravisingh1c ~]$
```

# Filters



- Tr

To translate characters we can use tr . Below example will convert all instance of lowercase e to uppercase E in a file

```
[cloud_user@ravisingh1c ~]$ cat c.txt  
this is a test file  
[cloud_user@ravisingh1c ~]$ cat c.txt | tr 't' 'T'  
This is a Test file  
[cloud_user@ravisingh1c ~]$
```

To convert all text

```
[cloud_user@ravisingh1c ~]$ cat c.txt | tr 'a-z' 'A-Z'  
THIS IS A TEST FILE  
[cloud_user@ravisingh1c ~]$
```

# Filters



- **wc**

Wc is used to count words, lines and characters and is widely used command and great asset for admins .

```
[cloud_user@ravisingh1c ~]$ wc c.txt
1 5 20 c.txt
[cloud_user@ravisingh1c ~]$ wc -l c.txt
1 c.txt
[cloud_user@ravisingh1c ~]$ cat c.txt | wc -l
1
[cloud_user@ravisingh1c ~]$
[cloud_user@ravisingh1c ~]$ wc -c c.txt
20 c.txt
[cloud_user@ravisingh1c ~]$ wc -w c.txt
```



# Filters



- **sort**

The sort filter will default to an alphabetical sort.

```
[cloud_user@ravisingh1c ~]$ cat d.txt  
one  
two  
three  
1  
2  
3  
10  
8
```

```
[cloud_user@ravisingh1c ~]$ cat d.txt | sort  
1  
10  
2  
3  
8  
one  
three  
two
```

# Filters



- **Uniq**

uniq is used to remove duplicates from a sorted list.

The below example has couple of duplicates and we will apply uniq to remove duplicates

```
[cloud_user@ravisingh1c ~]$ cat d.txt
three
three
1
2
```

```
2
3
3
[cloud_user@ravisingh1c ~]$ cat d.txt | uniq
three
1
2
3
[cloud_user@ravisingh1c ~]$
```

# LAB : PRACTISE – ERROR REDIRECTION , PIPES , FILTERS

# ***Compare files in Linux with 'diff'***



# LAB : DIFF and CMP COAMMAND

- Go to home directory and create file1 and file2
- Echo and add content in file 1 “Hello world I am superman “  
echo Hello world I am superman > file1
- Echo and add content in file 2 “Hello world I am superwoman”  
echo Hello world I am superwoman > file2

- Use diff command :

diff file1 file2

Use cmp command :

Cmp file1 file2

See the diff in the commands

# Common Unix tools



- **Date**

The date command can display the date, time, time zone.

```
[root@ravingh1c ~]# date
Wed Sep  4 18:24:04 UTC 2019
[root@ravingh1c ~]# date +%s ( Unix time in seconds )
1567621470
[root@ravingh1c ~]#
```

- **Sleep**

The sleep command is used in scripts or otherwise to put a wait for a number of seconds. This example shows a ten second sleep.

```
[root@ravingh1c ~]# date;sleep 10;date
Wed Sep  4 18:26:08 UTC 2019
Wed Sep  4 18:26:18 UTC 2019
[root@ravingh1c ~]#
```

# Common Unix tools



- **Time**

The time command can display how long it takes to execute a command. The date command takes only a little time.

```
[cloud_user@ravisingh1c ~]$ time cat d.txt
three
three
1
2
2
3
3

real  0m0.001s
user  0m0.001s
sys   0m0.000s
[cloud_user@ravisingh1c ~]$
```

# Common Unix tools



```
[cloud_user@ravisingh1c ~]$ time sleep 3

real    0m3.001s
user    0m0.001s
sys     0m0.000s
[cloud_user@ravisingh1c ~]$
```

- **Tar - Gzip - gunzip : Explained in upcoming lab**

Users never have enough disk space, so compression comes in handy. The gzip command can make files take up less space.

```
[cloud_user@ravisingh1c ~]$ cat /etc/passwd >>d.txt
[cloud_user@ravisingh1c ~]$ ls -ld d.txt
-rw-rw-r--. 1 cloud_user cloud_user 5828 Sep  4 18:31 d.txt
[cloud_user@ravisingh1c ~]$ gzip d.txt
[cloud_user@ravisingh1c ~]$ ls -ld d.txt.gz
-rw-rw-r--. 1 cloud_user cloud_user 842 Sep  4 18:31 d.txt.gz
```



# Common Unix tools



To get back to original state

```
[cloud_user@ravisingh1c ~]$ gunzip d.txt.gz  
[cloud_user@ravisingh1c ~]$ ls -ld d.txt  
-rw-rw-r--. 1 cloud_user cloud_user 5828 Sep  4 18:31 d.txt  
[cloud_user@ravisingh1c ~]$
```

- **Zcat - zmore**

Text files that are compressed with gzip can be viewed with zcat and zmore.

```
[cloud_user@ravisingh1c ~]$ head -4 d.txt  
three  
three  
1  
2  
[cloud_user@ravisingh1c ~]$
```

# Common Unix tools



```
[cloud_user@ravisingh1c ~]$ head -4 d.txt
three
three
1
2
[cloud_user@ravisingh1c ~]$ gzip d.txt
[cloud_user@ravisingh1c ~]$ head -4 d.txt.gz ( When run will print junk )

[cloud_user@ravisingh1c ~]$ zcat d.txt.gz|head -4
three
three
1
2
[cloud_user@ravisingh1c ~]$
```

**Pro Tip :** One can also use **bzip2**, **bunzip2**, **bzcat** and **bzmore**

# LAB : tar –gzip -gunzip

- Go to the home directory and then user directory.

```
Cd /home/rkhanna
```

- To combine all the data in /home/rkhanna directory , use the following command :

```
tar cvf rkhanna.tar .
```

```
ls -ltr
```

\*Create another directory named untarredfile and move the tarred file into this directory :

```
mkdir untarredfile
```

```
mv rkhanna.tar /home/rkhanna/untarredfile/
```

```
cd untarredfile/
```

\*Now try to untar or extract the tarred file created using the following command :

```
tar xvf rkhanna.tar
```

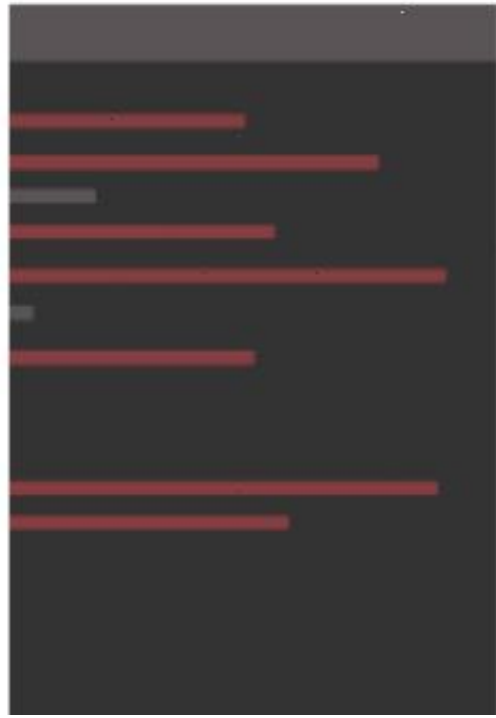
\*Now try to compress with file :

```
gzip rkhanna.tar
```

\*To unzip again :

```
gunzip rkhanna.tar
```

# How to Use Truncate Command in Linux



# Truncate Command :

```
truncate -s 40 filename
```

# echo

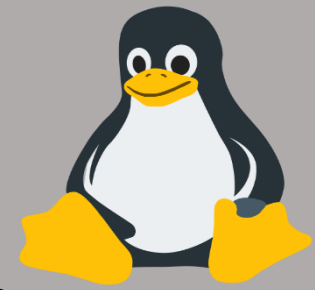
```
rrrrrrrrrrrrrrraaaaaaaaaaaaaaaammmmmmmmmmmmmmm  
mmmmmmmmaaaaaaaaaaaaannnnnnnnnnnnnnn > superman
```

```
truncate -s 40 superman
```

cat superman

(Observe the difference in file after truncating)

# User management



This little module will help to identify user account on a Unix computer using commands like who am i, id, and more.

- **Whoami**

The whoami command tells you your username.

```
[cloud_user@ravisingh1c ~]$ whoami
cloud_user
[cloud_user@ravisingh1c ~]$
```

- **Who**

The who command will give you information about who is logged on the system

```
[cloud_user@ravisingh1c ~]$ who
cloud_user pts/0    2019-09-04 18:01
who am i
[cloud_user@ravisingh1c ~]$ who am i
cloud_user pts/0    2019-09-04 18:01
```

With who am i the who command will display only the line pointing to your current session.

# User management



- **w**

The w command shows you who is logged on and what they are doing.

```
[cloud_user@ravisingh1c ~]$ w
19:08:08 up 1:07, 1 user, load average: 0.11, 0.05, 0.05
USER  TTY  FROM          LOGIN@  IDLE  JCPU  PCPU  WHAT
cloud_us pts/0  cpc87275-slou4-2 18:01   0.00s 0.36s 0.03s w
[cloud_user@ravisingh1c ~]$
```

- **id**

The id command will give you your user id, primary group id, and a list of the groups that you belong to.

```
[cloud_user@ravisingh1c ~]$ id
uid=1002(cloud_user) gid=1003(cloud_user)
groups=1003(cloud_user),10(wheel)
context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
[cloud_user@ravisingh1c ~]$
```

# User management



- **su to another user**

The su command allows a user to run a shell as another user.

```
[cloud_user@ravisingh1c ~]$ su - xyz
Password:
[xyz@ravisingh1c ~]$ id
uid=1004(xyz) gid=1006(xyz) groups=1006(xyz)
context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
[xyz@ravisingh1c ~]$
```

- **su to root**

you can also su to become root, when you know the root password.

```
[cloud_user@ravisingh1c ~]$ su root
Password:
[root@ravisingh1c cloud_user]#
[root@ravisingh1c cloud_user]# id
uid=0(root) gid=0(root) groups=0(root)
context=unconfined_u:unconfined_r:unconfined_t:s0-
s0:c0.c1023
[root@ravisingh1c cloud_user]#
```



# User management



- **su as root**

You need to know the password of the user you want to substitute to, unless your are logged in as root. The root user can become any existing user without knowing that user's password.

```
[root@ravisingh1c cloud_user]# su - xyz
Last login: Wed Sep  4 19:13:53 UTC 2019 on pts/1
[xyz@ravisingh1c ~]$ id
uid=1004(xyz) gid=1006(xyz) groups=1006(xyz)
context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
[xyz@ravisingh1c ~]$
```

- **su - \$username**

By default, the su command maintains the same shell environment. To become another user and also get the target user's environment, issue the su - command followed by the target username.

- **su -**

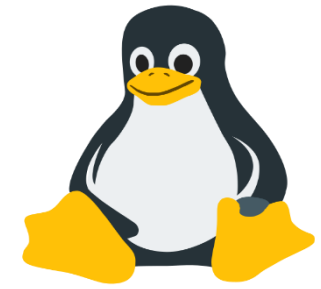
When no username is provided to su or su -, the command will assume root is the target.

LAB : LINUX

FUNDAMENTALS

..

# Introduction to vi



- The vi editor is installed on almost every Unix. Linux will very often install vim (vi improved) which is similar. Every system administrator should know vi(m), because it is an easy tool to solve problems. The vi once you get to know it, vi becomes a very powerful application.

- **Starting vi**

To start using vi, at the Unix prompt type vi followed by a file name. If you wish to edit an existing file, type in its name; if you are creating a new file, type in the name you wish to give to the new file.

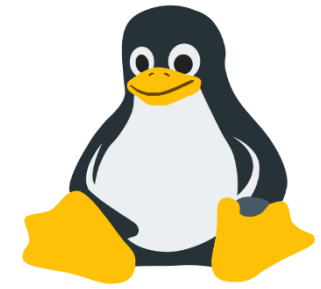
- **%vi filename**

Then hit Return. You will see a screen similar to the one below which shows blank lines with tildes and the name and status of the file.

eg.

vi x.txt

# Introduction to vi



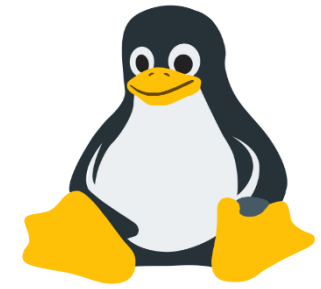
- **vi's Modes and Moods**

vi has two modes: the command mode and the insert mode. It is essential that you know which mode you are in at any given point in time. When you are in command mode, letters of the keyboard will be interpreted as commands. When you are in insert mode the same letters of the keyboard will type or edit text. vi always starts out in command mode. When you wish to move between the two modes, keep these things in mind. You can type `i` to enter the insert mode. If you wish to leave insert mode and return to the command mode, hit the ESC key. If you're not sure where you are, hit ESC a couple of times and that should put you back in command mode.

- **General Command Information**

As mentioned previously, vi uses letters as commands. It is important to note that in general vi commands: are case sensitive - lowercase and uppercase command letters do different things are not displayed on the screen when you type them generally do not require a Return after you type the command. You will see some commands which start with a colon (:). These commands are ex commands which are used by the ex editor. ex is the true editor which lies underneath vi -- in other words, vi is the interface for the ex editor.

# Introduction to vi



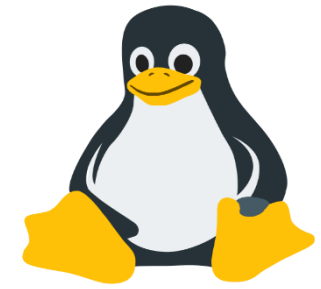
- **Entering Text**

To begin entering text in an empty file, you must first change from the command mode to the insert mode. To do this, type the letter i. When you start typing, anything you type will be entered into the file. Type a few short lines and hit Return at the end of each of line. Unlike word processors, vi does not use word wrap. It will break a line at the edge of the screen. If you make a mistake, you can use the Backspace key to remove your errors. If the Backspace key doesn't work properly on your system, try using the Ctrl h key combination.

- **Cursor Movement**

You must be in command mode if you wish to move the cursor to another position in your file. If you've just finished typing text, you're still in insert mode and will need to press ESC to return to the command mode.

# Introduction to vi



- **Moving One Character at a Time**

Try using your direction keys to move up, down, left and right in your file. Sometimes, you may find that the direction keys don't work. If that is the case, to move the cursor one character at the time, you may use the h, j, k, and l keys. These keys move you in the following directions:

- h left one space
- l right one space
- j down one space
- k up one space

- **Moving among Words and Lines**

While these four keys (or your direction keys) can move you just about anywhere you want to go in your file, there are some shortcut keys that you can use to move a little more quickly through a document. To move more quickly among words, you might use the following:

- w moves the cursor forward one word
- b moves the cursor backward one word (if in the middle of a word, b will move you to the beginning of the current word).
- e moves to the end of a word.

# Introduction to vi



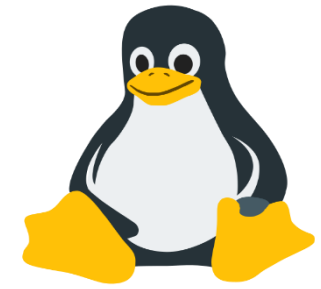
- **Command Keys and Case**

You will find when using vi that lower case and upper case command keys are interpreted differently. For example, when using the lower case w, b, and e commands, words will be defined by a space or a punctuation mark. On the other hand, W, B, and E commands may be used to move between words also, but these commands ignore punctuation.

- **Shortcuts**

Two short cuts for moving quickly on a line include the \$ and the 0 (zero) keys. The \$ key will move you to the end of a line, while the 0 will move you quickly to the beginning of a line.

# Introduction to vi



- **Moving by Searching**

One method for moving quickly to a particular spot in your file is to search for specific text. When you are in command mode, type a / followed the text you wish to search for. When you press Return, the cursor will move to the first incidence of that string of text. You can repeat the search by typing n or search in a backwards direction by using N.

- **Basic Editing**

To issue editing commands, you must be in command mode. As mentioned before, commands will be interpreted differently depending upon whether they are issued in lower or upper case. Also, many of the editing commands can be preceded by a number to indicate a repetition of the command.

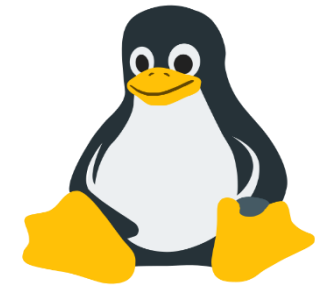
- **Deleting (or Cutting) Characters, Words, and Lines**

To delete a character, first place your cursor on that character. Then, you may use any of the following commands:

- x deletes the character under the cursor.
- X deletes the character to the left of your cursor.
- dw deletes from the character selected to the end of the word.
- dd deletes all the current line.
- D deletes from the current character to the end of the line.



# Introduction to vi



Preceding the command with a number will delete multiple characters. For example, `10x` will delete the character selected and the next 9 characters; `10X` will delete the 10 characters to the left of the currently selected character. The command `5dw` will delete 5 words, while `4dd` deletes four lines.

## Pasting Text using Put

Often, when you delete or cut text, you may wish to reinsert it in another location of the document. The Put command will paste in the last portion of text that was deleted since deleted text is stored in a buffer. To use this command, place the cursor where you wish the deleted text to appear. Then use `p` to reinsert the text. If you are inserting a line or paragraph use the lower case `p` to insert on the line below the cursor or upper case `P` to place in on the line above the cursor.

# Introduction to vi



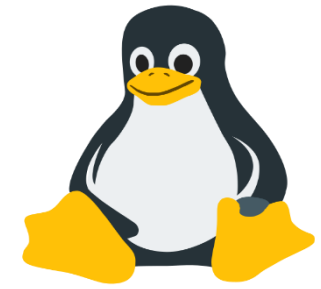
Once the desired text is yanked, place the cursor in the spot in which you wish to insert the text and then use the put command (p for line below or P for line above) to insert the contents of the buffer.

- **Replacing or Changing Characters, Words, and Lines**

When you are using the following commands to replace text, you will be put temporarily into insert mode so that you can change a character, word, line, or paragraph of text.

- **r**  
replaces the current character with the next character you enter/type. Once you enter the character you are returned to command mode.
- **R**  
puts you in overwrite mode until you hit ESC which will then return you to command mode.

# Introduction to vi



- **Inserting Text**

If you wish to insert new text in a line, first position the cursor to the right of where you wish the inserted text to appear. Type `i` to get into insert mode and then type in the desired text (note that the text is inserted before the cursor). Press `ESC` to return to command mode.

- **Inserting a Blank Line**

To insert a blank line below the line your cursor is currently located on, use the `o` key and then hit `ESC` to return to the command mode. Use `O` to insert a line above the line the cursor is located on.

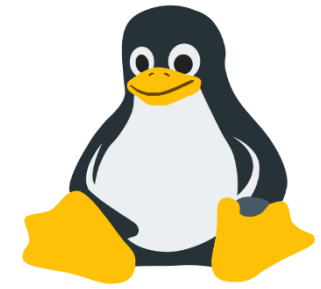
- **Appending Text**

You can use the append command to add text at any place in your file. Append (`a`) works very much like Insert (`i`) except that it insert text after the cursor rather than before it. Append is probably used most often for adding text to the end of a line. Simply place your cursor where you wish to append text and press `a`. Once you've finished appending, press `ESC` to go back to command mode.

- **Joining Lines**

use automatic word wrap, it is not unusual in editing lines to end up with lines that are too short and that might be improved if joined together. To do this, place your cursor on the first line to be joined and type `J`. As with other commands, you can precede `J` with a number to join multiple lines (`4J` joins 4 lines).

# Introduction to vi



- **Undoing**

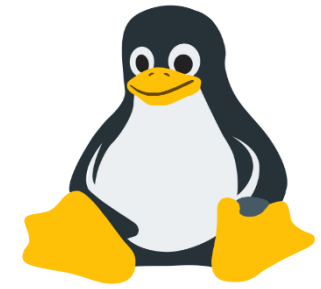
Be sure to remember this command. When you make a mistake you can undo it. DO NOT move the cursor from the line where you made the change. Then try using one of the following two commands:

- **u** undoes the last change you made anywhere in the file. Using u again will "undo the undo".
- **U** undoes all recent changes to the current line. You can not have moved from the line to recover the original line.

- **Closing and Saving Files**

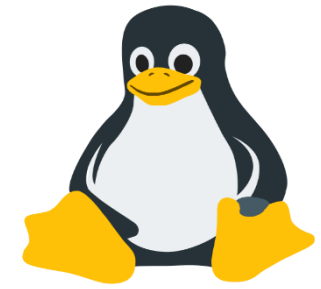
- When you edit a file in vi, you are actually editing a copy of the file rather than the original. The following sections describe methods you might use when closing a file, quitting vi, or both.
- Quitting and Saving a File

# Introduction to vi



- The command ZZ (notice that it is in uppercase) will allow you to quit vi and save the edits made to a file. You will then return to a Unix prompt. Note that you can also use the following commands:
- :w to save your file but not quit vi (this is good to do periodically in case of machine crash!).
- :q to quit if you haven't made any edits.
- :wq to quit and save edits (basically the same as ZZ).
- **Quitting without Saving Edits**  
Sometimes, when you create a mess (when you first start using vi this is easy to do!) you may wish to erase all edits made to the file and either start over or quit. To do this, you can choose from the following two commands:
- :q! wipes out all edits and allows you to exit from vi.

# Introduction to vi



Either of these combinations would work. So, as you can see, the general format for a command can be

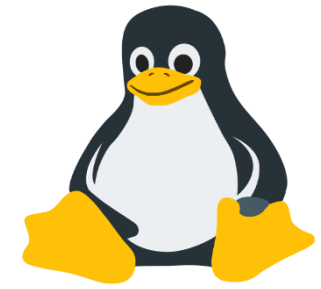
- (number) (command) (text object) or (command) (number) (text object)
- You might wish to try out some of the following combinations of commands and objects:

- Command      Text Object
- d (delete)      w (word to the left)

- **Repeating a Command**

If you are doing repetitive editing, you may wish to use the same command over and over. vi will allow you to use the dot (.) to repeat the last basic command you issued. If for example, you wished to delete several lines, you could use dd and then . (dot) in quick succession to delete a few lines.

# Introduction to vi



## Useful vi Commands

### Cut/Paste Commands:

- x delete one character (destructive backspace)
- dw delete the current word (Note: ndw deletes n numbered words)
- dd delete the current line (Note: ndd deletes n numbered lines)
- D delete all content to the right of the cursor
- :u undo last command

# SED COMMAND :

- SED command in UNIX stands for stream editor and it can perform lots of functions on file like searching, find and replace, insertion or deletion. Though most common use of SED command in UNIX is for substitution or for find and replace.



LAB : SED COMMAND ..

# User management



- This module will provide to do user management i.e how to use `useradd`, `usermod` and `userdel` to create, modify and remove user accounts. We will need to be root .

- **User management**

User management on Linux can be by using graphical tools provided by your distribution. These tools have a look and feel that depends on the distribution. Another option is to use command line tools like `useradd`, `usermod`, `gpasswd`, `passwd` and others. Server administrators are likely to use these tools, since they are familiar and very similar across many different distributions.

# User management



- **/etc/passwd**

The local user database on Linux (and on most Unixes) is /etc/passwd.

```
[xyz@ravisingh1c ~]$ head -10 /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
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
```

As you can see, this file contains seven columns separated by a colon. The columns contain the username, an x, the user id, the primary group id, a description, the name of the home directory, and the login shell.

# User management



- **Root**

The root user also called the superuser is the most powerful account on your Linux system. This user can do almost anything, including the creation of other users. The root user always has userid 0 (regardless of the name of the account).

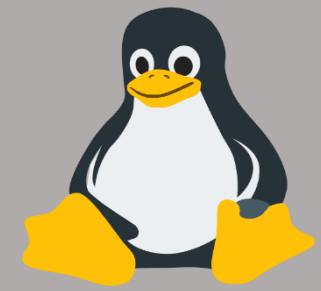
```
[xyz@ravisingh1c ~]$ head -1  
/etc/passwd  
root:x:0:0:root:/root:/bin/bash  
[xyz@ravisingh1c ~]$
```

## **Useradd**

You can add users with the useradd command. The example below shows how to add a user named yanina (last parameter) and at the same time forcing the creation of the home directory (-m), setting the name of the home directory (-d), and setting a description (-c).

```
[root@MyFirstLinuxVM home]# useradd spiderman
```

# User management



- **Userdel**

You can delete the user user1 with userdel. The -r option of userdel will also remove the home directory.

```
userdel -r spiderman
```

## **Usermod**

You can modify the properties of a user with the usermod command. This example uses usermod to change the description of the user user2

```
Useradd hulk  
usermod -G rkhanna hulk  
# added hulk user to rkhanna group
```

# User management



## Passwd

Passwords of users can be set with the passwd command. Users will have to provide their old password before twice entering the new one.

```
[root@ravisingh1c cloud_user]# passwd user2
Changing password for user user2.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
[root@ravisingh1c cloud_user]#
```

As you can see, the passwd tool will do some basic verification to prevent users from using too simple passwords. The root user does not have to follow these rules (there will be a warning though). The root user also does not have to provide the old password before entering the new password twice.

# Groups



Users can be listed in groups. Groups allow you to set permissions on the group level instead of having to set permissions for every individual user. Every Unix or Linux distribution will have a graphical tool to manage groups. Novice users are advised to use this graphical tool. More experienced users can use command line tools to manage users, but be careful: Some distributions do not allow the mixed use of GUI and CLI tools to manage groups (YaST in Novell Suse). Senior administrators can edit the relevant files directly with vi or vigr.

- **Groupadd**

Groups can be created with the groupadd command. The example below shows the creation of five (empty) groups.

- **Group file**

Users can be a member of several groups. Group membership is defined by the `/etc/group` file.

The first field is the group's name. The second field is the group's (encrypted) password (can be empty). The third field is the group identification or GID. The fourth field is the list of members, these groups have no members.

# Groups



- **Groups**

A user can type the groups command to see a list of groups where the user belongs to

- **Usermod**

Group membership can be modified with the useradd or usermod command.

- **Groupmod**

You can change the group name with the groupmod command

- **Groupdel**

You can permanently remove a group with the groupdel command.



# TALKING TO USERS



users  
wall  
write

# USERS COMMAND :

‘users’ command in Linux system is used to show the user names of users currently logged in to the current host.

# WALL COMMAND :



- wall is a command-line utility that displays a message on the terminals of all logged-in users. The messages can be either typed on the terminal or the contents of a file. wall stands for write all, to send a message only to a specific user use the write command.

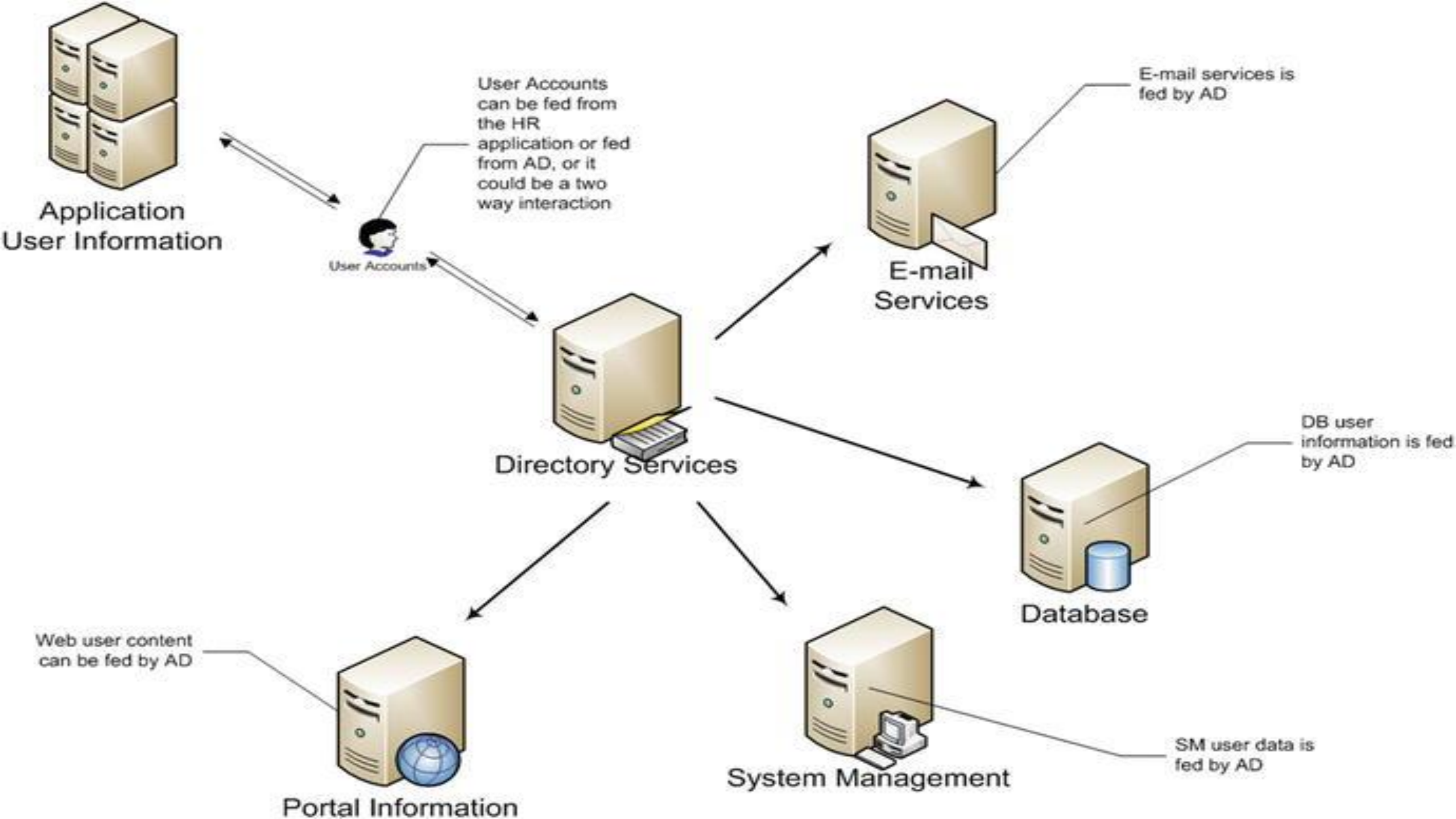
[~]\$ write

LAB : USERS, WALL ,WRITE ..

# **Recover Root Password LAB**

..

Account Authentication :





# DIFFERENT DIRECTORY SERVICES USED :

- ACTIVE DIRECTORY ( WINDOWS USERS – MICROSOFT)
- IDM : IDENTITY MANAGER (LINUX USERS)
- WinBIND (Samba) (ACTIVE DIRECTORY (WINDOWS USERS) TO AUTHENTICATE WITH LINUX SERVER OR CLIENT)
- IBM DIRECTORY SERVER
- JUMP CLOUD
- LDAP : LIGHTWEIGHT DIRECTORY ACCESS PROTOCOL

# **System Utility Commands**

# UPTIME COMMAND :

- It is used to find out how long the system is active (running). This command returns set of values that involve, the current time, and the amount of time system is in running state, number of users currently logged into, and the load time for the past 1, 5 and 15 minutes respectively.

# UNAME COMMAND :

- Uname command is used to display basic information about the operating system and hardware. With options, Uname prints kernel details, and system architecture. Uname is the short name for 'UNIX name'. Uname command works on all Linux and Unix like operating systems.

`uname -a`

## CAL COMMAND :

- The cal command displays a calendar of the specified year or month.

cal

cal 2 1994

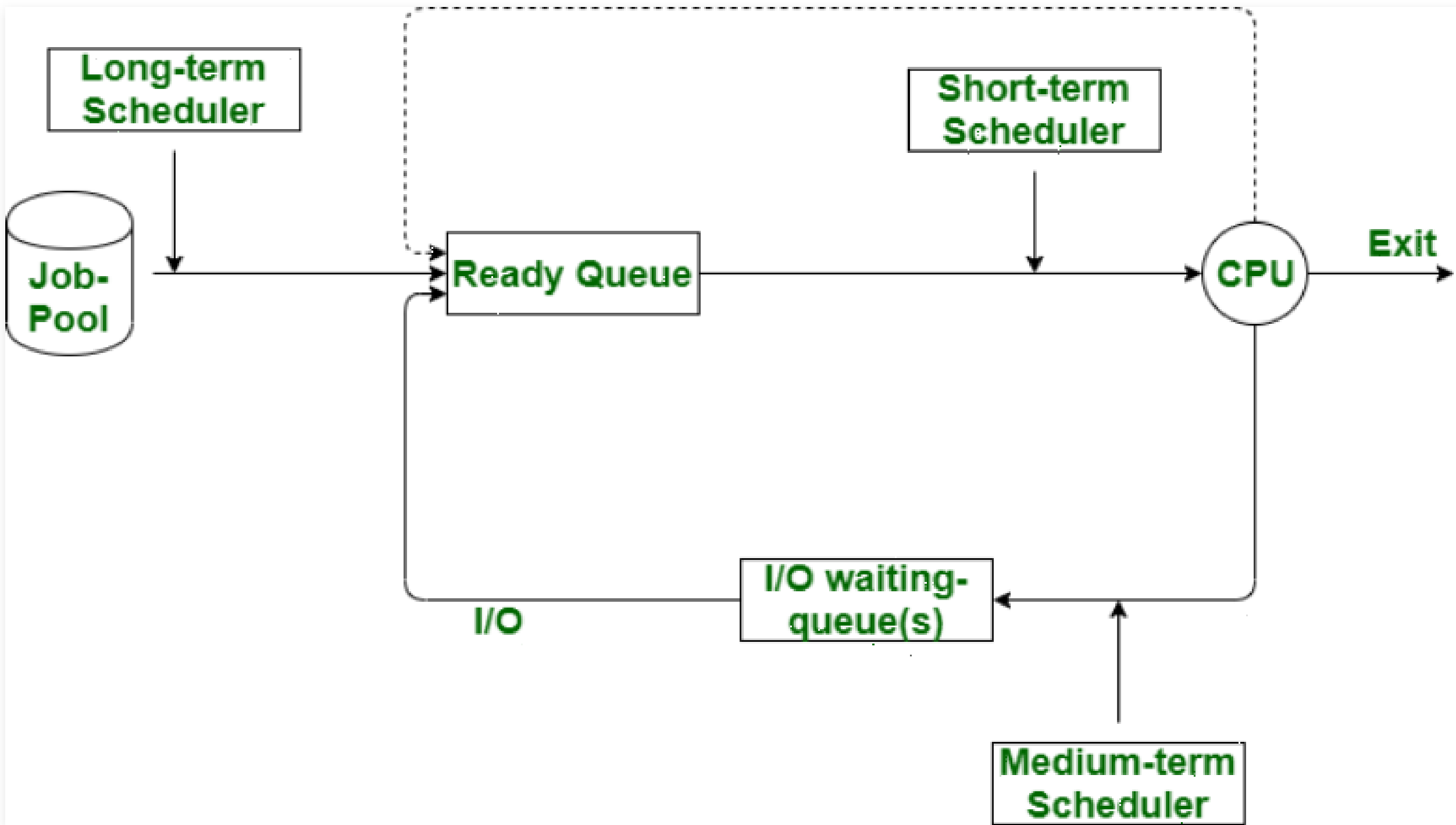
## Bc command :

- bc command is used for command line calculator. It is similar to basic calculator by using which we can do basic mathematical calculations. Arithmetic operations are the most basic in any kind of programming language.

PRACTISE SYSTEM UTILITY  
COMMANDS

# **Processes, Jobs and Scheduling**





# Systemctl command

- Systemctl is a Linux command-line utility used to control and manage services. You can think of Systemctl as a control interface allowing you to communicate with perform operations.

# COMMANDS :

- `systemctl status firewalld`
- `systemctl stop firewalld`
- `systemctl disable firewalld`
- `systemctl list-units`
- `systemctl list-units --all`
- `Systemctl restart firewalld`
- `systemctl enable firewalld`
- `Systemctl poweroff`
- `Systemctl halt`
- `Systemctl reboot`

# Ps command

ps command is used to list the currently running processes and their PIDs along with some other information depends on different options. It reads the process information from the virtual files in /proc file-system.

# Commands

- `ps` : shows all running processes in current shell
- `ps -e` : shows all running processes
- `ps aux` : shows all running processes in BSD format
- `ps -ef` : most commonly used
- `ps -u`
- `ps -u rkhanna`

# Top command :

- The top (table of processes) command shows a real-time view of running processes in Linux and displays kernel-managed tasks. The command also provides a system information summary that shows resource utilization, including CPU and memory usage.

# Commands :

- `top -u rkhanna`
- `top` and then press `c` : gives absolute path of process running
- `top` and then press `k` : to kill a particular process using PID
- `top` and then `M` and `P` : to show all processes acc to mem usage

## KILL COMMAND :

- It can be defined as a built-in command. It is used for manually terminating the processes.



# commands

- kill PID
- kill -1 PID : to restart a process
- kill -2 PID : interrupt from the keyboard
- kill -9 : forcefully kill a process
- kill -15 : kill a process gracefully
- killall : kill all parent and child processes
- pkill : kill process by name if doesn't know PID

## FINDING SYSTEM INFORMATION :

- `uname -a`
- `cat /etc/os-release`
- `dmidecode`
- `arch`



GNU/Linux Monitoring Tools

# Monitoring Commands

- top
- df -h
- dmesg
- iostat
- iostat 1
- netstat -rnv
- free -h
- cat /proc/cpuinfo
- cat /proc/meminfo

## SYSTEM MAINTENANCE COMMANDS :

- shutdown
- init 0-6 (runlevels on next slide )
- reboot
- halt

0 – Halt

1 – Single user text mode

2 – Not used (user – definable)

3 – Full multi-user text mode

4 – Not used – ( user – definable )

5 -- Full multi-user graphical mode (with an X-based login screen)

6 -- Reboot

# Hostname command

- To change the hostname of a particular server or VM :

`hostnamectl set-hostname raman`

- \* To change hostname from etc config file :

`cat /etc/hostname`

- \* After changing reboot will be needed to see the changes

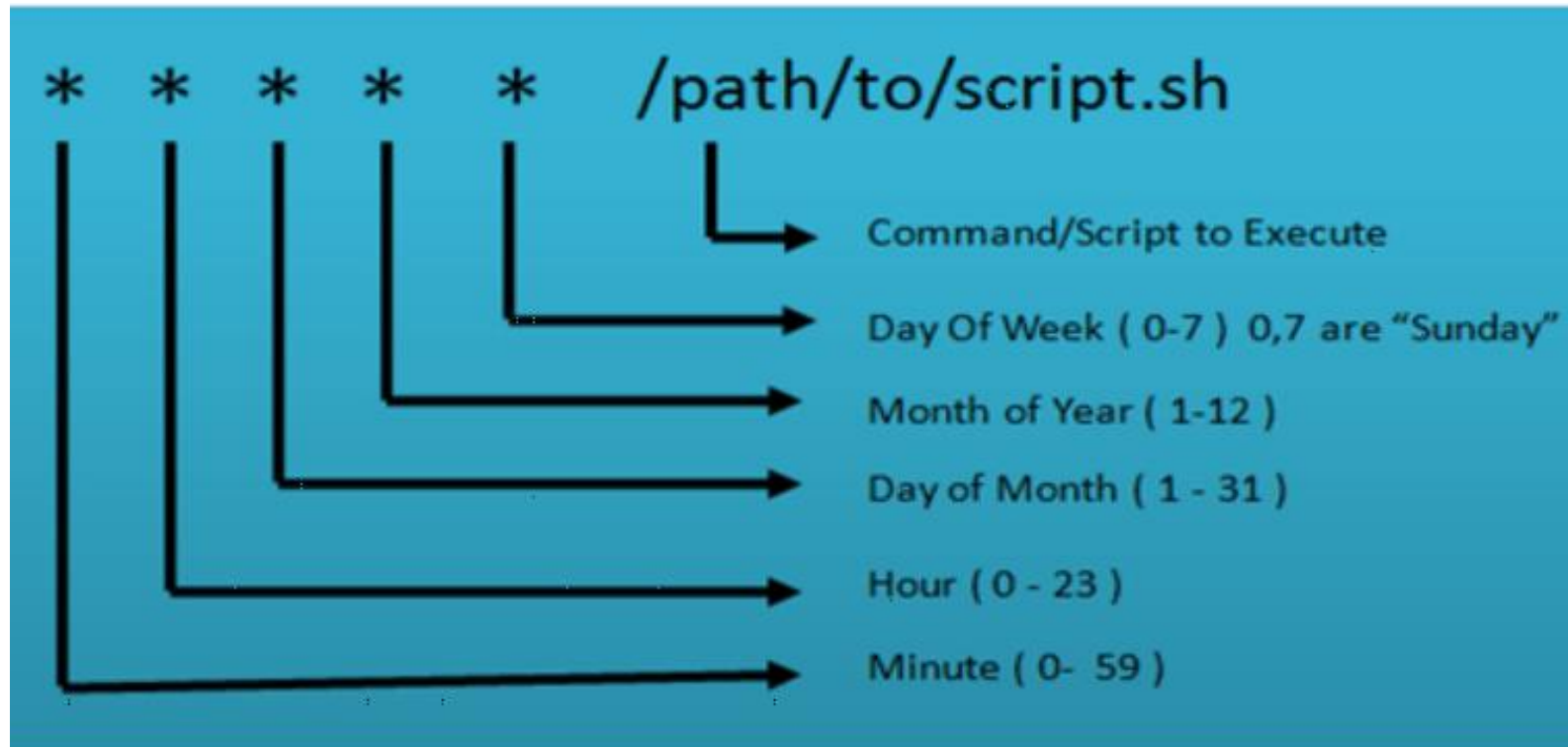
# CRONTAB COMMAND :

- The crontab command is used to view or edit the table of commands to be run by cron. Each user on your system can have a personal crontab. Crontab files are located in /var/spool/ (or a subdirectory such as /var/spool/cron/crontabs), but they are not intended to be edited directly.



# Commands :

- crontab -e : to edit the crontab file
- crontab -l : list all the crontabs
- crontabs -r : to remove the crontabs



# LAB : SCHEDULING A JOB USING CRONTAB

..

# AT command :

- at command is a command-line utility that is used to schedule a command to be executed at a particular time in the future. Jobs created with at command are executed only once. The at command can be used to execute any program or mail at any time in the future.

# commands

- `at HH:MM PM` : to create an at schedule
- `atq` : to list all at entries
- `atrm #` : to remove any at entry

LAB - AT COMMAND LAB

..

# PROCESS MANAGEMENT IN LINUX



# COMMANDS :

- Background :

Cntrl+Z : this stops a particular process.

Jobs : this shows u all the processes stopped

bg : run the process in background

- Foreground : fg
- nohup sleep 150 &

# PROCESS MGMT LAB

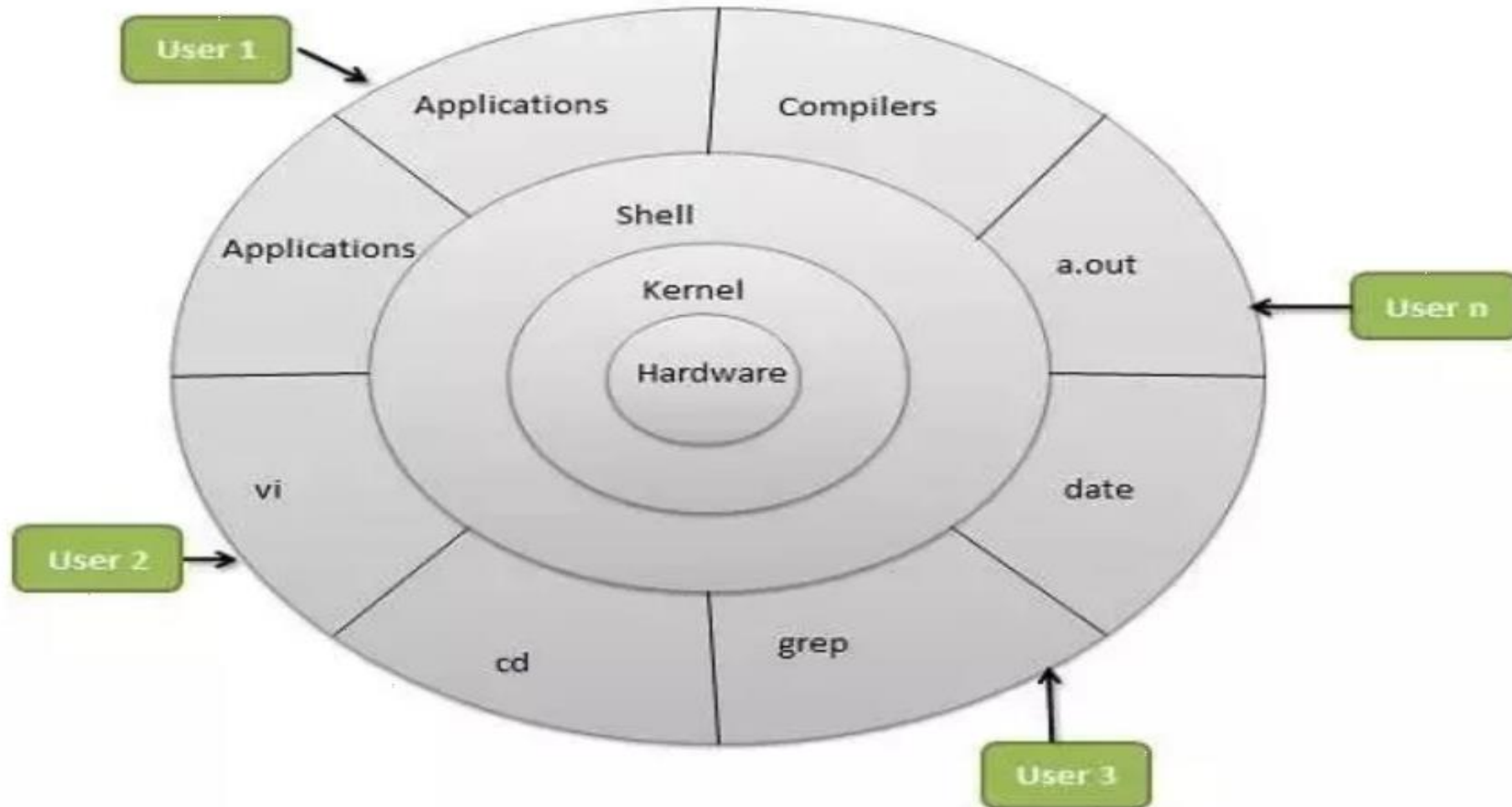
..



SYSTEM ADMINISTRATION MODULE LAB..

**SCRIPTING**

# KERNEL



# SHELL

The shell is the Linux command line interpreter. It provides an interface between the user and the kernel and executes programs called commands. For example, if a user enters `ls` then the shell executes the `ls` command.

# TYPES OF LINUX SHELLS :

- The Bourne Shell (sh) Developed at AT&T Bell Labs by Steve Bourne, the Bourne shell is regarded as the first UNIX shell ever.  
...
- GNOME or KDE
- The GNU Bourne-Again Shell (bash) ...
- The C Shell (csh) ...
- The Korn Shell (ksh) ...
- The Z Shell (zsh)

# SHELL SCRIPTING :

- Using a shell script is most useful for repetitive tasks that may be time consuming to execute by typing one line at a time. A few examples of applications shell scripts can be used for include: Automating the code compiling process. Running a program or creating a program environment.

# LAB - BASIC OUTPUT SHELL SCRIPT ..

- Basic Script to get an output onto the screen.

# LAB - RUN TASKS SCRIPT ..

- Run tasks script to execute commands together



LAB- Defining variable in script ..

LAB- INPUT OUTPUT SCRIPT ..

LAB- IF-THeN SCRIPT ..

## File existence check script :

- If – then –else script to check if file exist or not .

LAB- Check response and output script

..

LAB- SIMPLE FOR LOOP OUTPUT FIRST  
SCRIPT ..

**LAB- SIMPLE FOR LOOP OUTPUT SECOND SCRIPT ..**

LAB- for loop to create 5 files named 1-5

..



LAB- do-done Script ..

LAB- CASE SCRIPT

..

# PRACTISE SCRIPTS :

- \*Create a script to the output of `ls -ltr` to a file called `golistig`
- \*Create a script to output "First day of Spring is March 20th"
- \*Create a script that will run commands, `who`, `dmidecode`, `cal`, and `free`
- \*Create a script that will touch a new file and then change its permissions to read, write and execute for everyone
- Create a script that will ask you for your parent name and then output your parent name on the screen as, Your parent name is  
.....
- Write a case script that will give the option to the user to run commands as, `top`, `iostat`, `free`, `/proc/cpuinfo`, `dmesg`