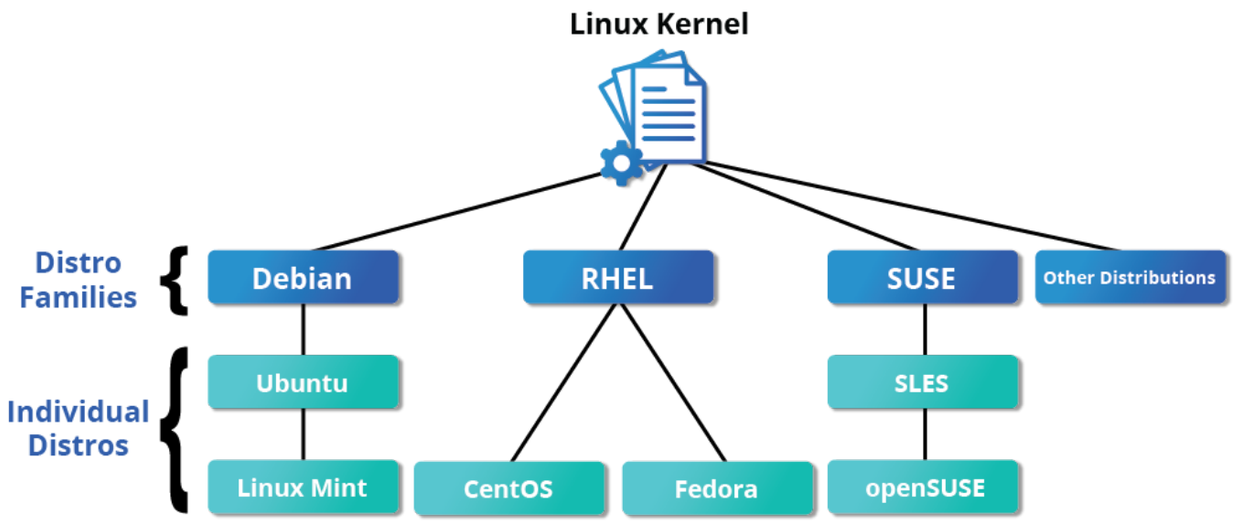
Chap1 - The Linux Foundation



Red Hat Family

> Red Hat Enterprise Linux (RHEL) heads the family that includes **CentOS, CentOSStream, Fedora and Oracle Linux**. Supports multiple hardware systems, used by enterprises that host their own systems.
>
> **Fedora** - contains significantly more software than RedHat's enterprise version - more diverse community involved in building Fedora. It's used as upstream testing platform for future RHEL releases.
>
> **CentOS & CentOS Stream** - No cost to end user and and longer release cycle than Fedora. The basic version of CentOS is also virtually identical to RHEL, the most popular Linux distribution in enterprise environments.The difference between the two versions is **CentOS Stream** gets updates **before** RHEL, while **CentOS** gets them **after**.

SUSE Family

> **openSUSE** - reference distribution for SUSE family, available to end users at no cost. Some situations material that covers openSUSE can cover SLES.
>
> **SLES** (SUSE Linux Enterprise Server) - upstream for SUSE family. Used in retail sector.
>
> Uses **RPM-based zypper package manager** - install, remove and update packages.
> **YaST**(Yet Another Setup Tool) - administration process

Debian Family

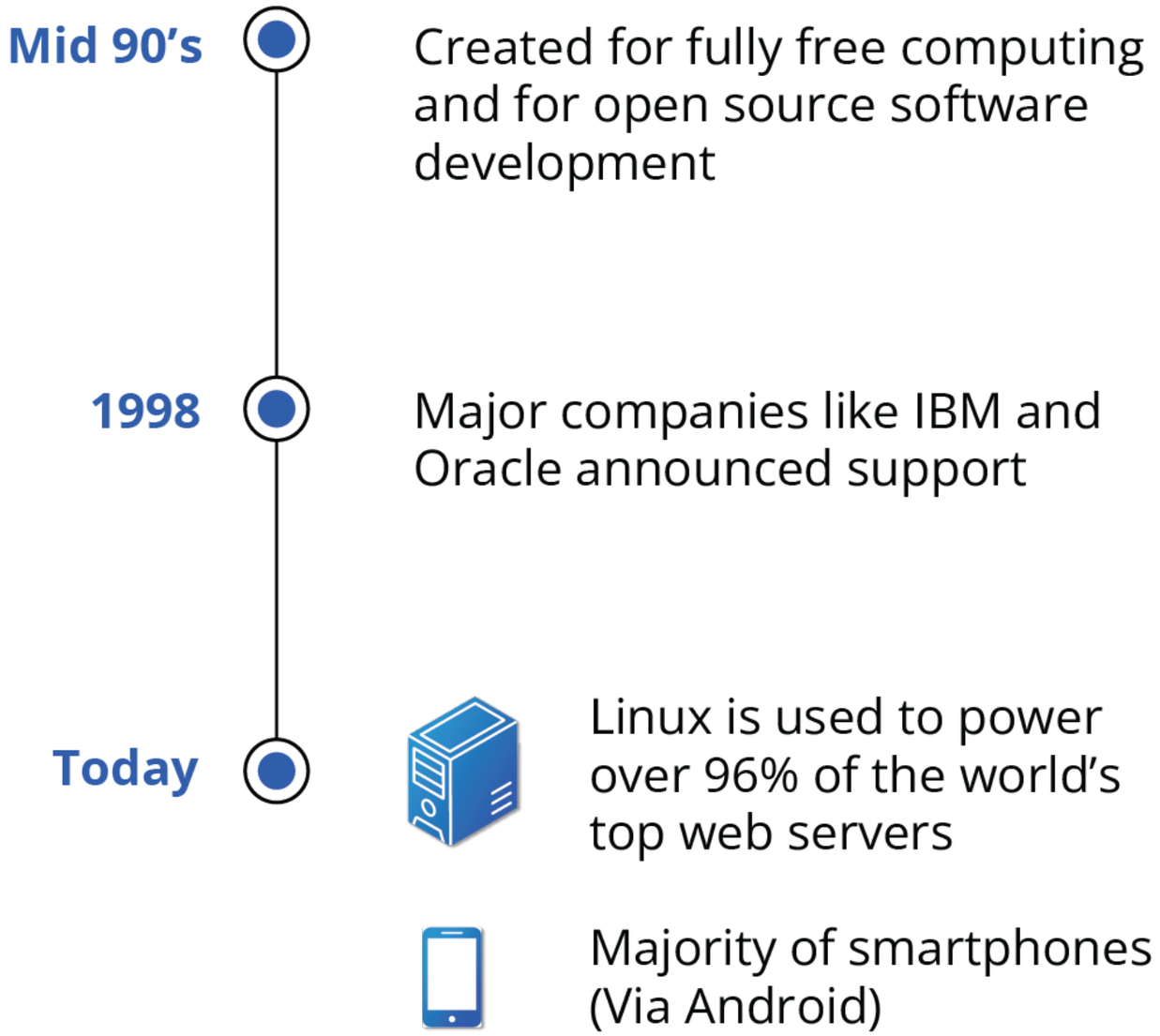
> Debian -> upstream for many distros including Ubuntu -> which is upstream for Linux Mint and other distros.
> Used in both server and desktop computers, open source and focus on stability. Used for cloud deployments.
> Debian - largest and most complete software software repository to its users of any Linux distribution.

> **Ubuntu** - compromise between stability and ease of use. Gets packages from stable Debian branch so has access to very large software repository. Built on top of Debian and is GNOME-based under the hood, but differs visually from the interface on standard Debian, as well as other distributions.
> Uses **DPKG-based APT package manager** - to install, remove, update packages.

Chap2 - Linux Philosophy and Concepts

History

> **Linux** - open source computer operating system, initially developed on and for **Intel x86-based personal computers**. Later, ported to other hardware platforms, from tiny embedded appliances to the world's largest supercomputers.
> Created by ***Linus Tolvards*** in 1991. Re-licensed using **General Public License(GPL)** by **GNU**.
> Combining kernel with other system components from GNU project - developers created complete systems called **Linux Distributions** (created in mid-90s).



> Today, Linux powers more than half of the servers on the Internet, the majority of smartphones (via the Android system, which is built on top of Linux), more than 90 percent of the public cloud workload, and all of the world's most powerful supercomputers.

Philosophy

> Linux is constantly enhanced and maintained by a network of developers from all over the world collaborating over the Internet, with Linus Torvalds at the head. Technical skills, a desire to contribute, and the ability to collaborate with others are the only qualifications for participating.
>
> Linux borrows heavily from the well-established family of UNIX operating systems. It was written to be a free and open source alternative; at the time, UNIX was designed for computers much more powerful than PCs, and furthermore, it was quite expensive.
>
> Files - stored in hierarchical filesystem - top node called **root** or **'/'**. Processes, devices and network devices represented as file-like objects and can be worked with using same utilities as regular files.
>
> Linux - fully multitasking, multiuser OS with multiple built-in networking and service processes called ***daemons***.

Community

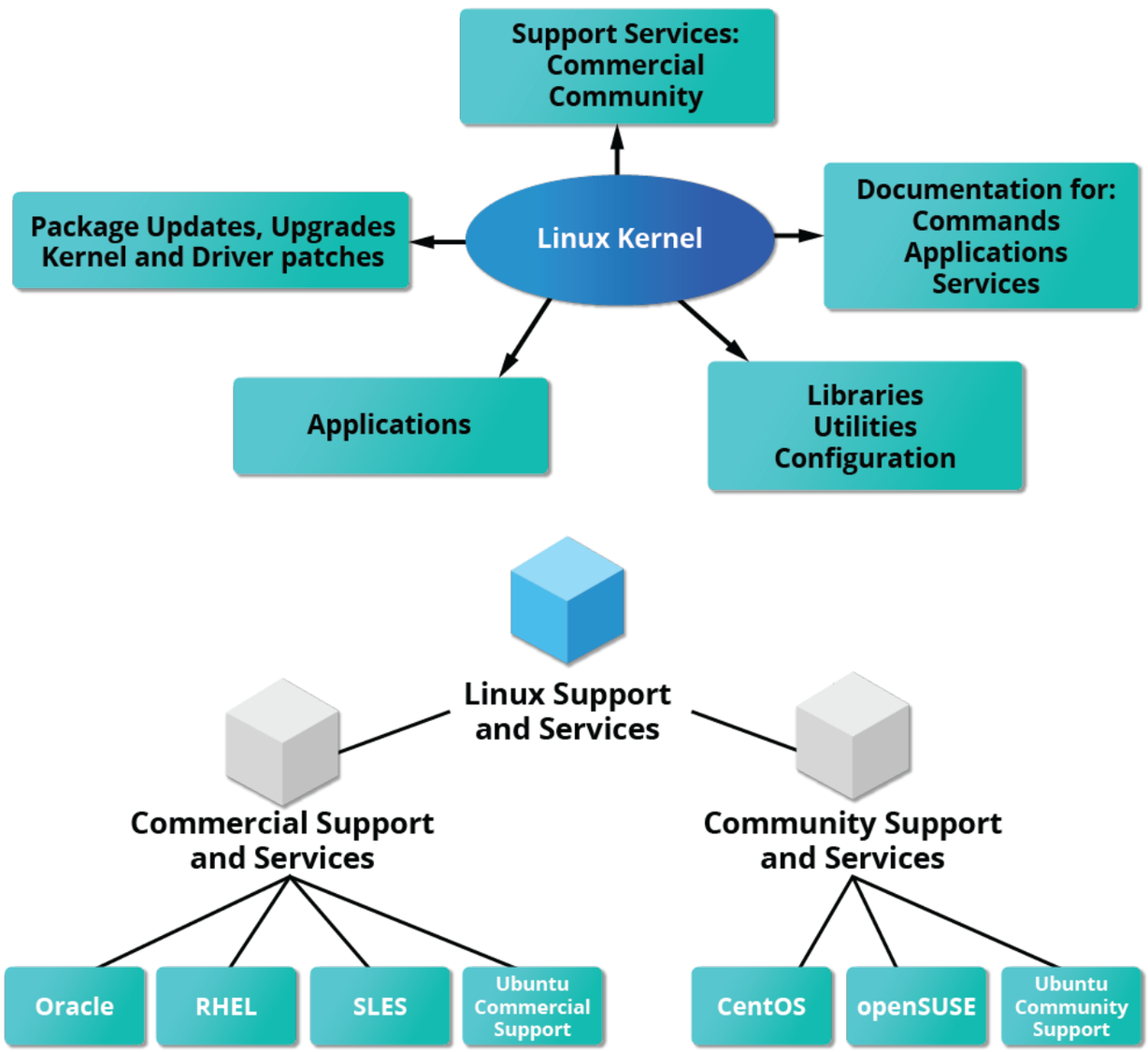
> The Linux community is a far-reaching ecosystem consisting of developers, system administrators, users, and vendors who use many different forums to connect with one another. Among the most popular are:
>
> - Internet Relay Chat (IRC) software (such as WeeChat, HexChat, Pidgin, and XChat)
> - Online communities and discussion boards including Linux User Groups (both local and online)
> - Many collaborative projects hosted on services such as GitHub and GitLab
> - Newsgroups and mailing lists, including the Linux Kernel Mailing List
> - Community events, e.g., Hackathons, Install Fests, Open Source Summits, Embedded Linux Conferences, and many other conferences and get-togethers.

Terminology

> **Kernel** - glue between hardware and applications - considered the brain of the Linux operating system. It controls the hardware and makes the hardware interact with the applications. The most recent Linux kernel, along with past Linux kernels, can be found at ***kernel.org web-site***.
> **Distribution** - collection of softwares/programs making up Linux-based OS.
> **Boot Loader** - program that boots the OS. Ex: ***GRUB*** and ***ISOLINUX***.
> **Service** - program that runs as background process. Ex: ***httpd, nfsd, ntpd, ftpd and named.***
> **Filesystem** - method for storing and organizing files in Linux. Ex: ***ext3, ext4, FAT, XFS and Btrfs.***
> **X Window System** - provides the standard toolkit and protocol to build graphical user interfaces on nearly all Linux systems.
> **Desktop environment** - graphical user interface on top of the operating system. Ex: ***GNOME, KDE, Xfce and Fluxbox***.

Distribution

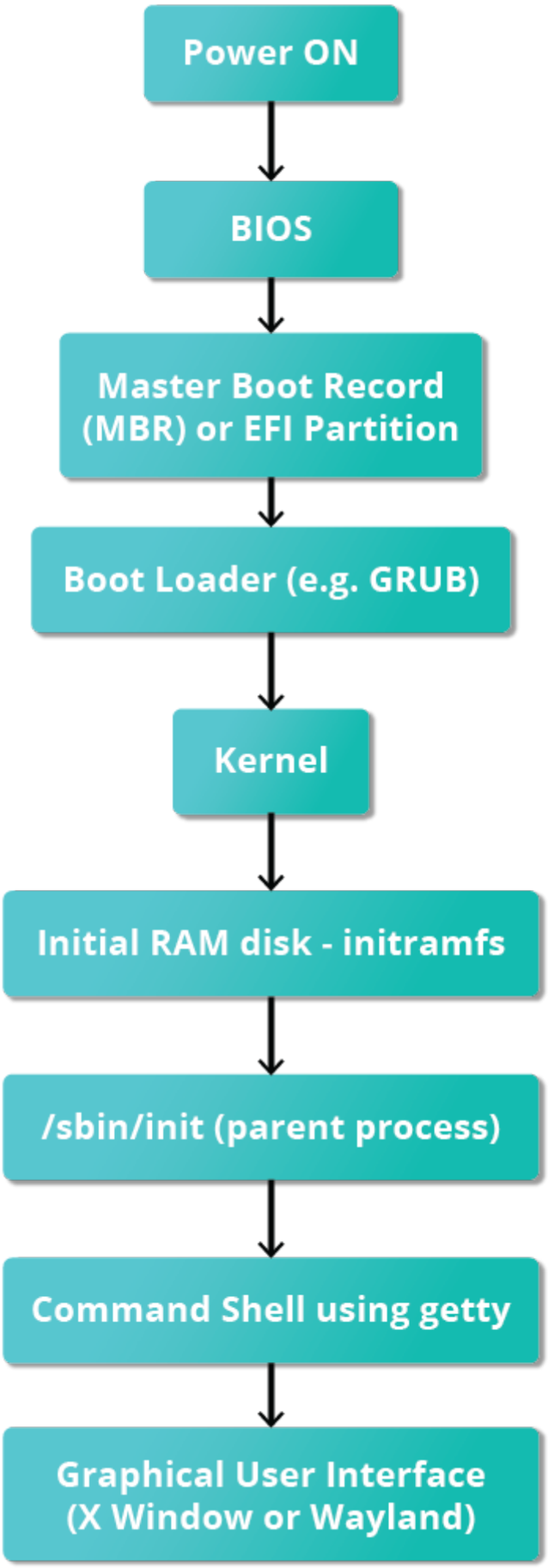
> The **Linux kernel** is the core of the operating system. A full Linux distribution consists of the kernel plus a number of other software tools for file-related operations, user management, and software package management. Each of these tools provides a part of the complete system. Each tool is often its own separate project, with its own developers working to perfect that piece of the system.
>
> Examples of other essential tools and ingredients provided by distributions include the C/C++ and Clang compilers, the gdb debugger, the core system libraries applications need to link with in order to run, the low-level interface for drawing graphics on the screen, as well as the higher-level desktop environment, and the system for installing and updating the various components, including the kernel itself. And all distributions come with a rather complete suite of applications already installed.



Chap3 - Linux Basics and System Startup

Boot Process

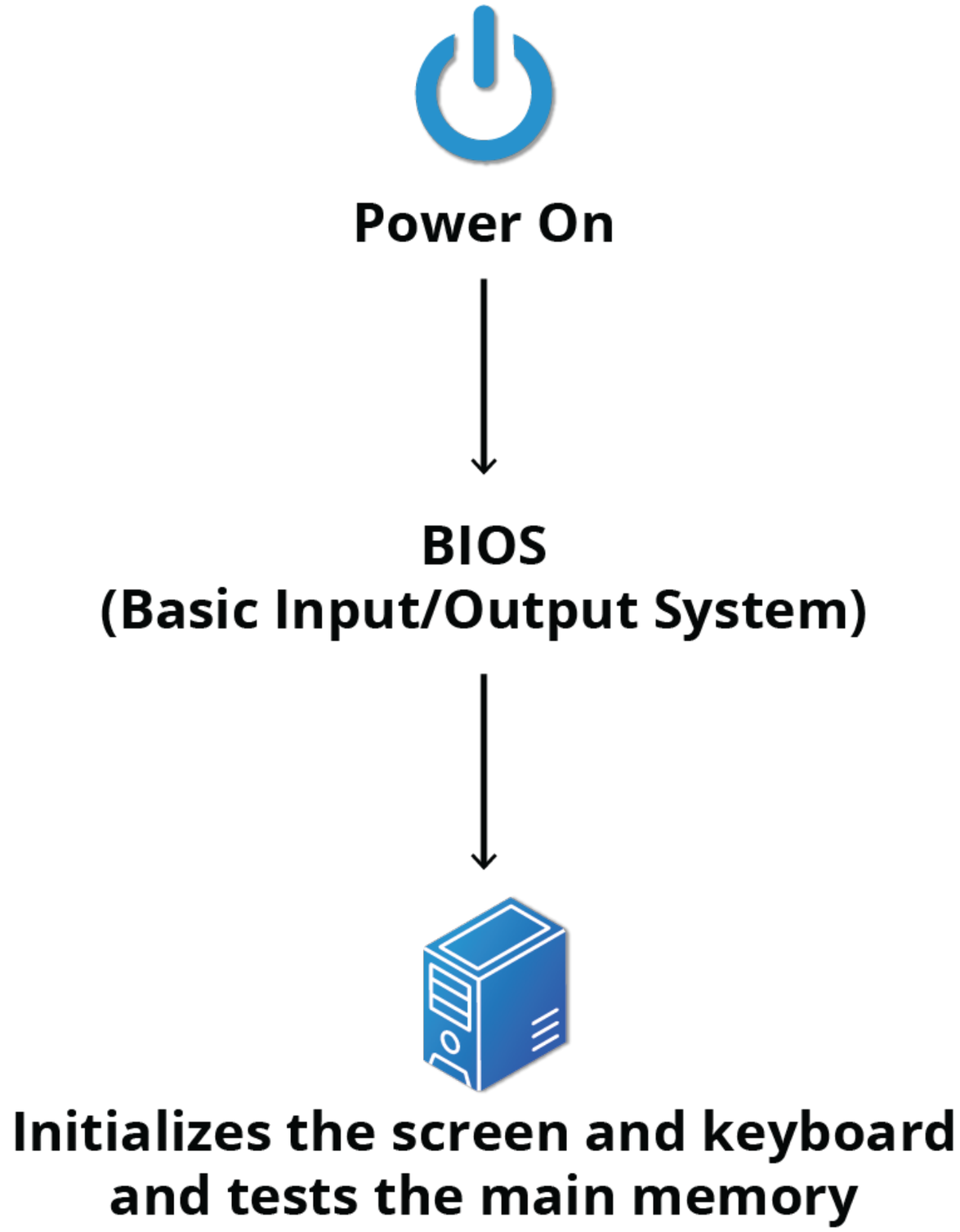
> Procedure for initializing the system. It consists of everything that happens from when the computer power is first switched on until the user interface is fully operational.



BIOS

x-86-based Linux system switched on - **Basic Input/Output System (BIOS)** initializes the hardware, including the screen and keyboard, and tests the main memory. This process is also called **POST (Power On Self Test)**. BIOS software - stored in ROM. After this,

remainder of the boot process is controlled by the OS.



Master Boot Record (MBR), EFI Partition, and Boot Loader

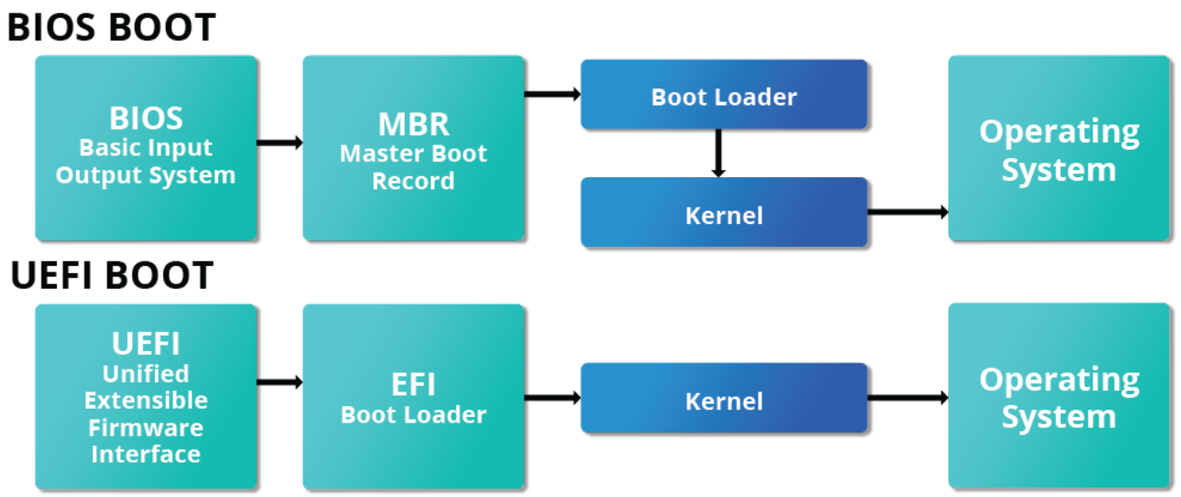
Once **POST** completed - system control passes from the BIOS to the **boot loader**. The boot loader - stored on ***hard disk or SSD drive***, either in the **boot sector** (for traditional BIOS/MBR systems) or the **EFI partition** (Unified Extensible Firmware Interface or EFI/UEFI systems).

Up to this stage, the machine does not access any mass storage media. Then, information on the date, time, and the most important peripherals are loaded from the **CMOS** values (allows the system to keep track of the date and time even when it is powered off).

A number of boot loaders exist for Linux; the most common ones are **GRUB** (for GRand Unified Boot loader), **ISOLINUX** (for booting from removable media), and **DAS U-Boot** (for

booting on embedded devices/appliances).

When booting Linux, the boot loader is responsible for loading the kernel image and the initial RAM disk or filesystem into memory.



Boot Loader

For systems using the **BIOS/MBR** method, the boot loader resides at the first sector of the hard disk, **Master Boot Record (MBR)**. The size of the MBR is **512 bytes**. In this stage, the boot loader examines the **partition table** and finds a bootable partition. Once it finds a bootable partition, it then searches for the second stage boot loader, for example **GRUB**, and loads it into RAM (Random Access Memory).

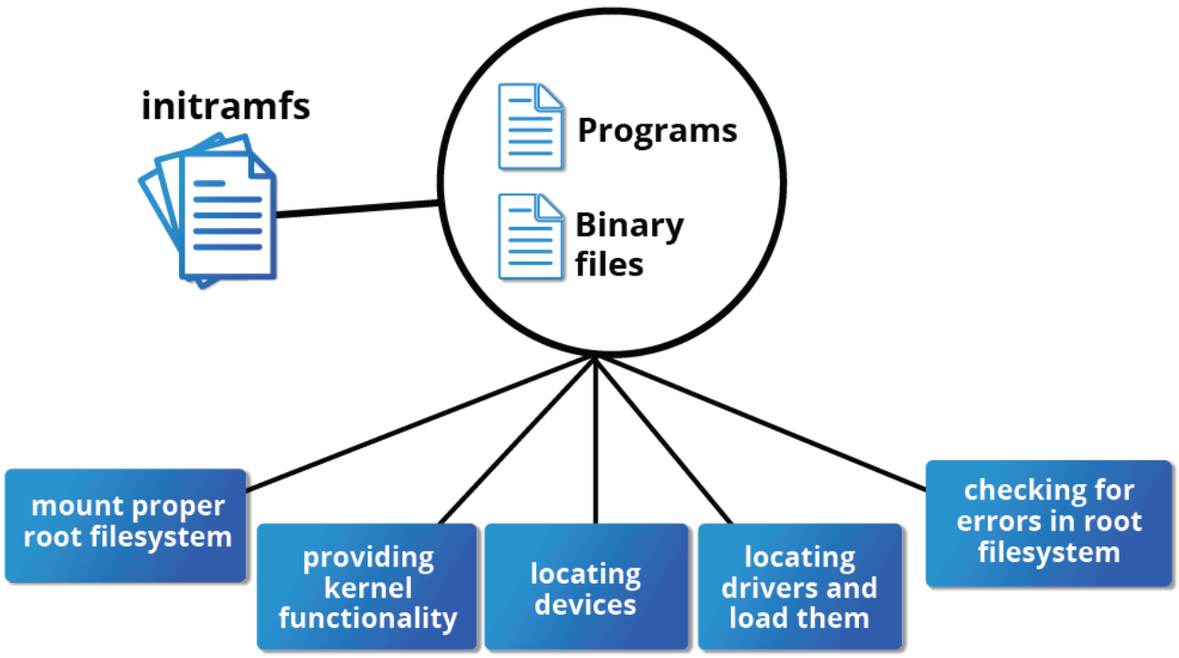
For systems using the **EFI/UEFI** method, UEFI **firmware reads Boot Manager data to determine which UEFI application is to be launched and from where**. The firmware then launches the UEFI application, for example GRUB, defined in the boot entry in the firmware's boot manager. This procedure is more complicated but more versatile than the older MBR methods.

The second stage boot loader resides under **/boot**. After the OS and kernel are selected, the boot loader loads the kernel of the OS into RAM and passes control to it. Kernels are almost always compressed, so the first job they have is to uncompress themself. After this, it will check and analyze the system hardware and initialize any hardware device drivers built into the kernel.

Initial RAM disk

The **initramfs** filesystem image contains programs and binary files that perform all actions needed to **mount the proper root filesystem**, including providing the kernel functionality required for the specific filesystem and loading the device drivers for mass storage controllers, by taking advantage of the **udev** system (for user device), responsible for **figuring out which devices are present, locating the device drivers they need to operate properly, and loading them**. After the root filesystem has been found, it is checked for errors and mounted.

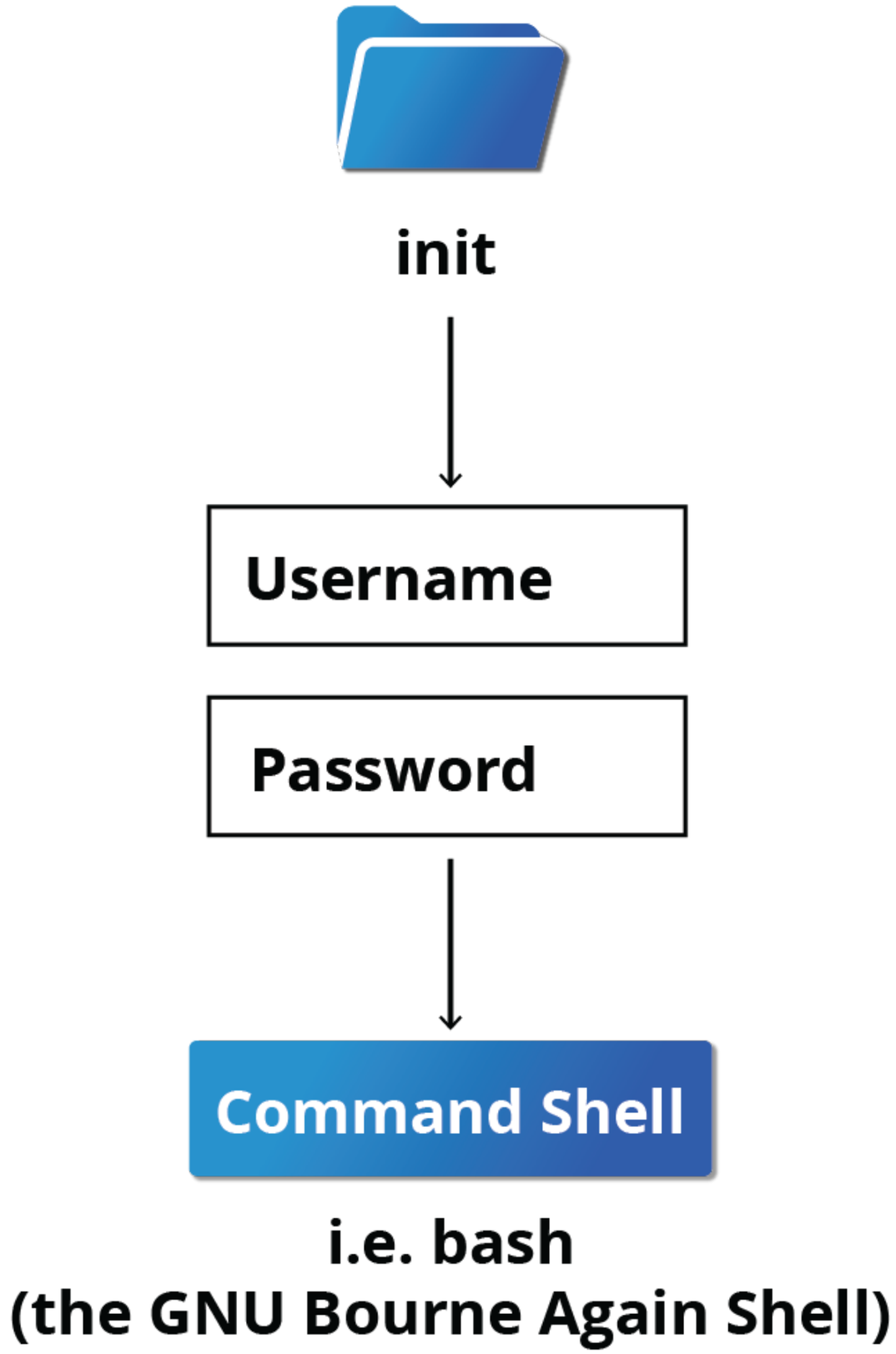
The **mount** program instructs the operating system - filesystem is ready for use and associates it with a particular point in the overall hierarchy of the filesystem (the **mount point**). If this is successful, the initramfs is cleared from RAM, and the **init** program on the root filesystem (**/sbin/init**) is executed.



Text-Mode Login

End of boot process- **init** starts number of text-mode login prompts. It enables you to type your username, followed by your password, and to eventually get a command shell.

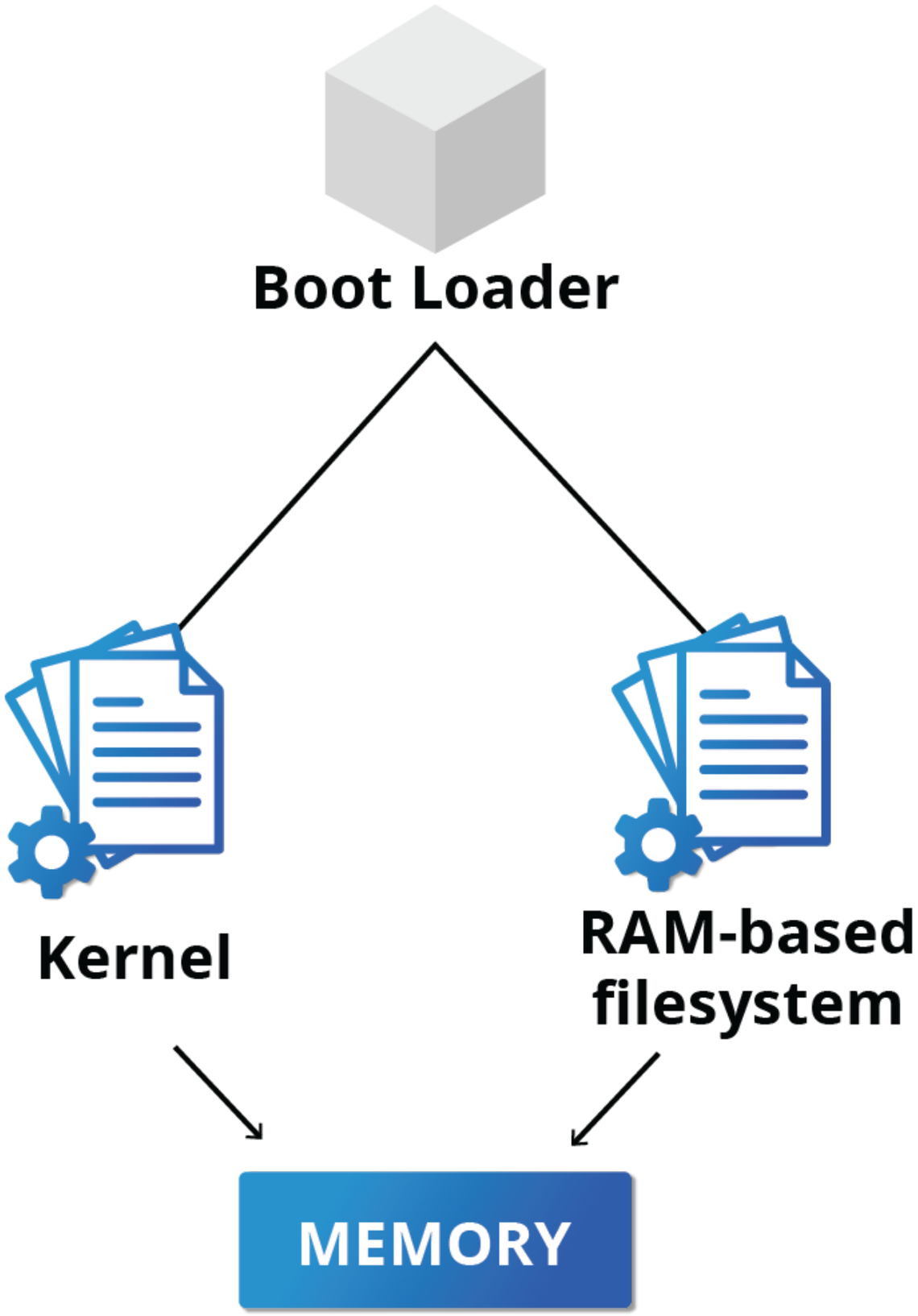
Default command shell - **bash** (GNU Bourne Again Shell).



Linux Kernel

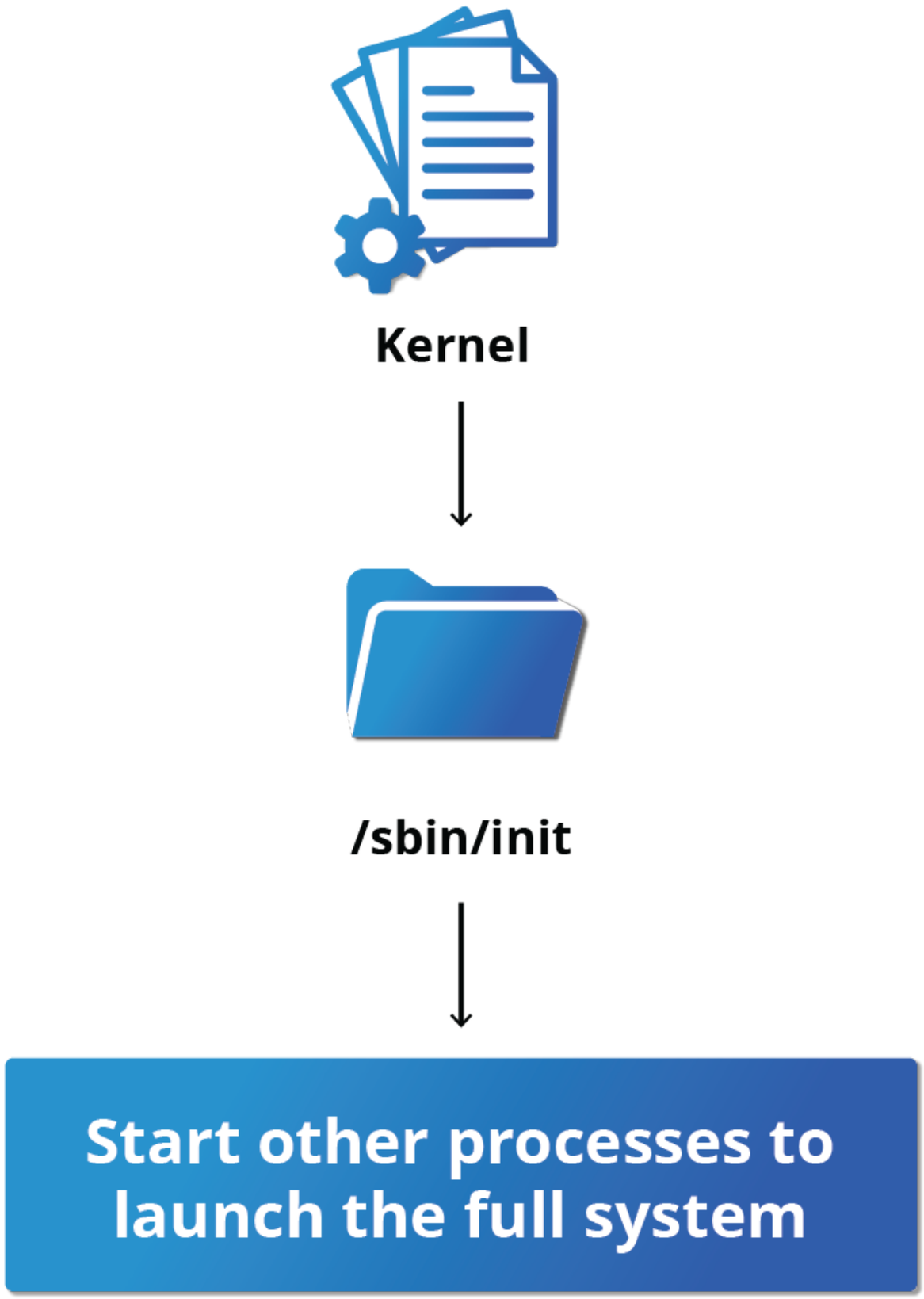
Boot loader loads both **kernel** and **initramfs** into RAM, so that it can be directly used by the kernel. When the kernel is loaded in RAM, it immediately initializes and configures the computer’s memory and also configures all the hardware attached to the system. Kernel

also loads necessary user-space applications.



/sbin/init and Services

Once kernel - setup all hardware and mounted root filesystems - kernel runs **/sbin/init**. This becomes initial process that starts other processes to get system running.
Most processes - trace back to **init**, except kernel processes - started directly by kernel to manage internal OS details.
init - responsible for keeping system running and shutting down cleanly. It **manages all non-kernel processes** - cleans up after them upon completion, and restarts user login services as needed when users log in and out, and does the same for other background system services.



Traditionally, this process startup was done using conventions that date back to the 1980s and the System V variety of UNIX. This serial process (called **SysVinit**) had the system pass through a sequence of **runlevels** containing collections of scripts that start and stop services. Each runlevel supported a different mode of running the system. Within each runlevel, individual services could be set to run, or to be shut down if running.

Startup alternatives

SysVinit - serial process, divided into series of sequential stages, each stage required completion before next could proceed. No parallel processing. No consideration for startup time and rebooting.

Main alternatives:>

Upstart

- Developed by Ubuntu and first included in 2006
- Adopted in Fedora 9 (in 2008) and in RHEL 6 and its clones

systemd

- Adopted by Fedora first (in 2011)
- Adopted by RHEL 7 and SUSE
- Replaced Upstart in Ubuntu 16.04

---

systemd Features

Faster start up than earlier **init** methods. Replaces serialised steps with **aggressive parallelization techniques**.

Complicated start up shell scripts replaced with **simple config files** - enumerate what has to be done before a service is started, how to execute service startup, and what conditions the service should indicate have been accomplished when startup is finished.

/sbin/init now just points to **/lib/systemd/systemd**; i.e. **systemd** takes over the **init** process

One **systemd** command (**systemctl**) is used for most basic tasks:

- Starting, stopping, restarting a service (using **httpd**, the Apache web server, as an example) on a currently running system:
  `$ sudo systemctl start|stop|restart httpd.service`
- Enabling or disabling a system service from starting up at system boot:
  `$ sudo systemctl enable|disable httpd.service`
- Checking on the status of a service:
  `$ sudo systemctl status httpd.service`

Linux Filesystems

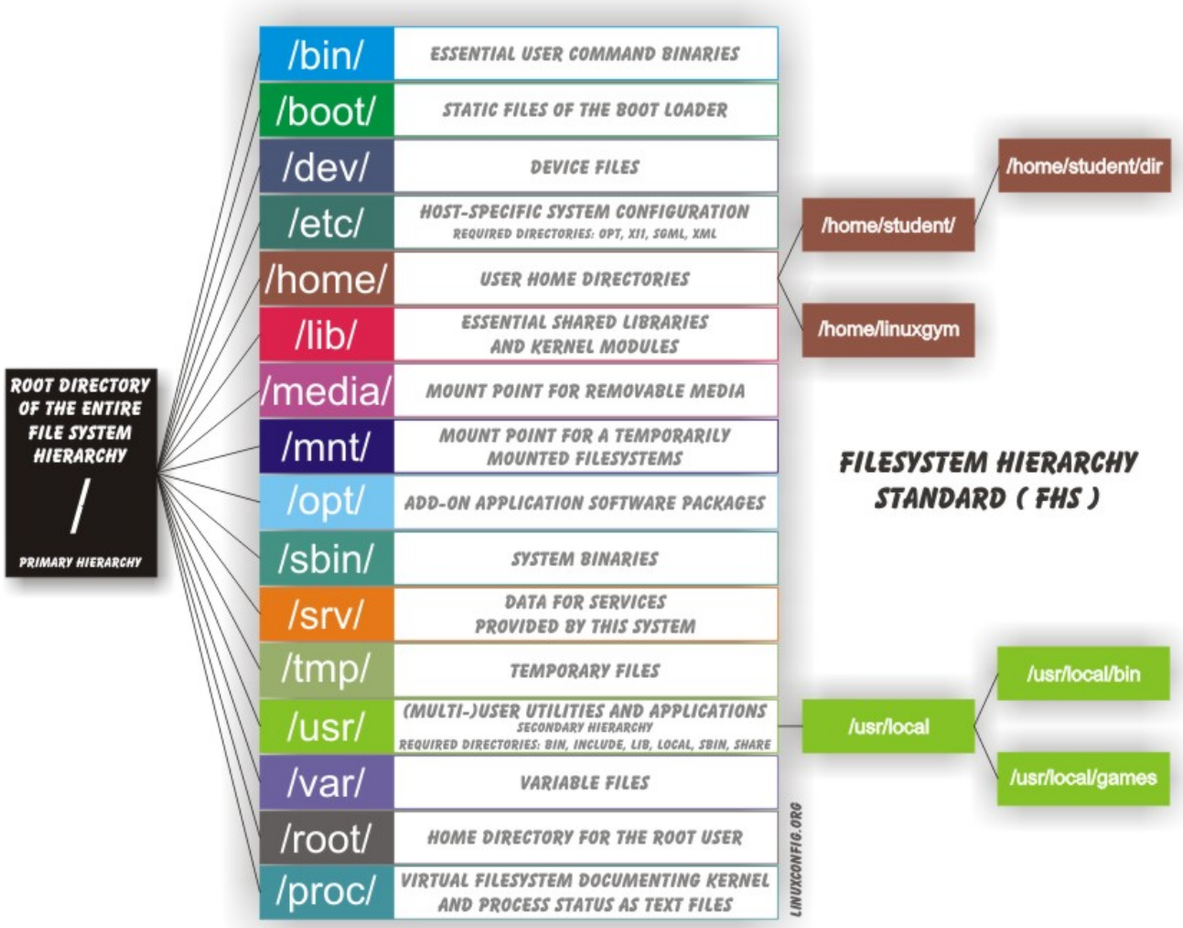
Method of storing and organizing arbitrary collections of data in a human-usable form. Different types of filesystems supported by Linux:

- Conventional disk filesystems: **`ext3`**, **`ext4`**, **`XFS`**, **`Btrfs`**, **`JFS`**, **`NTFS`**, **`vfat`**, **`exfat`**, etc.
- Flash storage filesystems: **`ubifs`**, **`jffs2`**, **`yaffs`**, etc.
- Database filesystems
- Special purpose filesystems: **`procfs`**, **`sysfs`**, **`tmpfs`**, **`squashfs`**, **`debugfs`**, **`fuse`**, etc.

Partitions - subsection of physical storage media. Container in which filesystem resides. Filesystem can span more than one partition if one uses **symbolic links**.

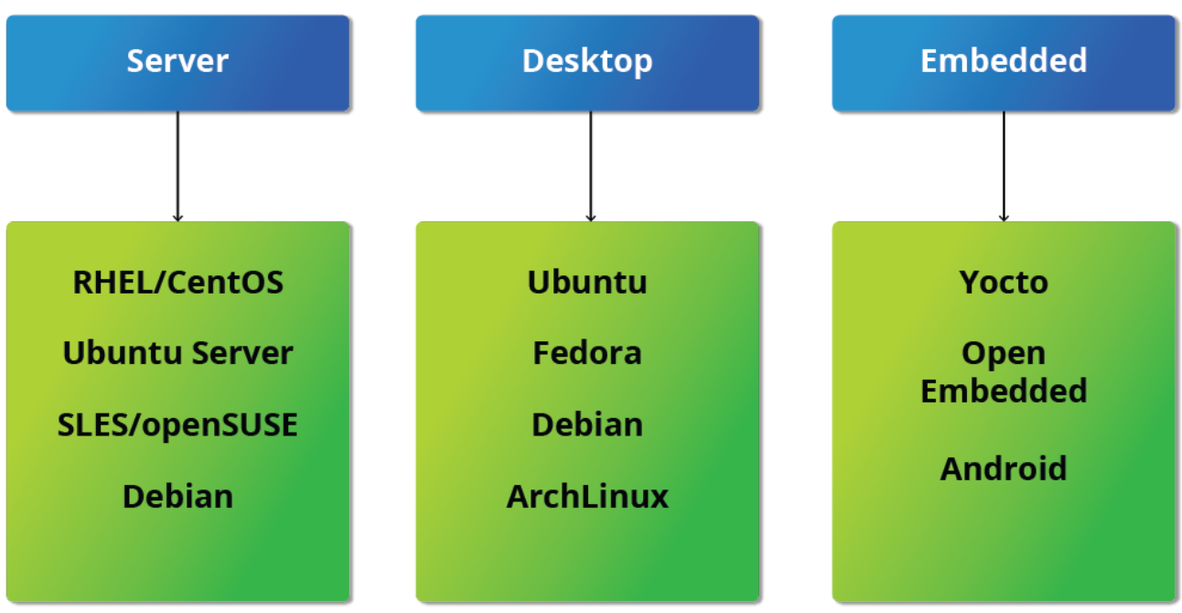
Filesystem Hierarchy Standard

> Linux uses the '/' character to separate paths (as sis UNIX unlike Windows, which uses '\') and does not have drive letters. Multiple drives and/or partitions are mounted as directories in the single filesystem.



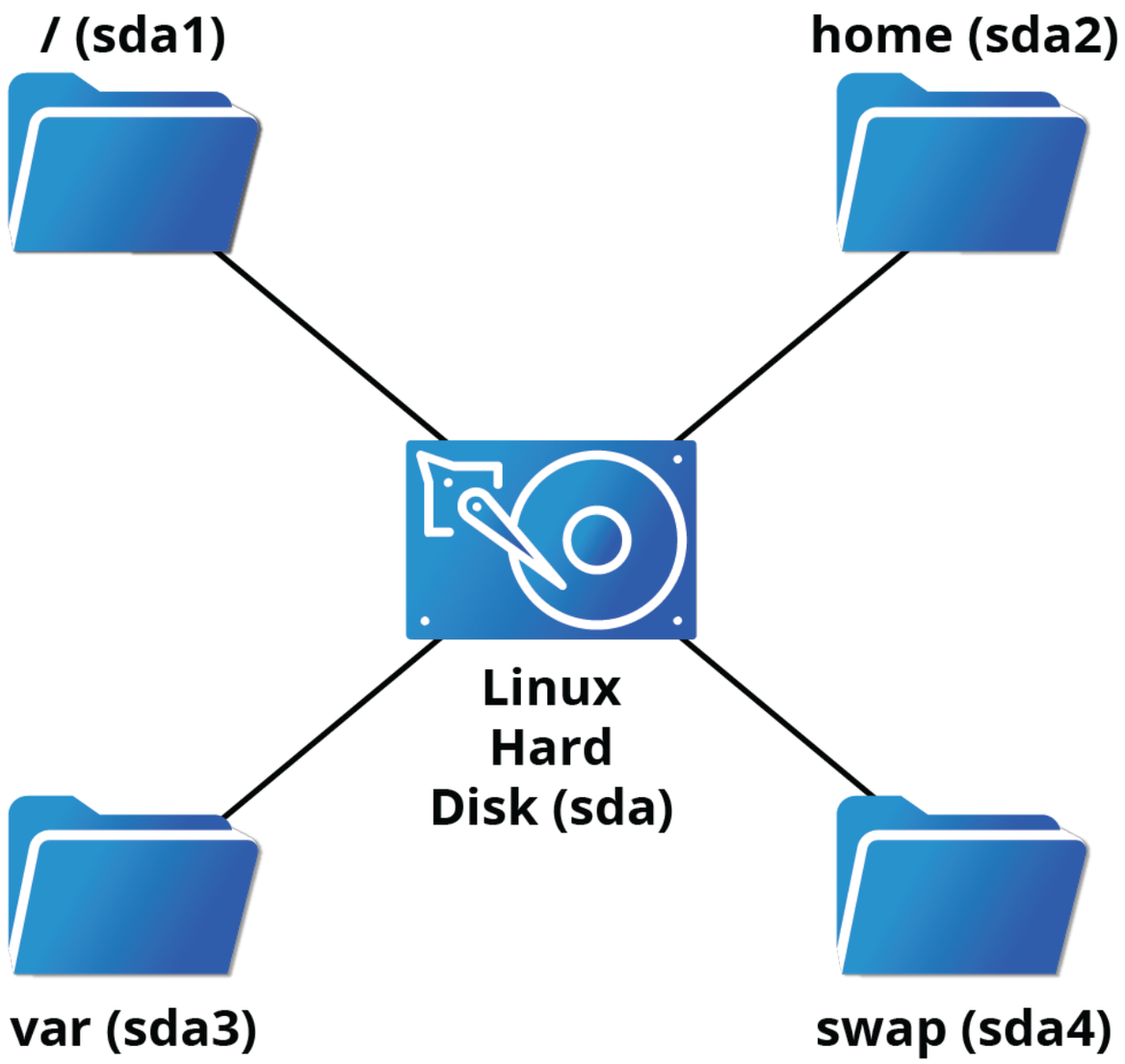
All Linux filesystems - case-sensitive. Many distributions distinguish between core utilities needed for proper system operation and other programs, and place the latter in directories under **/usr**.

Choosing Linux distribution



Partitions in Linux Hard Disk

> Default layout - either all space dedicated to normal files on one big partition and a smaller swap partition or with separate partitions for some space-sensitive areas
> like **/home** and **/var**.



Linux installation

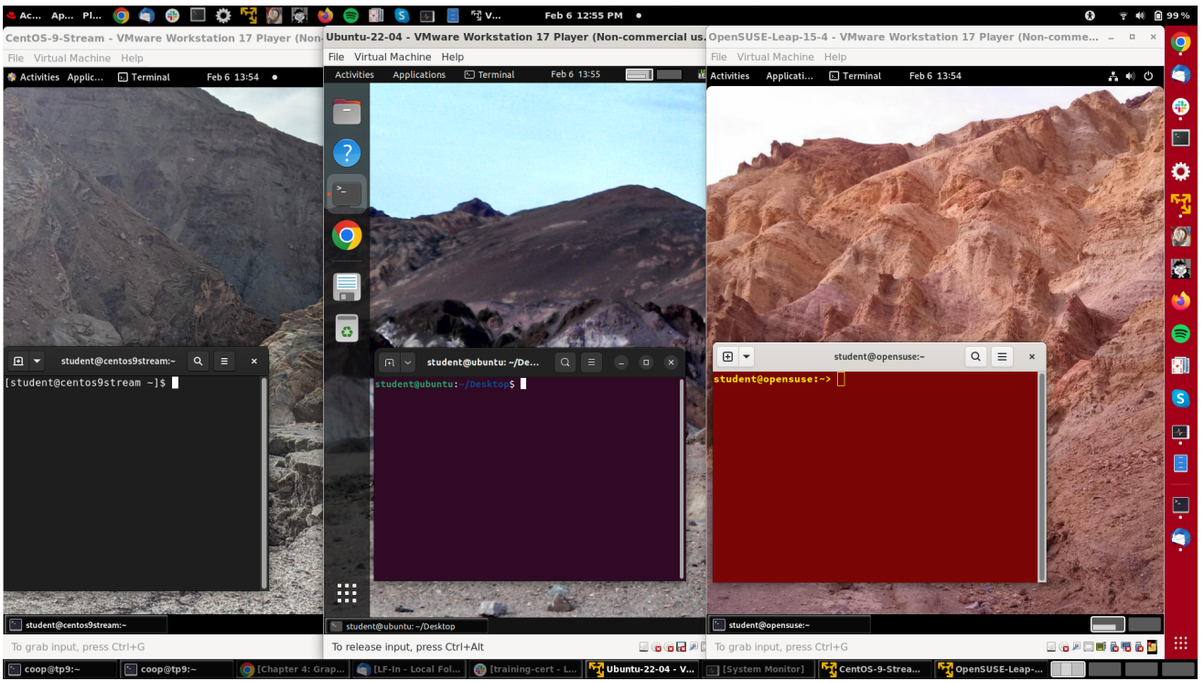
> Linux distributions are provided on removable media such as **USB drives** and **CDs or DVDs**.
> Many installers can do an installation completely automatically, using a **configuration file** to specify installation options. This file is called a **Kickstart file for Red Hat-based systems**, an **AutoYAST profile for SUSE-based systems**, and a **Preseed file for Debian-based systems**.

Chap4 - Graphical Interface

Graphical Desktop

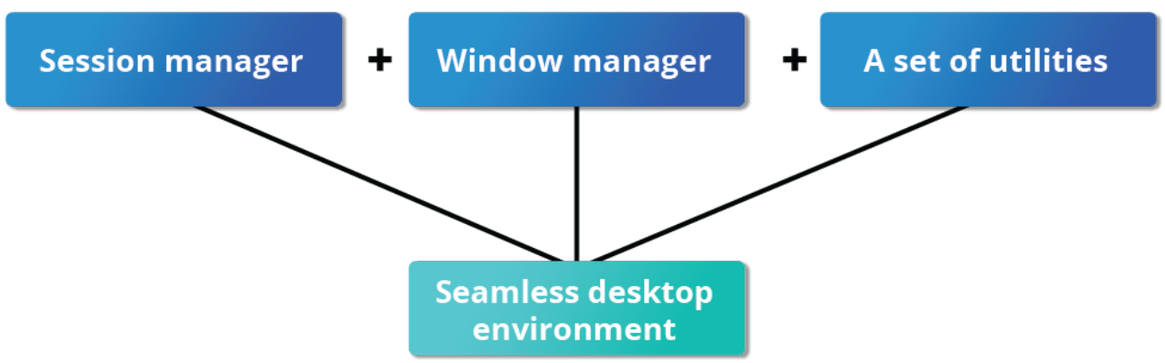
> **CLI**(Command Line Interface) - remember which programs and commands are used to perform tasks, and how to quickly and accurately obtain more information about their use and options.
> **GUI**(Graphical User Interface) - allows you to interact with your system through graphical icons and screens.

For repetitive tasks, the CLI is often more efficient, while the GUI is easier to navigate if you do not remember all the details or do something only rarely.



Desktop environment - session manager, which starts and maintains the components of the graphical session, and the window manager, which controls the placement and movement of windows, window title-bars, and controls. Form one unit along with a set of utilities.

Display manager started by running startx from command line, or start the display manager (**gdm**, **kdm**, **xdm**, etc.) manually from the command line.

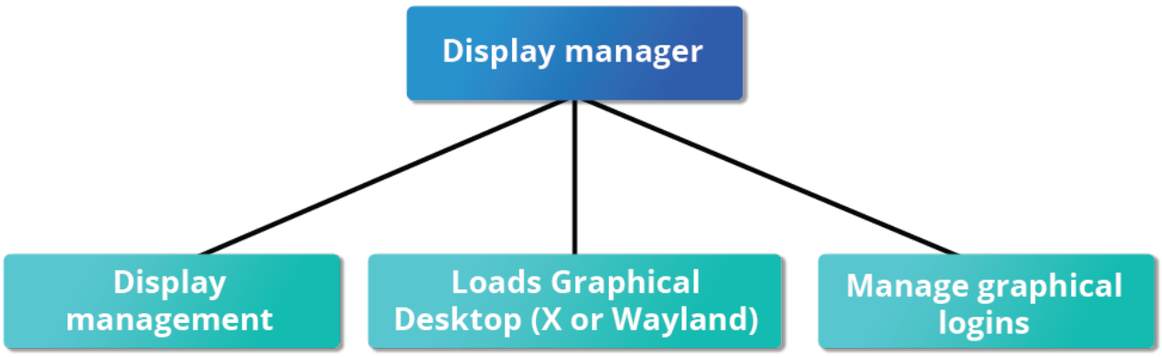


X Window System

Loading the graphical desktop is one of the final steps in the boot process of a Linux desktop. Historically, this was known as the **X Windows System**.

A service called the **Display Manager** keeps track of the displays being provided and loads the **X server**. Applications - **X clients**. Display manager also handles graphical logins and starts the appropriate desktop environment after a user logs in.

A newer system, known as **Wayland**, is gradually superseding it and is the default display system for Fedora, RHEL, and other recent distributions.



GNOME

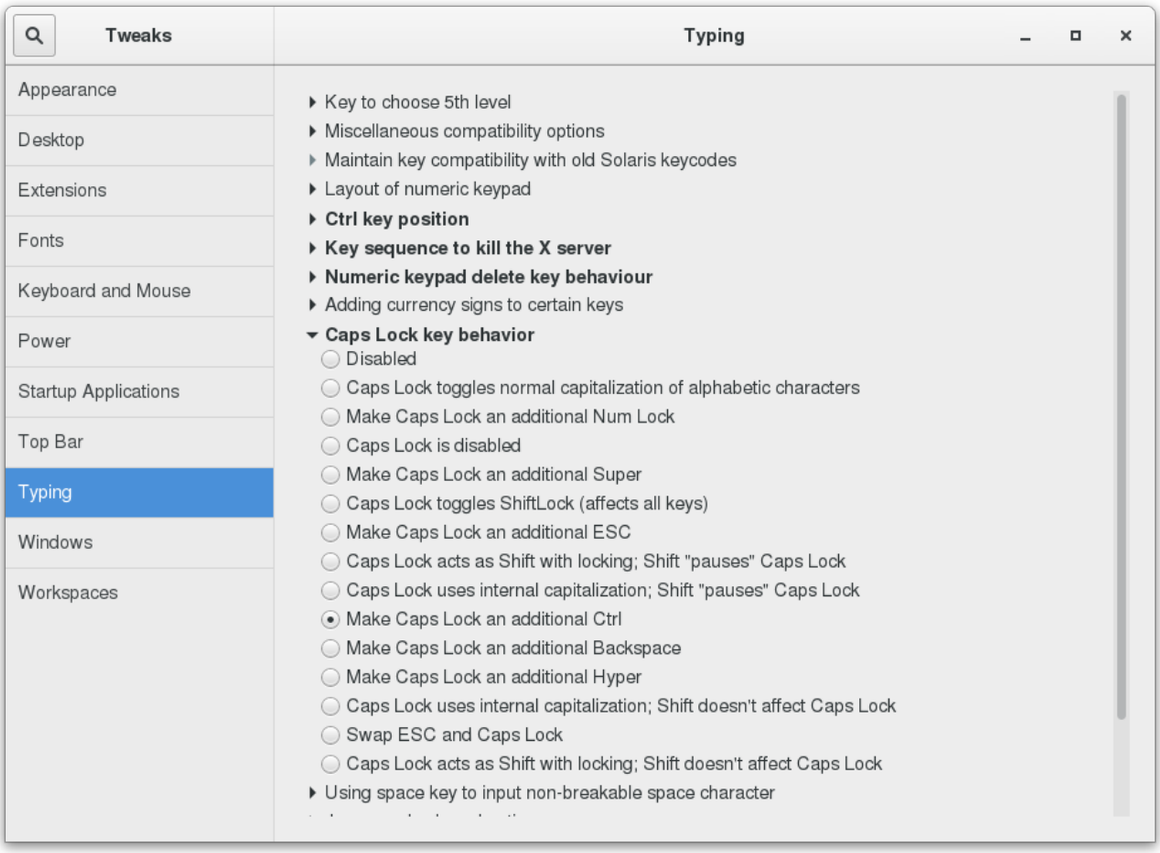
Desktop environment with an easy-to-use graphical user interface. It is bundled as the default desktop environment for most Linux distributions. Has menu-based navigation.

KDE - often used in conjunction with SUSE and openSUSE. Other alternatives Unity , XFCE, and LXDE.
Default display manager for GNOME - **gdm** and for KDE - **kdm**.

---

gnome-tweaks

Standard utility that allows installation of extensions by external parties. run it by hitting **Alt-F2** and then typing in the name.



Keyboard Shortcuts

> Locking screen - **SUPER + L** or **SUPER + esc**
>
> While viewing files - **Ctrl-1** and **Ctrl-2** to switch between Icons and List formats. **Ctrl-H** to show hidden files.
>
> To open the **File Manager** from the command line, on most systems simply type **nautilus**.
>
> The shortcut key to get to the search text box is **CTRL-F**. You can exit the search text box view by clicking the *Search* button or **CTRL-F** again.
>
> Another quick way to access a specific directory is to press **CTRL-L**, which will give you a **Location** text box to type in a path to a directory.
>
> **CTRL-DELETE** or right click on file then move to trash - file moved to **.local/share/Trash/files/**.
> Within trash directory - select the file or directory you want to permanently delete and press **Shift-Delete**.
>
> Nautilus - three formats to view files
>
> Text editors - located in **Accessories** submenu.

Chap5 - System Configuration from the Graphical Interface

System Settings

The **System Settings** panel allows you to control most of the basic configuration options and desktop settings, such as **specifying the screen resolution, managing network connections, or changing the date and time of the system**.

Clicking on **Applications** lets you configure the options relevant to many installed programs.

Users icon (which may be under **Details**) - to set values for system users, such as their login picture, password, etc.

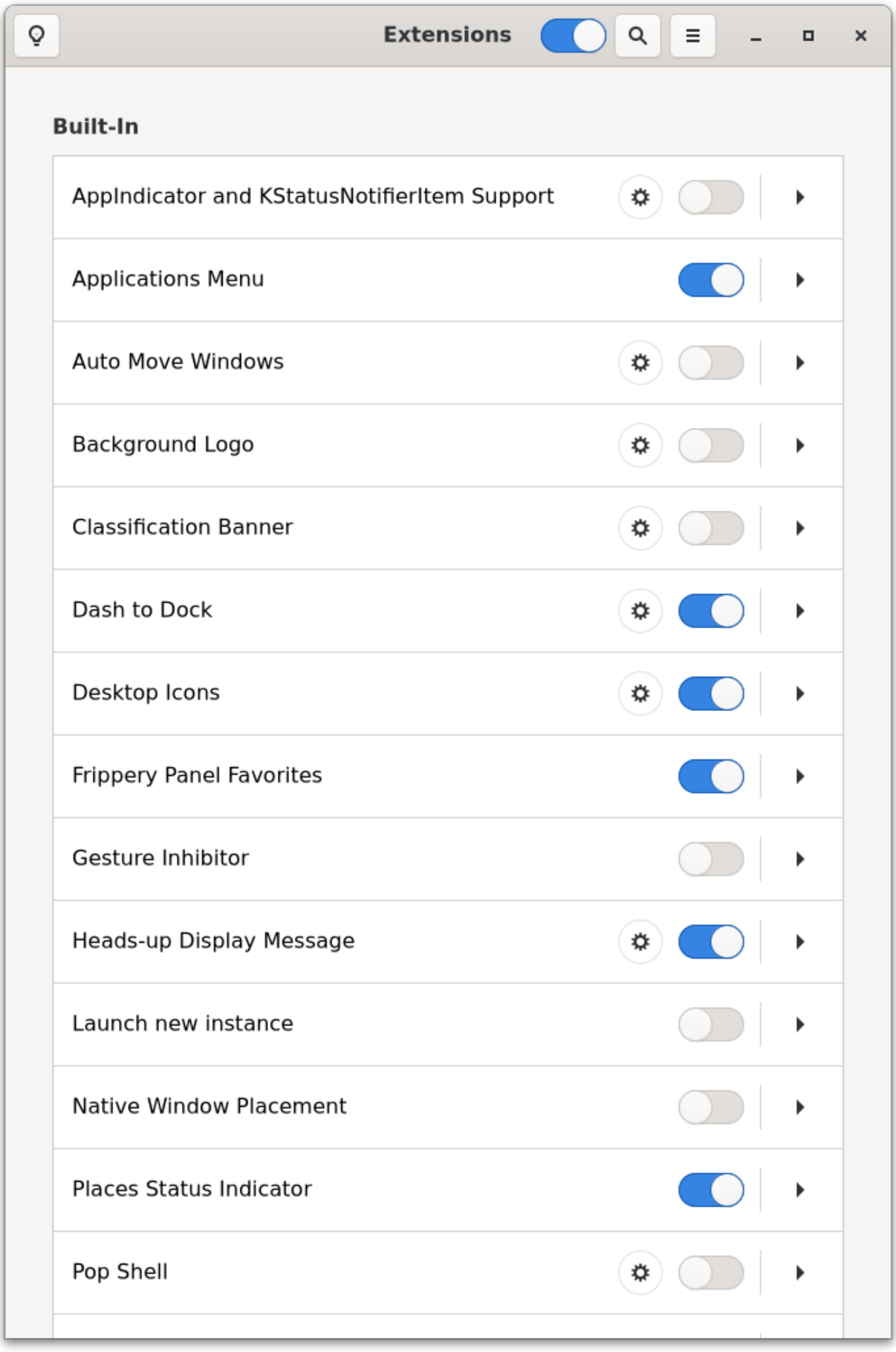
---

gnome-tweaks

Launch a tool called either **gnome-tweaks** (or **gnome-tweak-tool** on older Linux distributions) by doing **Alt-F2** and typing in the command.

Selecting a **theme**, configuring **extensions** which you can get from your distribution or download from the Internet, control fonts, modify the keyboard layout, and set which programs start when you login.

Extensions now have to be configured using a new app called **gnome-extensions-app**.



Display Settings

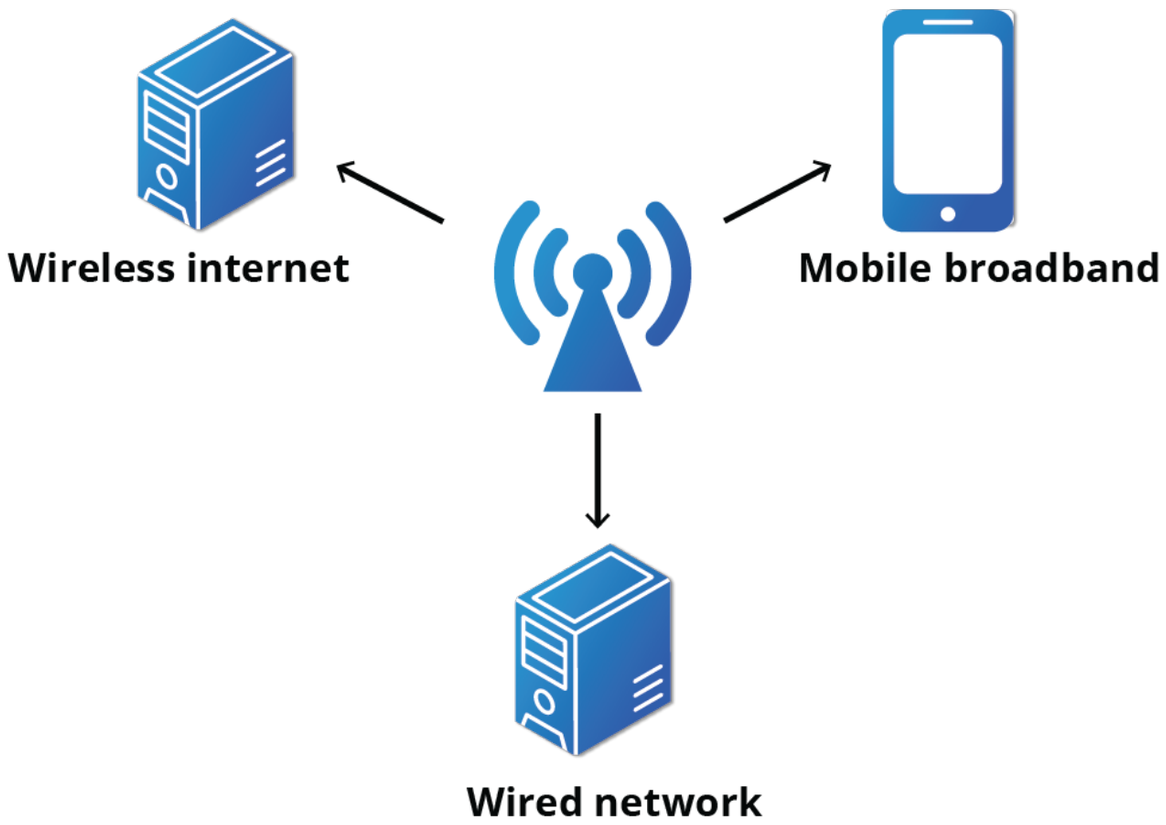
Clicking on **Settings > Displays** (or **Settings > Devices > Displays**) or by right-clicking anywhere on the desktop and selecting **Display Settings**.
On systems utilizing the X Window system, the server which actually provides the GUI, uses **/etc/X11/xorg.conf** as its configuration file if it exists.

Date and Time Settings

> Linux always uses **Coordinated Universal Time (UTC)** for its own internal timekeeping. **Date & Time** window in the System Settings Menu.
>
> The **Network Time Protocol (NTP)** is the most popular and reliable protocol for setting the local time by consulting established Internet servers. Linux distributions always come with a working NTP setup, which refers to specific time servers run or relied on by the distribution. This means that no setup, beyond "on" or "off", is generally required for network time synchronization.

Network Configuration

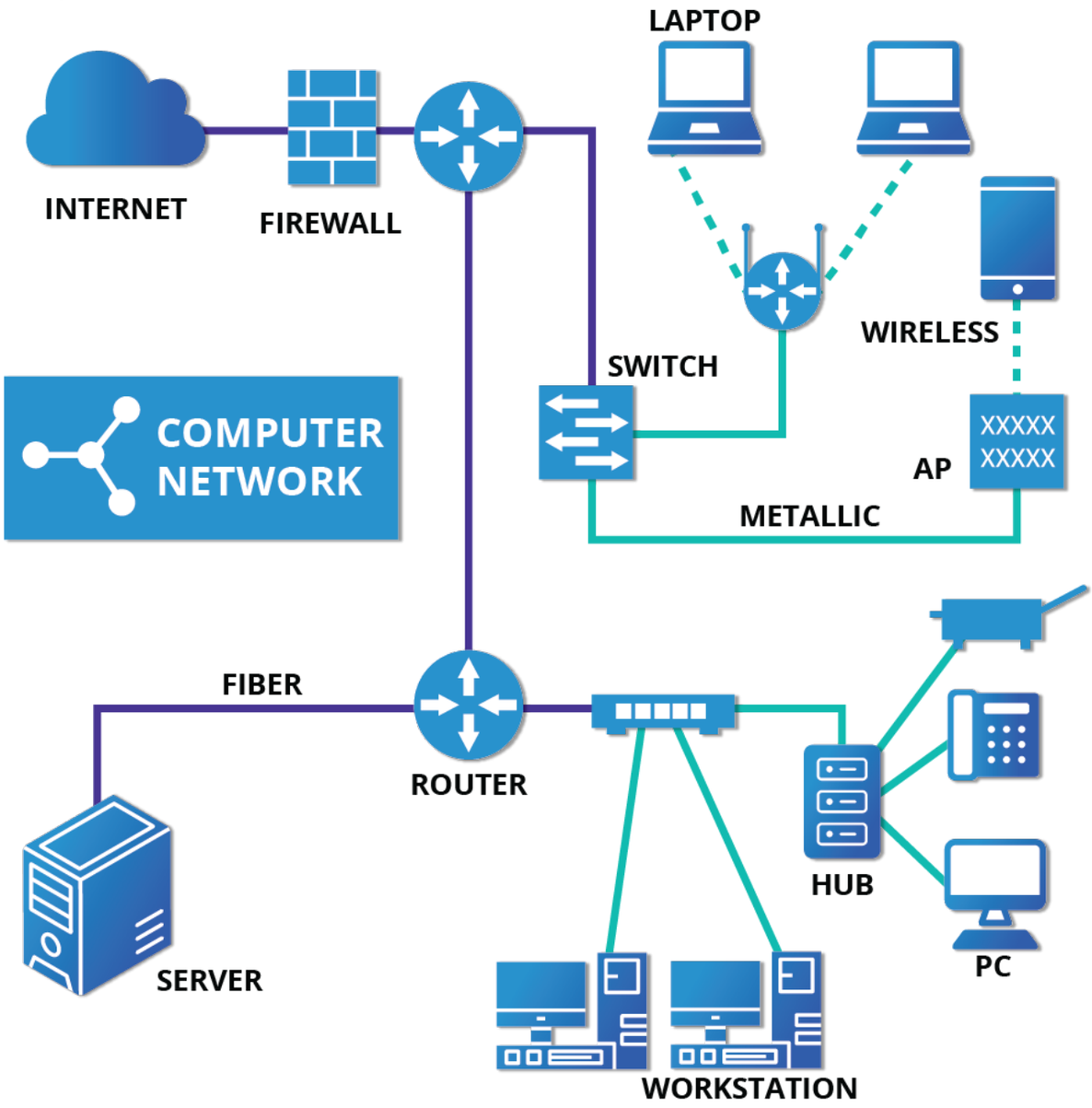
> All Linux distributions have network configuration files, but file formats and locations can differ from one distribution to another.
>
> **Network Manager** was developed to make things easier and more uniform across distributions. It can list all available networks (both wired and wireless), allow the choice of a wired, wireless, or mobile broadband network, handle passwords, and set up Virtual Private Networks (VPNs).



Connections

> **Wired connections** usually do not require complicated or manual configuration. The hardware interface and signal presence are automatically detected, and then Network Manager sets the actual network settings via **Dynamic Host Configuration Protocol (DHCP)**.
>
> For **static** configurations that do not use DHCP, manual setup can also be done easily through Network Manager. Ethernet Media Access Control (MAC) address can also be

changed.



Wireless networks are usually not connected by default. You can view the list of available wireless networks and see which one (if any) you are currently connected to by using Network Manager. You can then add, edit, or remove known wireless networks, and also specify which ones you want connected by default when present.

Set up a **mobile broadband connection** with Network Manager, which will launch a wizard to set up the connection details for each connection. Once the configuration is done, the network is configured automatically each time the broadband network is attached.

It supports many VPN technologies, such as native IPSec, Cisco OpenConnect (via either the Cisco client or a native open source client), Microsoft PPTP, and OpenVPN.

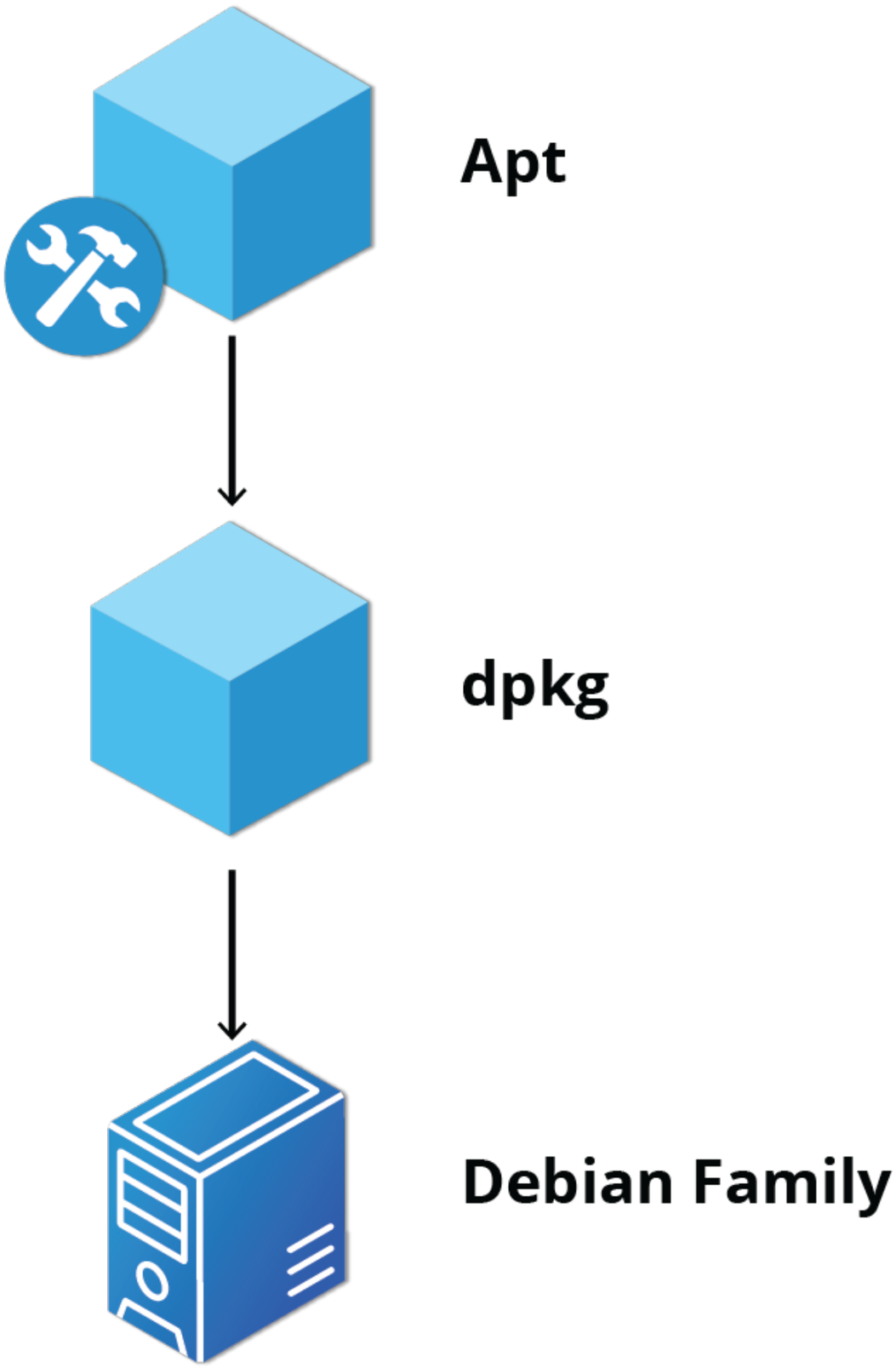
Installing and Updating Software

> Each package in a Linux distribution provides one piece of the system, such as the Linux kernel, the **C** compiler, utilities for manipulating text or configuring the network, or for your favorite web browsers and email clients. All systems have a lower-level utility that handles the details of unpacking a package and putting the pieces in the right places.
>
> **Debian packaging system** (used by systems such as Ubuntu as well) and **RPM packaging systems** (which are used by both Red Hat and SUSE family systems). These are the main ones in use, although others work well for other distributions which have a smaller user base, such as **Archlinux and Gentoo**.

Debian Packaging

> **dpkg** is the underlying package manager for these systems. It can install, remove, and build packages. Unlike higher-level package management systems, it does not automatically download and install packages and satisfy their dependencies.
>
> For Debian-based systems, the higher-level package management system is the **Advanced Package Tool** (APT) system of utilities. Generally, while each distribution within the Debian family uses APT, it creates its own user interface on top of it (for example, apt and apt-get, synaptic, gnome-software, Ubuntu Software Center, etc).
>
> Although apt repositories are generally compatible with each other, the software they contain generally is not. Therefore, most repositories target a particular distribution (like Ubuntu), and often software distributors ship with multiple repositories to support multiple

distributions.

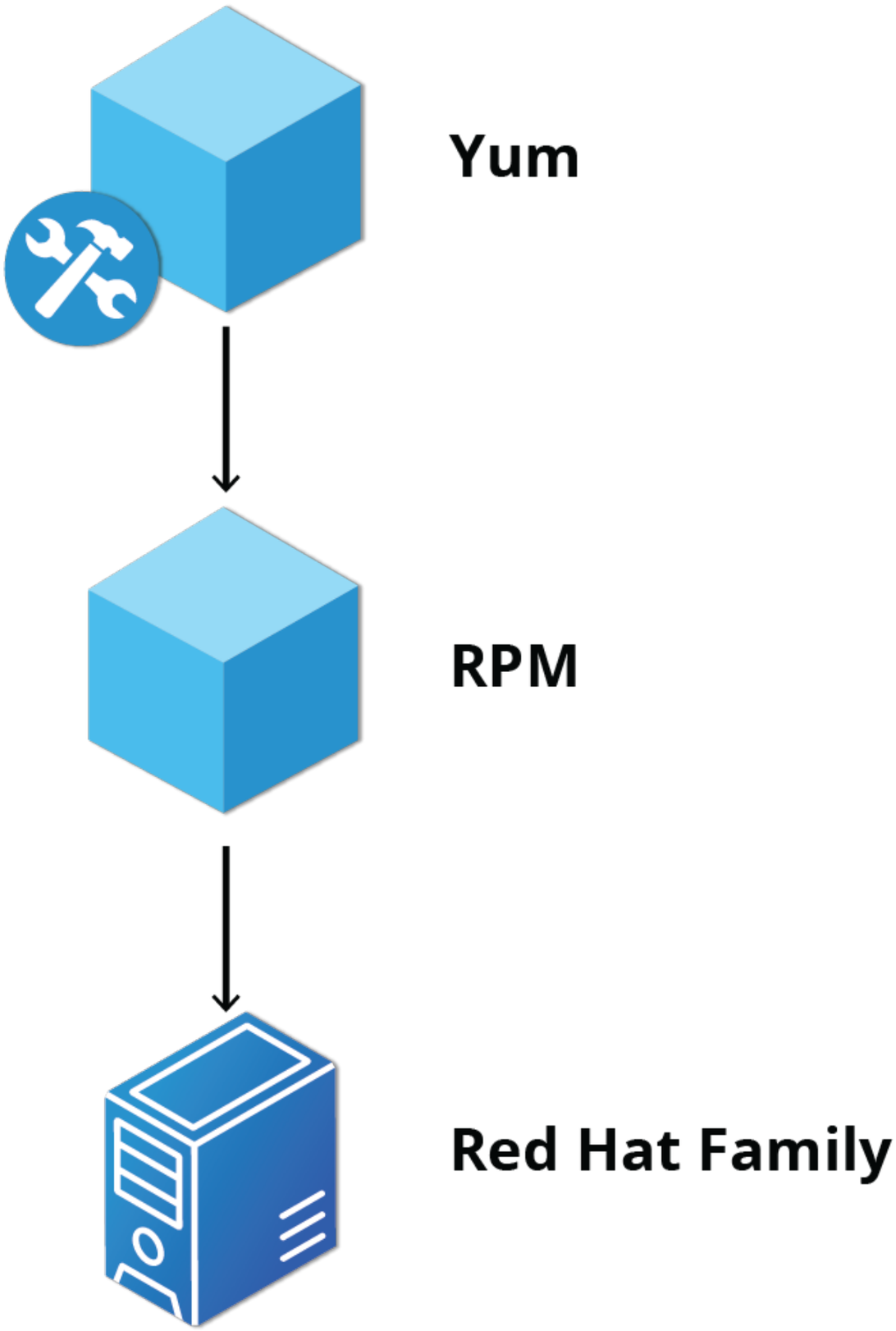


Red Hat Package Manager (RPM)

Red Hat Package Manager (RPM) is the other package management system popular on Linux distributions. It was developed by Red Hat and adopted by a number of other distributions, including Fedora, CentOS, SUSE/openSUSE, Oracle Linux, and others.

The higher-level package manager differs between distributions. **Red Hat** family distributions historically use **RHEL/CentOS**, and **Fedora** uses **dnf**, while **SUSE** family

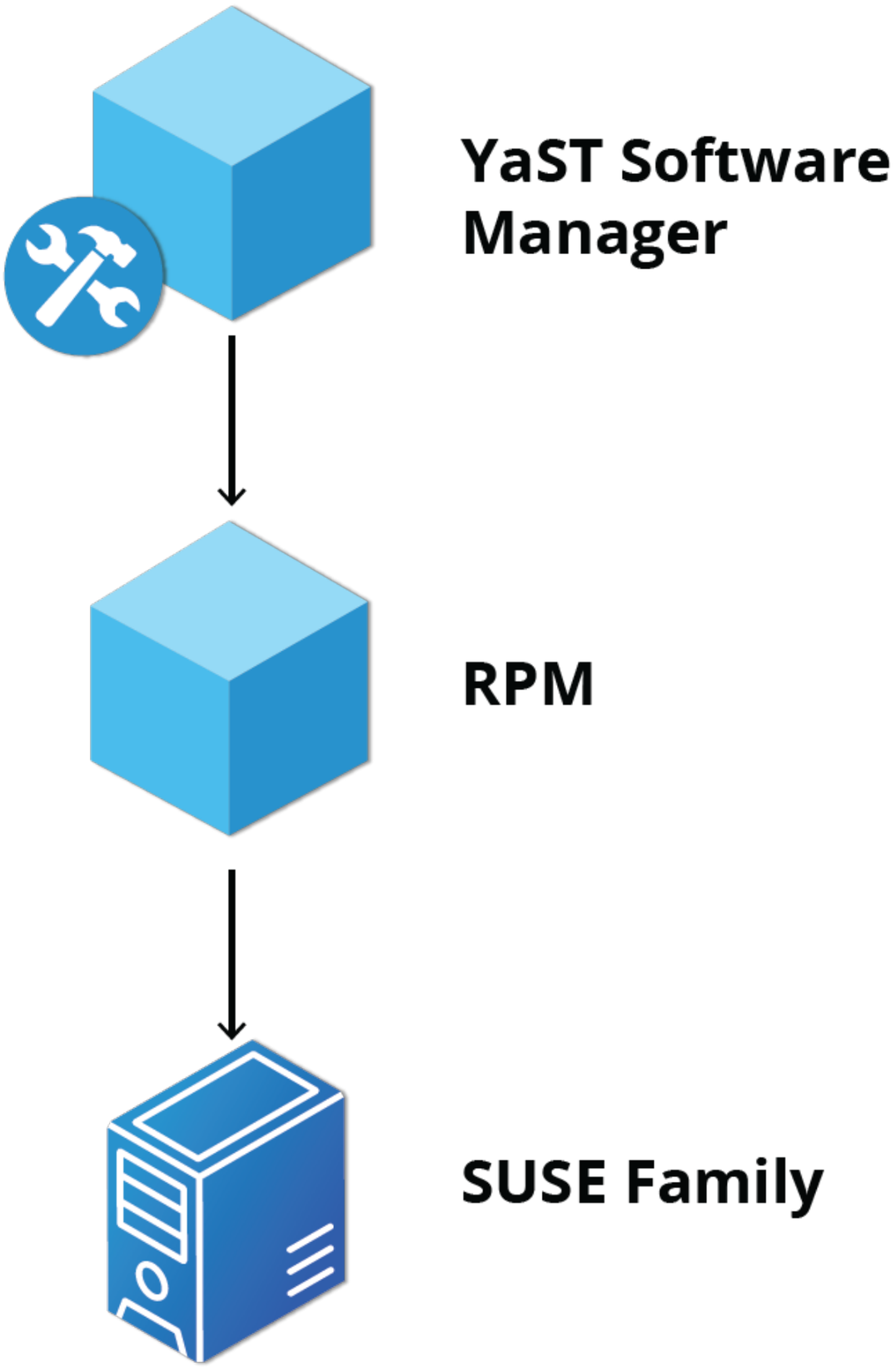
distributions such as openSUSE also use **RPM but use the zypper interface**.



openSUSE’s YaST Software Management

The Yet another Setup Tool (YaST) software manager is similar to other graphical package managers. It is an RPM-based application. You can add, remove, or update packages using this application.
To access the YaST software manager:

1. Click **Activities**
2. In the **Search** box, type **YaST**
3. Click the YaST icon
4. Click **Software Management**

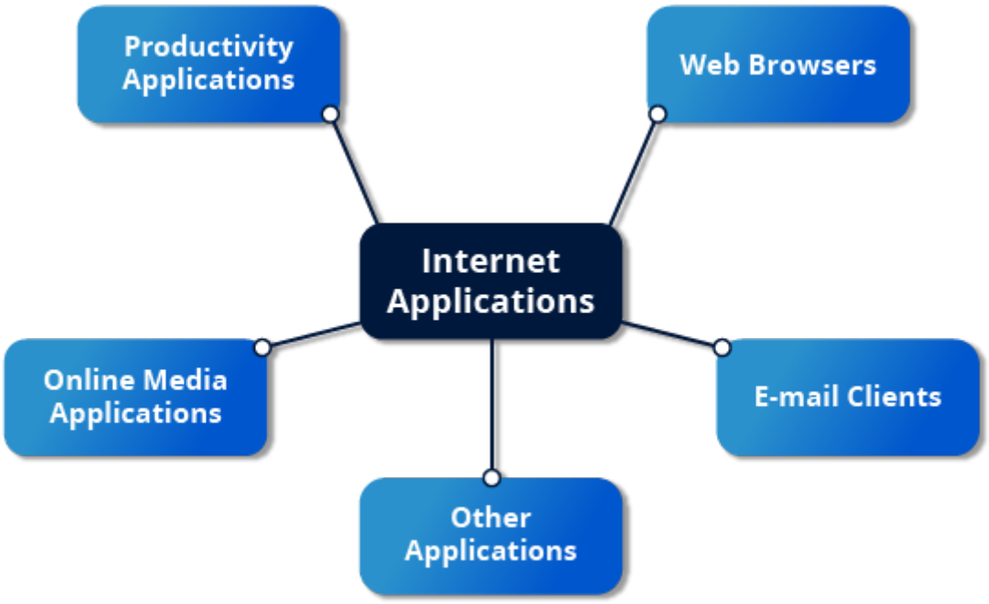


Chap6 - Common Applications

Internet Applications

The Internet is a global network that allows users around the world to perform multiple tasks, such as searching for data, communicating through emails and online shopping. Network aware applications that take advantage of the Internet:

- Web browsers
- Email clients
- Streaming media applications
- Internet Relay Chats
- Conferencing software



Web Browsers

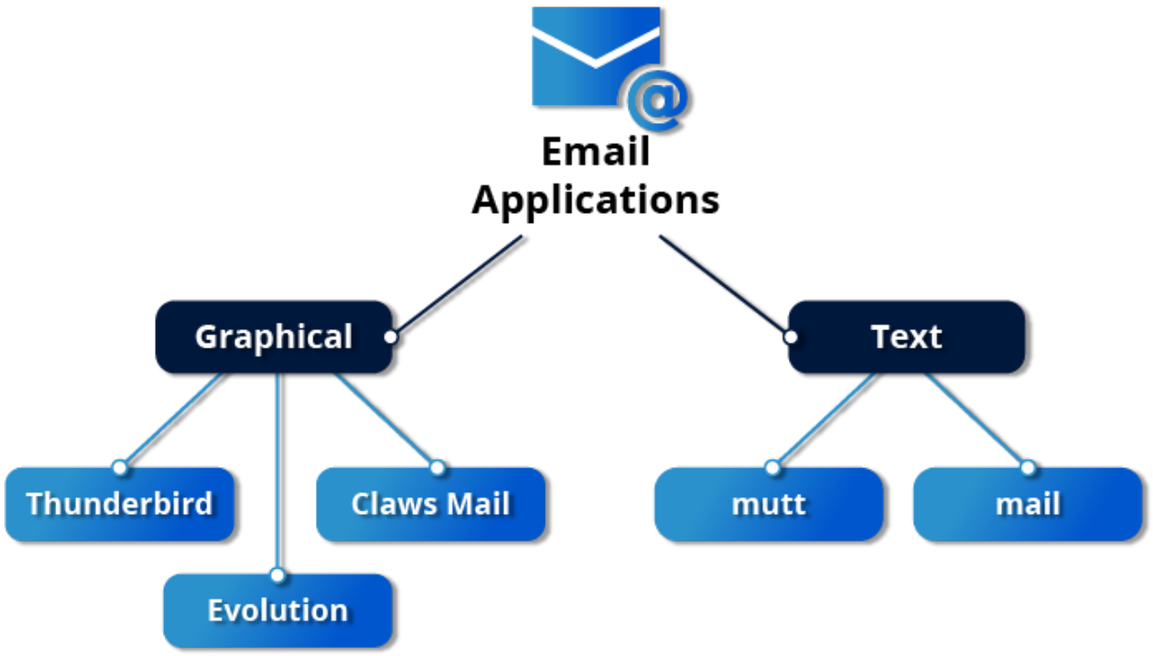
Linux - offers variety of web browsers - both graphical and text-based :

- Firefox
- Google Chrome
- Chromium
- Epiphany (renamed web)
- Konqueror
- linx, lynx, w3m
- Opera

Email Applications

Email applications allow for sending, receiving, and reading messages over the Internet. Most email clients use the **Internet Message Access Protocol (IMAP)** or the older **Post Office Protocol (POP)** to access emails stored on a remote mail server. Most email applications also display HTML (HyperText Markup Language) formatted emails that display objects, such as pictures and hyperlinks.

> The features of advanced email applications include the ability of **importing address books/contact lists, configuration information, and emails from other email applications**.



Other Applications

> | Application | Use |
> | --- | --- |
> | **FileZilla** | Intuitive graphical FTP client that supports FTP, Secure File Transfer Protocol (SFTP), and FTP Secured (FTPS). Used to transfer files to/from (FTP) servers. |
> | **Pidgin** | To access GTalk, AIM, ICQ, MSN, IRC and other messaging networks. |
> | **Hexchat** | To access Internet Relay Chat (IRC) networks. |
>
> 

Productivity Applications

> LibreOffice Components:
>
> - Writer: Word Processing
> - Calc: Spreadsheets
> - Impress: Presentations
> - Draw: Create and edit graphics and diagrams.

Development Applications

- Advanced editors customized for programmers' needs, such as vi and emacs.
- Compilers (such as gcc and clang for programs in C and C++) for every computer language that has ever existed, including very popular new ones such as Golang and Rust.
- Debuggers such as gdb and various graphical front ends to it and many other debugging tools (such as Valgrind).
- Performance measuring and monitoring programs, some with easy to use graphical interfaces, others more arcane and meant to be used only by serious experienced development engineers.
- Source and Revision control systems, such as git (and associated interfaces to it, including GitHub and GitLab) and older systems, such as Apache Subversion.
- Complete Integrated Development Environments (IDE's) such as Eclipse and Visual Studio Code that put all these tools together.

Multimedia Applications

Sound players:

Application	Use
Amarok	Mature MP3 player with a graphical interface, that plays audio and video files, and streams (online audio files). It allows you to create a playlist that contains a group of songs, and uses a database to store information about the music collection.
Audacity	Used to record and edit sounds. Audacity has a simple interface to get you started.
Audacious	Another smart audio media player.
Rhythmbox	Supports a large variety of digital music sources, including streaming Internet audio and podcasts. The application also enables search of particular audio in a library. It supports *smart playlists* with an *automatic update* feature, which can revise playlists based on specified selection criteria.

Movie players:

VLC

Xine

MPlayer

Totem

Movie editors:

Application	Use
Blender	Create 3D animation and design. Blender is a professional tool that uses modeling as a starting point. There are complex and powerful tools for camera capture, recording, editing, enhancing and creating video, each having its own focus.
Cinelerra	Capture, compose, and edit audio/video.
FFmpeg	Record, convert, and stream audio/video. FFmpeg is a format converter, among other things, and has other tools such as **ffplay** and **ffserver**.

GIMP(GNU Image Manipulation Program)

> Graphic editors allow you to create, edit, view, and organize images of various formats, like Joint Photographic Experts Group (JPEG or JPG), Portable Network Graphics (PNG), Graphics Interchange Format (GIF), and Tagged Image File Format (TIFF).
>
> GIMP - feature rich image retouching and editing tool. Features:
>
> - It can handle any image file format.
> - It has many special purpose plugins and filters.
> - It provides extensive information about the image, such as layers, channels, and histograms.

Other Graphic Utilities

> | Graphic Utility | Use |
> | --- | --- |
> | **eog** | Eye of Gnome (eog) is an image viewer that provides slide show capability and a few image editing tools, such as rotate and resize. It can also step through the images in a directory with just a click. |
> | **Inkscape** | Inkscape is an image editor with lots of editing features. It works with layers and transformations of the image. It is sometimes compared to Adobe Illustrator. |
> | **convert** | convert is a command line tool (part of the ImageMagick set of applications) that can modify image files in many ways. The options include file format conversion and numerous image modification options, such as blur, resize, despeckle, etc. |
> | **Scribus** | Scribus is used for creating documents used for publishing and providing a *What You See Is What You Get (WYSIWYG)* environment. It also provides numerous editing tools. |

Chap7 - Command Line Operators

> Advantages of CLI:
>
> - No GUI overhead is incurred.
> - Virtually any and every task can be accomplished while sitting at the command line.
> - You can implement scripts for often-used (or easy-to-forget) tasks and series of procedures.
> - You can sign into remote machines anywhere on the Internet.
> - You can initiate graphical applications directly from the command line instead of hunting through menus.
> - While graphical tools may vary among Linux distributions, the command line interface does not.

> A **terminal emulator** program emulates (simulates) a standalone terminal within a window on the desktop. Ex: gnome, xterm, konsole, terminator.
> Input lines - three elements - Command, Options, Arguments.

Basic command line utilities:

- **`cat`**: used to type out a file (or combine files).
- **`head`**: used to show the first few lines of a file.
- **`tail`**: used to show the last few lines of a file.
- **`man`**: used to view documentation.

---

sudo - provide user with admin privileges when required. Allows to run programs using security privileges of another user (root).

Virtual Terminals

Virtual Terminals (**VT**) are console sessions that use the entire display and keyboard outside of a graphical environment. Such terminals are considered "virtual" because, although there can be multiple active terminals, only one terminal remains visible at a time.

To switch between VTs, press **CTRL-ALT-function** key for the VT. For example, press **CTRL-ALT-F6** for VT 6.

Turning off Graphical Desktop:

`$ sudo systemctl stop gdm` (or **`sudo telinit 3`**)

and restart it (after logging into the console) with:

`$ sudo systemctl start gdm` (or **`sudo telinit 5`**)

Viewing Files

Table: Command Line Utilities Used to View Files

Command	Usage
`cat`	Used for viewing files that are not very long; it does not provide any scroll-back.
`tac`	Used to look at a file backwards, starting with the last line.
`less`	Used to view larger files because it is a paging program. It pauses at each screen full of text, provides scroll-back capabilities, and lets you search and navigate within the file. ***NOTE**: Use / to search for a pattern in the forward direction and **?** for a pattern in the backward direction. An older program named more is still used, but has fewer capabilities: "less is more".*
`tail`	Used to print the last 10 lines of a file by default. You can change the number of lines by doing **`-n 15`** or just **`-15`** if you wanted to look at the last 15 lines instead of the default.
`head`	The opposite of **`tail`**; by default, it prints the first 10 lines of a file.

---

touch is often used to set or update the access, change, and modify times of files. By default, it resets a file's timestamp to match the current time.

Standard File Streams

Table: Standard File Streams

Name	Symbolic Name	Value	Example
standard input	`stdin`	0	keyboard
standard output	`stdout`	1	terminal
standard error	`stderr`	2	log file

Usually, **stdin** is your keyboard, and **stdout** and **stderr** are printed on your terminal. **stderr** is often redirected to an error logging file, while **stdin** is supplied by directing input to come from a file or from the output of a previous command through a pipe. **stdout** is also often redirected into a file. Since **stderr** is where error messages (and warning) are written, usually nothing will go there.

Pipes

Pipe the output of one command or program into another as its input.

```
$ command1 | command2 | command3
```

The above represents what we often call a pipeline, and allows Linux to combine the actions of several commands into one. This is extraordinarily efficient because **command2** and **command3** do not have to wait for the previous pipeline commands to complete before they can begin processing at the data in their input streams; on multiple CPU or core systems, the available computing power is much better utilized and things get done quicker.
Furthermore, there is no need to save output in (temporary) files between the stages in the pipeline, which saves disk space and reduces reading and writing from disk, which often constitutes the slowest bottleneck in getting something done.

Packate Management Systems

The core parts of a Linux distribution and most of its add-on software are installed via the **Package Management System**. Each package contains the files and other instructions needed to make one software component work well and cooperate with the other components that comprise the entire system. Packages can depend on each other.

There are two broad families of package managers widely deployed: those based on Debian and those which use **RPM** as their low-level package manager.

Both package management systems operate on two distinct levels: a low-level tool (such as **dpkg** or **rpm**) takes care of the details of unpacking individual packages, running scripts, getting the software installed correctly, while a high-level tool (such as **apt**, **dnf**, or **zypper**) works with groups of packages, downloads packages from the vendor, and figures out dependencies.

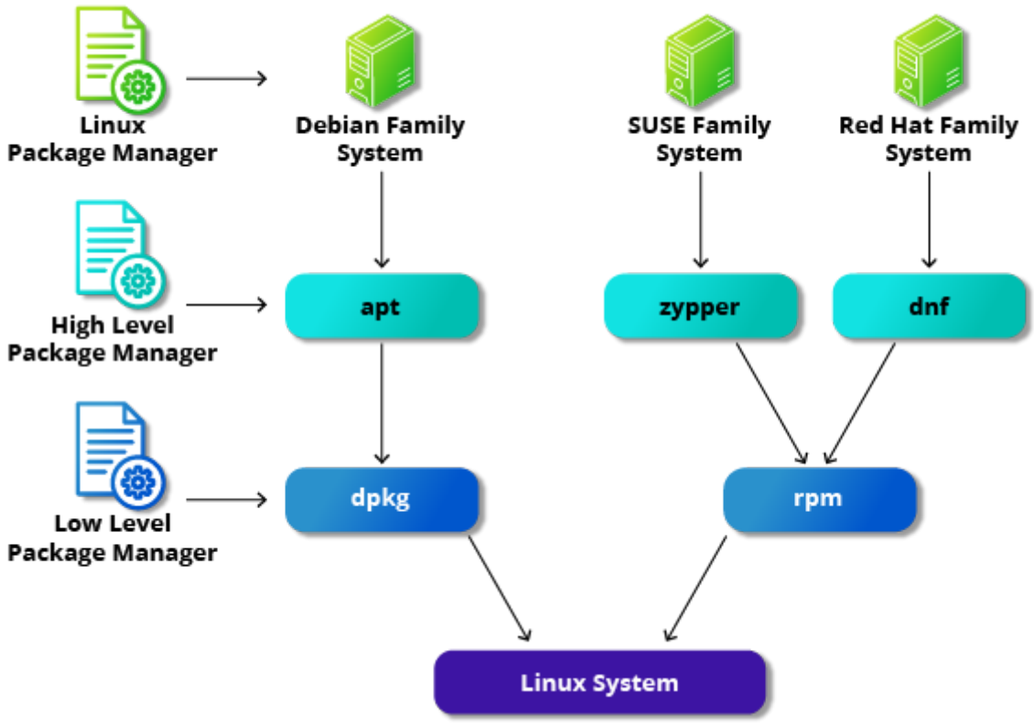
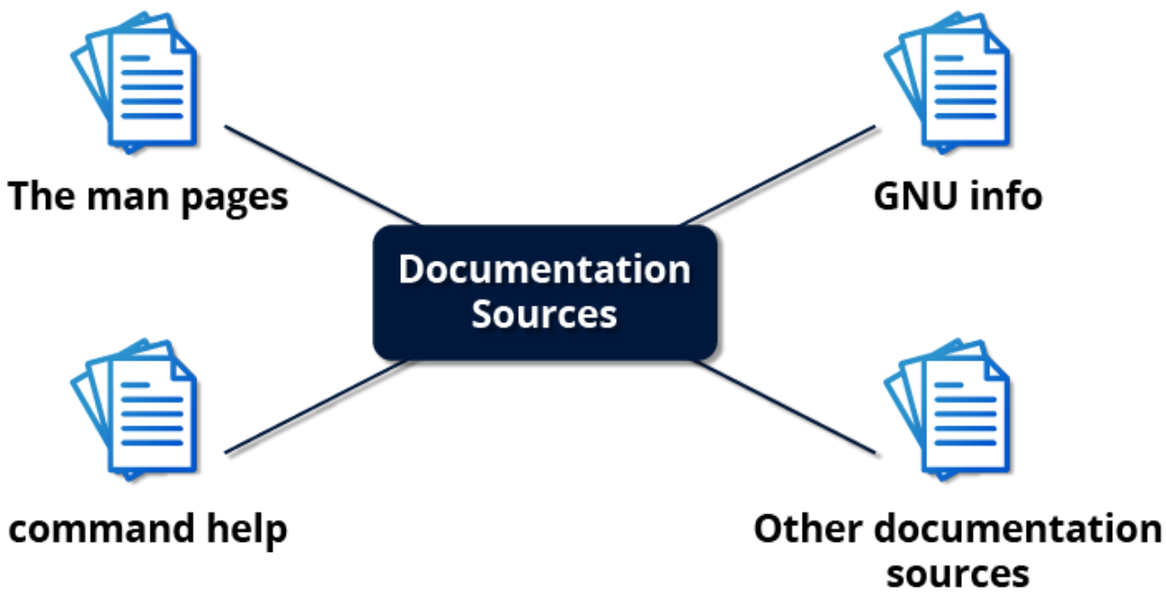


Table: Basic Packaging Commands

Operation	rpm	deb
Install package	`rpm -i foo.rpm`	`dpkg --install foo.deb`
Install package, dependencies	`dnf install foo`	`apt install foo`
Remove package	`rpm -e foo.rpm`	`dpkg --remove foo.deb`
Remove package, dependencies	`dnf remove foo`	`apt autoremove foo`
Update package	`rpm -U foo.rpm`	`dpkg --install foo.deb`
Update package, dependencies	`dnf update foo`	`apt install foo`
Update entire system	`dnf update`	`apt dist-upgrade`
Show all installed packages	`rpm -qa` or `dnf list installed`	`dpkg --list`
Get information on package	`rpm -qil foo`	`dpkg --listfiles foo`
Show packages named `foo`	`dnf list "foo"`	`apt-cache search foo`
Show all available packages	`dnf list`	`apt-cache dumpavail foo`
What package is `file` part of?	`rpm -qf file`	`dpkg --search file`

Chap8 - Finding Linux Documentation

> Linux-based systems draw from a large variety of sources, there are numerous reservoirs of documentation and ways of getting help. Distributors consolidate this material and present it in a comprehensive and easy-to-use manner; this is an important function of any Linux distribution.



---

man pages

> The man pages are the most often-used source of Linux documentation. They provide in-depth documentation about many programs and utilities, as well as other topics, including configuration files and programming APIs for system calls, library routines, and the kernel.
>
> The **man** program searches, formats, and displays the information contained in the man page system. To list all pages on the topic, use the **-f** option. To list all pages that discuss a specific topic use the **–k** option.
>
> The man pages are divided into chapters numbered 1 through 9. In some cases, a letter is appended to the chapter number to identify a specific topic.
>
> With the **-a** parameter, man will display all pages with the given name in all chapters

---

GNU Info System

> This is the GNU project's standard documentation format, which it prefers as an alternative to **man**. The Info System is basically free-form, and supports linked subsections. Functionally, **info** resembles man in many ways. However, topics are connected using links.
>
> Typing **info** with no arguments in a terminal window displays an index of available topics. View help for a particular topic by typing **info topic name**. The system then searches for the topic in all available **info** files.
>
> The topic which you view in an info page is called a **node**. Nodes are essentially sections and subsections in the documentation. You can move between nodes or view each node sequentially. Each node may contain menus and linked subtopics, or items.

Items function like browser links and are identified by an **asterisk** at the beginning of the item name. **Named items** (outside a menu) are identified with **double-colons**(**::**) at the end of the item name.

Table: Keys and Functions

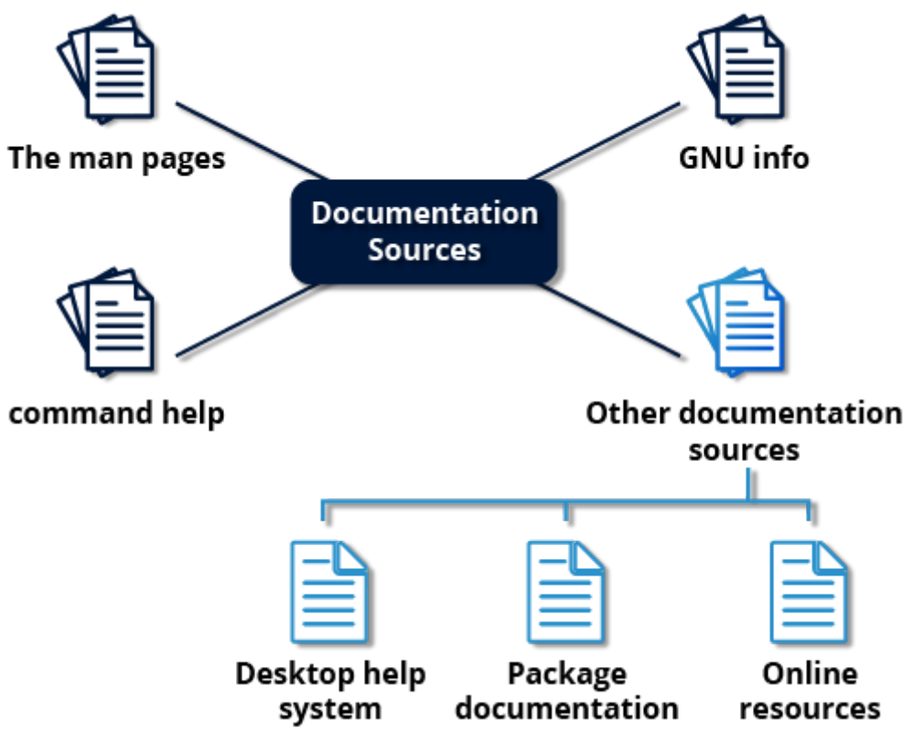
Key	Function
n	Go to the next node
p	Go to the previous node
u	Move one node up in the index

--help option

Most commands have an available short description which can be viewed using the **--help** or the **-h** option along with the command or application. The **--help** option is useful as a quick reference and it displays information faster than the **man** or **info** pages.

help - view synopsis of built-in commands. Performs same function on built-in commands that --help performs on standalone problems.

Other documentation sources



Graphical Help Systems

These programs usually contain custom help for the desktop itself and some of its applications, and will sometimes also include graphically-rendered **info** and **man** pages.

- GNOME: **`gnome-help`** or **`yelp`**
- KDE: **`khelpcenter`**

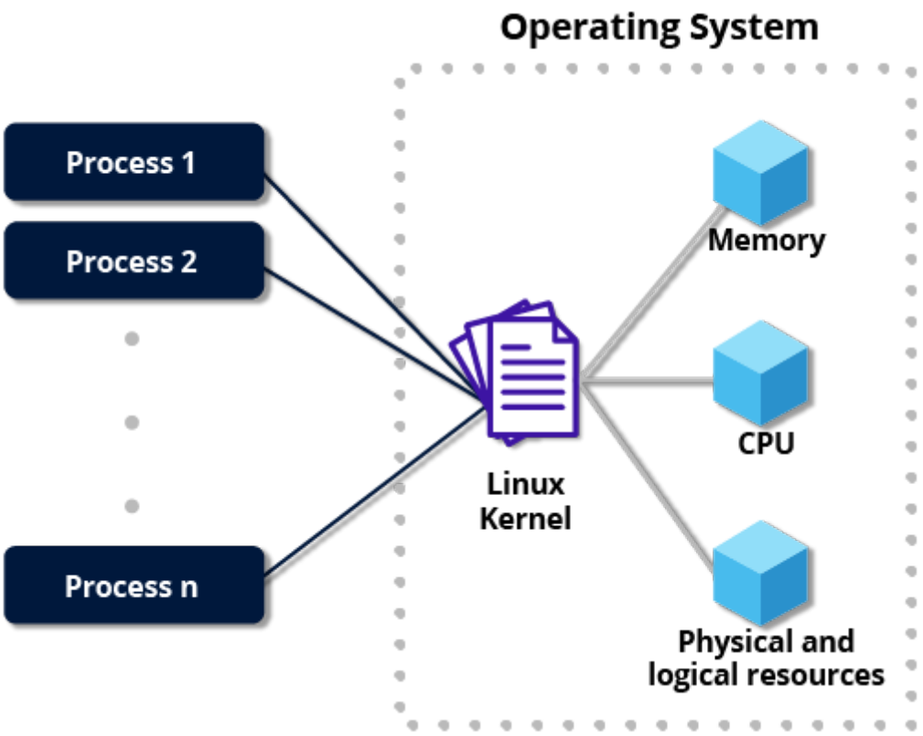
Package Documentation

> Linux documentation is also available as part of the package management system. Usually, this documentation is directly pulled from the upstream source code, but it can also contain information about how the distribution packaged and set up the software.
>
> Such information is placed under the **/usr/share/doc** directory, grouped in subdirectories named after each package.

Chap9 - Processes

Process

> A **process** is simply an instance of one or more related tasks (threads) executing on your computer. A single command may actually start several processes simultaneously.
> Some processes are independent of each other and others are related. A failure of one process may or may not affect the others running on the system.
>
> Processes use many system resources, such as memory, CPU (central processing unit) cycles, and peripheral devices, such as network cards, hard drives, printers, and displays. The operating system (especially the kernel) is responsible for allocating a proper share of these resources to each process and ensuring overall optimized system utilization.



Types of Processes

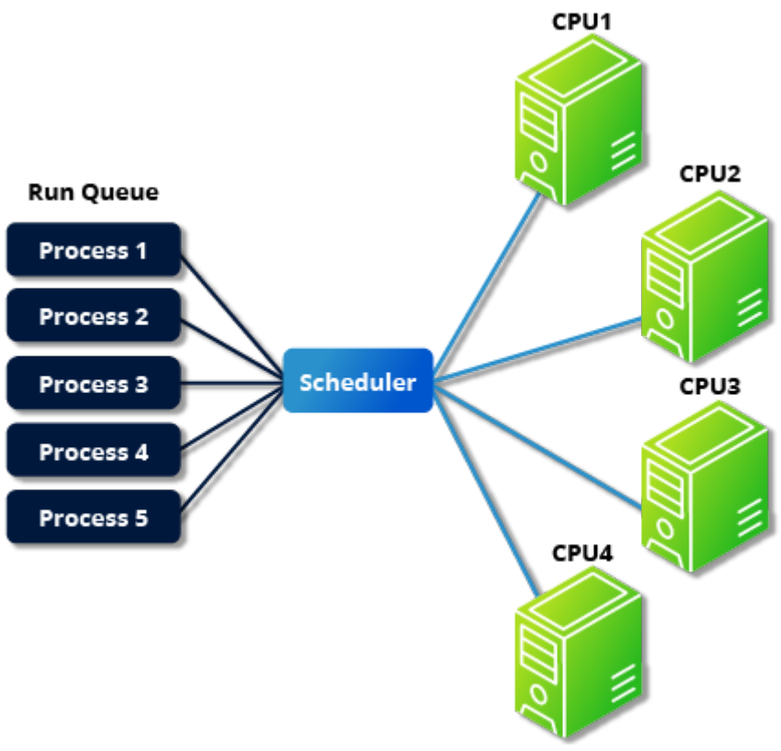
Table: Different Process Types, Descriptions and Examples

Process Type	Description	Examples
Interactive Processes	Need to be started by a user, either at a command line or through a graphical interface such as an icon or a menu selection.	**bash, firefox, top, Slack, Libreoffice**
Batch Processes	Automatic processes which are scheduled from and then disconnected from the terminal. These tasks are queued and work on a **FIFO** (**F**irst-**I**n, **F**irst-**O**ut) basis.	**updatedb, ldconfig**
Daemons	Server processes that run continuously. Many are launched during system startup and then wait for a user or system request indicating that their service is required.	**httpd, sshd, libvirtd, cupsd**
Threads	Lightweight processes. These are tasks that run under the umbrella of a main process, sharing memory and other resources, but are scheduled and run by the system on an individual basis. An individual thread can end without terminating the whole process and a process can create new threads at any time. Many non-trivial programs are multi-threaded.	**dconf-service, gnome-terminal-server**
Kernel Threads	Kernel tasks that users neither start nor terminate and have little control over. These may perform actions like moving a thread from one CPU to another, or making sure input/output operations to disk are completed.	**kthreadd, migration, ksoftirqd**

Process Scheduling and States

A critical kernel function called the **scheduler** constantly shifts processes on and off the CPU, sharing time according to relative priority, how much time is needed and how much has already been granted to a task.

When a process is in a so-called **running** state, it means it is either currently executing instructions on a CPU, or is waiting to be granted a share of time (a time slice) so it can execute. All processes in this state reside on what is called a run queue and on a computer with multiple CPUs, or cores, there is a run queue on each. As noted, the word running is a little misleading as the process may actually be swapped out, waiting its turn to get back in the race.



However, sometimes processes go into what is called a **sleep** state, generally when they are waiting for something to happen before they can resume, perhaps for the user to type something. In this condition, a process is said to be sitting in a wait queue.

There are some other less frequent process states, especially when a process is terminating. Sometimes, a child process completes, but its parent process has not asked about its state. Amusingly, such a process is said to be in a **zombie state**; it is not really alive, but still shows up in the system's list of processes.

Process and Thread IDs

New PIDs are usually assigned in ascending order as processes are born. Thus, PID 1 denotes the **init** process (system initialization process), and succeeding processes are gradually assigned higher numbers.

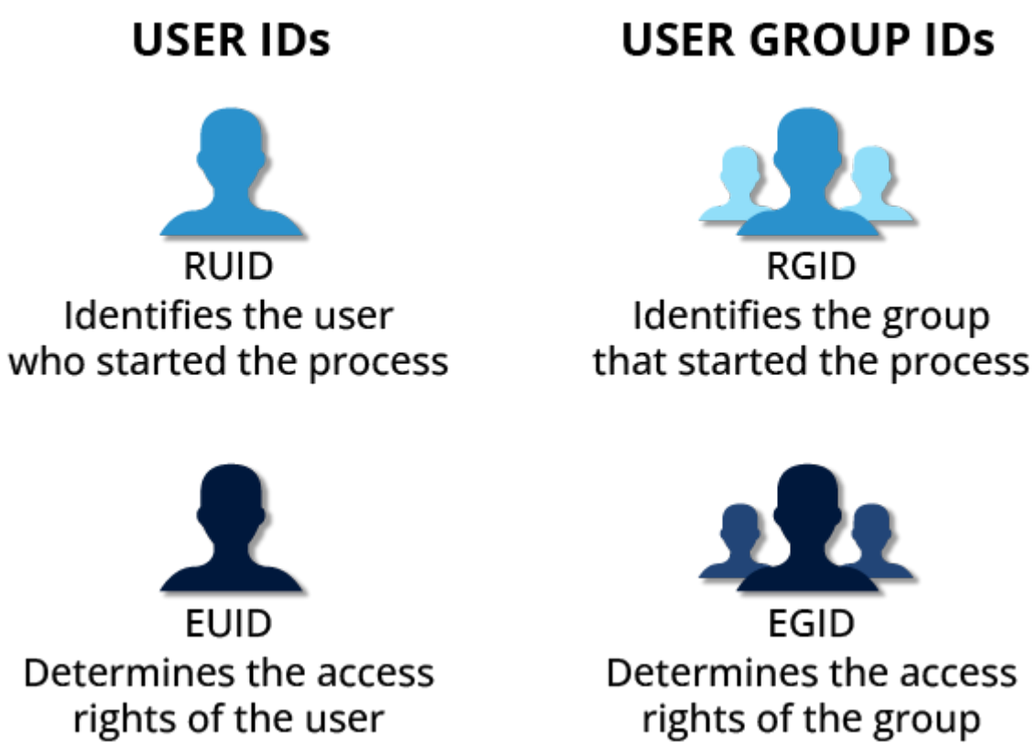
Table: PID Types and Descriptions

ID Type	Description
Process ID (PID)	Unique Process ID number.
Parent Process ID (PPID)	Process (Parent) that started this process. If the parent dies, the PPID will refer to an adoptive parent; on modern kernels, this is **kthreadd** which has PPID=2.
Thread ID (TID)	Thread ID number. This is the same as the PID for single-threaded processes. For a multi-threaded process, each thread shares the same PID, but has a unique TID.

Terminating Process

To terminate a process, you can type **kill -SIGKILL pid** or **kill -9 pid**.
However, you can only kill your own processes; those belonging to another user are off-limits unless you are root.

User and Group IDs



RUID - Real User ID
RGID - Real Group ID

EUID - Effective User ID
EGID - Effective Group ID

The priority for a process can be set by specifying a **nice value**, or niceness, for the process. The lower the nice value, the higher the priority. Low values are assigned to important processes, while high values are assigned to processes that can wait longer.

In Linux, a nice value of **-20** represents the highest priority and **+19** represents the lowest.

Real-time priority can be assigned to time-sensitive tasks - very high priority and makes sure a job gets completed within a well-defined time window.

Load Averages

The **load average** is the average of the load number for a given period of time. It takes into account processes that are:

- Actively running on a CPU.
- Considered runnable, but waiting on the run queue for a CPU to become available.
- Sleeping: i.e. waiting for some kind of resource (typically, I/O) to become available.

The load average can be viewed by running **w**, **top** or **uptime**.

The load average is displayed using three numbers (**0.45**, **0.17**, and **0.12**) in the below screenshot. Assuming our system is a single-CPU system, the three load average numbers are interpreted as follows:

- **`0.45`**: For the last minute the system has been 45% utilized on average.
- **`0.17`**: For the last 5 minutes utilization has been 17%.
- **`0.12`**: For the last 15 minutes utilization has been 12%.

If value = 1.00 - CPU 100% utilized, if value is over 1 - CPU over utilized.

Short-term increases are usually not a problem. A high peak you see is likely a burst of activity, not a new level. For example, at start up, many processes start and then activity settles down. If a high peak is seen in the 5 and 15 minute load averages, it may be cause for concern.

Background and Foreground Process

Foreground jobs run directly from the shell, and when one foreground job is running, other jobs need to wait for shell access, until it is completed.

In such cases, you can run the job in the background and free the shell for other tasks. The background job will be executed at lower priority, which, in turn, will allow smooth execution of the interactive tasks, and you can type other commands in the terminal window while the background job is running.

> Put a job in the background by suffixing **&** to the command, for example: **updatedb &**.
> You can use **CTRL-Z** to suspend a foreground job (i.e., put it in background) and **CTRL-C** to terminate it.
> You can always use the **bg** command to run a suspended process in the background, or the **fg** command to run a background process in the foreground.

> **jobs** - displays all jobs running in the background. Shows job ID, state and command name.
> **jobs -l** - displays same info - along with PID of the background jobs.

---

ps command

> **ps** (process status) provides information about currently running processes keyed by PID.
>
> Periodic update of this status, use **top**.
>
> Without options, **ps** will display all processes running under the current shell.
>
> Use the **- u** option to display information of processes for a specified username.
>
> The command **ps -ef** displays all the processes in the system in full detail.
>
> The command **ps -eLf** goes one step further and displays one line of information for every thread.
>
> **ps aux** displays all processes of all users.
>
> The command **ps axo** allows you to specify which attributes you want to view.
>
> **pstree** displays the processes running on the system in the form of a tree diagram showing the relationship between a process and its parent process and any other processes that it created. Repeated entries of a process are not displayed, and threads are displayed in curly braces.

---

top command

> Use **top** to get constant real-time updates (every two seconds by default), until you exit by typing **q.top**, it clearly highlights which processes are consuming the most CPU cycles and memory.
>
> First line of the **top** output displays - how long system has been up, how many users logged on, load average.
>
> The second line of the **top** output displays the total number of processes, the number of running, sleeping, stopped, and zombie processes. Comparing the number of running processes with the load average helps determine if the system has reached its capacity or perhaps a particular user is running too many processes. The stopped processes should be examined to see if everything is running correctly.
>
> The third line of the **top** output indicates how the CPU time is being divided between the users
> (**us**) and the kernel (**sy**) by displaying the percentage of CPU time used for each.

The percentage of user jobs running at a lower priority (**niceness - ni**) is then listed. Idle mode (**id**) should be low if the load average is high, and vice versa. The percentage of jobs waiting (**wa**) for I/O is listed. Interrupts include the percentage of hardware (**hi**) vs. software interrupts (**si**). Steal time (**st**) is generally used with virtual machines, which has some of its idle CPU time taken for other uses.

The fourth and fifth lines of the **top** output - memory usage, which is divided in two categories:

- Physical memory (RAM) – displayed on line 4.
- Swap space – displayed on line 5.

Each line in the process list of the **top** output displays information about a process. Processes are ordered by highest CPU usage. The following information about each process is displayed:

- Process Identification Number (**PID**)
- Process owner (**USER**)
- Priority (**PR**) and nice values (**NI**)
- Virtual (**VIRT**), physical (**RES**), and shared memory (**SHR**)
- Status (**S**)
- Percentage of CPU (**%CPU**) and memory (**%MEM**) used
- Execution time (**TIME+**)
- Command (**COMMAND**).

Table: What Happens When Pressing Various Keys When Running `top`

Command	Output
h or **?**	Display available interactive keys and their function
t	Display or hide summary information (rows 2 and 3)
m	Display or hide memory information (rows 4 and 5)
l	Show information for each CPU and not just totals
d	Change display update interval
A	Sort the process list by top resource consumers
r	Renice (change the priority of) a specific processes
k	Kill a specific process
f	Enter the **top** configuration screen
o	Interactively select a new sort order in the process list

Use **at** utility program to execute any non-interactive command **at** a specified time.

---

cron

> **cron** is a time-based scheduling utility program. It can launch routine background jobs at specific times and/or days on an ongoing basis. **cron** is driven by a configuration file called **/etc/crontab** (cron table), which contains the various shell commands that need to be run at the properly scheduled times.
>
> **crontab -e** will open the crontab editor to edit existing jobs or to create new jobs.
>
> Each line of a **crontab** file represents a job, and is composed of a so-called **CRON** expression, followed by a shell command to execute.

Table: Fields, Descriptions, Values

Field	Description	Values
MIN	Minutes	0 to 59
HOUR	Hour field	0 to 23
DOM	Day of Month	1-31
MON	Month field	1-12
DOW	Day Of Week	0-6 (0 = Sunday)
CMD	Command	Any command to be executed

> **sleep** suspends execution for at least the specified period of time, which can be given as the number of seconds (the default), minutes, hours, or days. After that time has passed (or an interrupting signal has been received), execution will resume.
>
> ```
> sleep NUMBER[SUFFIX]...
> ```
>
> where **SUFFIX** may be:
>
> - **s** for seconds (the default)
> - **m** for minutes
> - **h** for hours
> - **d** for days.

Chap10 - File Operations

File Systems

> The filesystem is structured like a tree. The tree is usually portrayed as inverted and starts at what is most often called the **root directory**, which marks the beginning of the hierarchical filesystem and is also sometimes referred to as the trunk and simply denoted by */*.
> The hierarchical filesystem also contains other elements in the path (directory names),

which are separated by forward slashes (*/*).

Linux supports a number of native filesystem types, expressly created by Linux developers, such as:

- ext3
- ext4
- squashfs
- btrfs

It also offers implementations of filesystems used on other alien operating systems, such as those from:

- Windows (ntfs, vfat, exfat)
- SGI (xfs)
- IBM (jfs)
- MacOS (hfs, hfs+)

Each filesystem on a Linux system occupies a disk **partition**. One advantage of this kind of isolation by type and variability is that when all available space on a particular partition is exhausted, the system may still operate normally. Furthermore, if data is either corrupted through error or hardware failure, or breached through a security problem, it might be possible to confine problems to an area smaller than the entire system.

Mounting and Unmounting

The **mount** command is used to attach a filesystem (which can be local to the computer or on a network) somewhere within the filesystem tree. The basic arguments are the **device node** and mount point. For example:

```
$ sudo mount /dev/sda5 /home
```

To unmount the partition, the command would be:

```
$ sudo umount /home
```

Executing **mount** without any arguments will show all presently mounted filesystems.

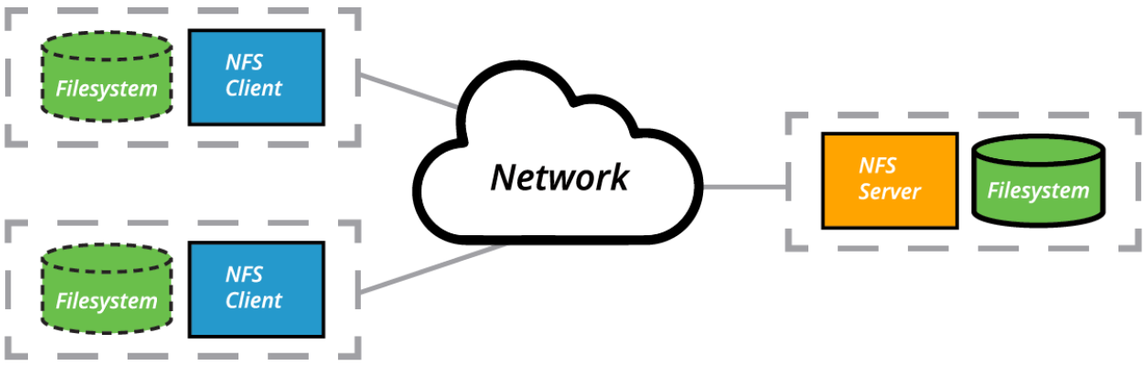
The command **df -Th** (**d**isk **f**ree) will display information about mounted filesystems, including the filesystem type, and usage statistics about currently used and available space.

NFS and Network Filesystems

A network (also sometimes called distributed) filesystem may have all its data on one machine or have it spread out on more than one network **node**. A variety of different filesystems can be used locally on individual machines; a network filesystem can be thought of as a grouping of lower-level filesystems of varying types.

Many system administrators mount remote users' home directories on a server in order to give them access to the same files and configuration files across multiple client systems. This allows the users to log in to different computers, yet still have access to the same files

and resources.



On the server machine, NFS uses **daemons** (built-in networking and service processes in Linux) and other system servers are started at the command line by typing:

```
$ sudo systemctl start nfs
```

The text file **/etc/exports** contains the directories and permissions that a host is willing to share with other systems over NFS.

On the client machine, if it is desired to have the remote filesystem mounted automatically upon system boot, **/etc/fstab** is modified to accomplish this.
Use the **nofail** option in **fstab** in case the NFS server is not live at boot.

/bin and /sbin

The **/bin** directory contains executable binaries, essential commands used to boot the system or in single-user mode, and essential commands required by all system users, such as **cat**, **cp**, **ls**, **mv**, **ps**, and **rm**.
Likewise, the **/sbin** directory is intended for essential binaries related to system administration, such as **fsck** and **ip**.

/proc

Certain filesystems, like the one mounted at **/proc**, are called **pseudo-filesystems** because they have no actual permanent presence anywhere on the disk.
The **/proc** filesystem contains virtual files (files that exist only in memory) that permit viewing constantly changing kernel data. **/proc** contains files and directories that mimic kernel structures and configuration information. It does not contain real files, but runtime

system information.

`/proc/cpuinfo`
`/proc/interrupts`
`/proc/meminfo`
`/proc/mounts`
`/proc/partitions`
`/proc/version`

`/proc` has subdirectories as well, including:

`/proc/<Process-ID-#>`
`/proc/sys`

/dev

The **/dev** directory contains **device nodes**, a type of pseudo-file used by most hardware and software devices, except for network devices. The directory is:

- Empty on the disk partition when it is not mounted
- Contains entries which are created by the **udev** system, which creates and manages device nodes on Linux, creating them dynamically when devices are found. The `/dev` directory contains items such as:
    1. `/dev/sda1` (first partition on the first hard disk)
    2. `/dev/lp1` (second printer)
    3. `/dev/random` (a source of random numbers).

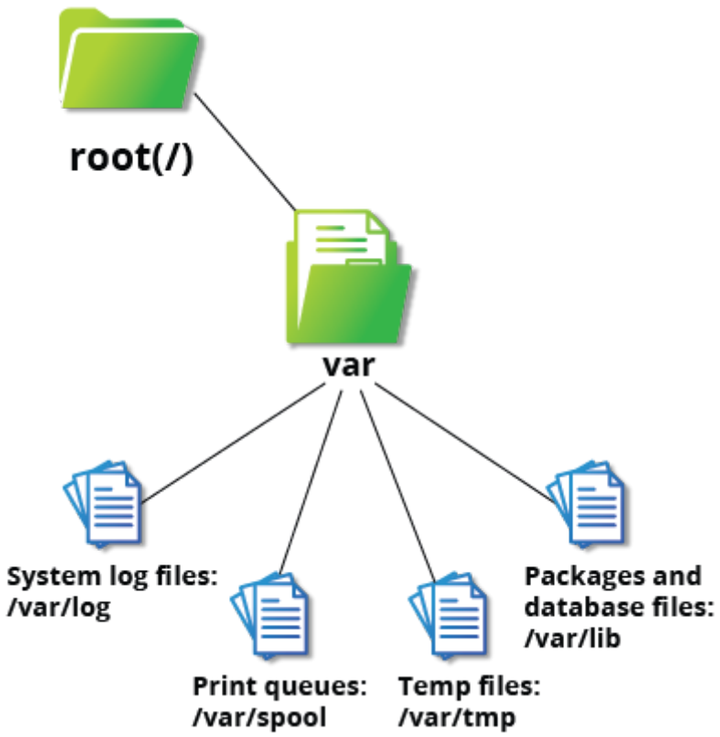
/var

The **/var** directory contains files that are expected to change in size and content as the system is running (var stands for variable), such as the entries in the following directories:

- System log files: `/var/log`
- Packages and database files: `/var/lib`
- Print queues: `/var/spool`
- Temporary files: `/var/tmp`.

The **/var** directory may be put on its own filesystem so that growth of the files can be accommodated and any exploding file sizes do not fatally affect the system. Network services directories such as **/var/ftp** (the FTP service) and **/var/www** (the HTTP web

service) are also found under **/var**.



/etc

The **/etc** directory is the home for system configuration files. It contains no binary programs, although there are some executable scripts.
Files like **passwd**, **shadow** and **group** for managing user accounts are found in the **/etc** directory.

/boot

The **/boot** directory contains the few essential files needed to boot the system. For every alternative kernel installed on the system there are four files:

1. `vmlinuz`
   The compressed Linux kernel, required for booting.
2. `initramfs`
   The initial ram filesystem, required for booting, sometimes called initrd, not initramfs.
3. `config`
   The kernel configuration file, only used for debugging and bookkeeping.
4. `System.map`
   Kernel symbol table, only used for debugging.

The Grand Unified Bootloader (GRUB) files such as **/boot/grub/grub.conf** or **/boot/grub2/grub2.cfg** are also found under the **/boot** directory.

/lib and /lib64

/lib contains libraries (common code shared by applications and needed for them to run) for the essential programs in **/bin** and **/sbin**. These library filenames either start with **ld** or **lib**.
/lib - 32-bit libraries, **/lib64** - 64-bit libraries.
Kernel modules (kernel code, often device drivers, that can be loaded and unloaded without re-starting the system) are located in **/lib/modules/kernel-version-number**.\

Most Linux systems are configured so any removable media are automatically mounted when the system notices something has been plugged in. While historically this was done under the **/media** directory, modern Linux distributions place these mount points under the **/run** directory.

The **/mnt** directory has been used since the early days of UNIX for temporarily mounting filesystems. These can be those on removable media, but more often might be network filesystems, which are not normally mounted. Or these can be temporary partitions, or so-called **loopback** filesystems, which are files which pretend to be partitions.

Table: Additional Directories Under Root Directory

Directory Name	Usage
`/opt`	Optional application software packages
`/sys`	Virtual pseudo-filesystem giving information about the system and the hardware Can be used to alter system parameters and for debugging purposes
`/srv`	Site-specific data served up by the system Seldom used
`/tmp`	Temporary files; on some distributions erased across a reboot and/or may actually be a ramdisk in memory
`/usr`	Multi-user applications, utilities and data

Table: Sub-directories

Directory Name	Usage
`/usr/include`	Header files used to compile applications
`/usr/lib`	Libraries for programs in `/usr/bin` and `/usr/sbin`
`/usr/lib64`	64-bit libraries for 64-bit programs in `/usr/bin` and `/usr/sbin`
`/usr/sbin`	Non-essential system binaries, such as system daemons, and scripts
`/usr/share`	Shared data used by applications, generally architecture-independent
`/usr/src`	Source code, usually for the Linux kernel
`/usr/local`	Data and programs specific to the local machine; subdirectories include `bin`, `sbin`, `lib`, `share`, `include`, etc.
`/usr/bin`	This is the primary directory of executable programs and scripts

---

diff

diff is used to compare files and directories. This often-used utility program has many useful options (see: **man diff**) including:

Table: diff Options and Usage

diff Option	Usage
`-c`	Provides a listing of differences that include three lines of context before and after the lines differing in content
`-r`	Used to recursively compare subdirectories, as well as the current directory
`-i`	Ignore the case of letters
`-w`	Ignore differences in spaces and tabs (white space)
`-q`	Be quiet: only report if files are different without listing the differences

To compare two files, at the command prompt, type:

```
diff [options] <filename1> <filename2>.
```

diff - text files, **cmp** - binary files.

You can compare three files at once using **diff3**, which uses one file as the reference basis for the other two.

patch file contains the deltas (changes) required to update an older version of a file to the new one. The patch files are actually produced by running **diff** with the correct options, as in:

```
$ diff -Nur originalfile newfile > patchfile
```

To apply a patch, you can just do either of the two methods below:

```
$ patch -p1 < patchfile
$ patch originalfile patchfile
```

The first usage is more common, as it is often used to apply changes to an entire directory tree, rather than just one file, as in the second example.

The real nature of a file can be ascertained by using the **file** utility. For the file names given as arguments, it examines the contents and certain characteristics to determine whether the files are plain text, shared libraries, executable programs, scripts, or something else.

Backing Up Data

There are many ways you can back up data or even your entire system. Basic ways to do so include the use of simple copying with **cp** and use of the more robust **rsync**.

rsync is more efficient, because it checks if the file being copied already exists. If the file exists and there is no change in size or modification time, **rsync** will avoid an unnecessary copy and save time. Furthermore, because **rsync** copies only the parts of files that have actually changed, it can be very fast.

cp can only copy files to and from destinations on the local machine (unless you are copying to or from a filesystem mounted using NFS), but **rsync** can also be used to copy files from one machine to another. Locations are designated in the **target:path** form, where **target** can be in the form of **someone@host**. The **someone@** part is optional and used if the remote user is different from the local user.

rsync is very efficient when recursively copying one directory tree to another, because only the differences are transmitted over the network. One often synchronizes the destination directory tree with the origin, using the -r option to recursively walk down the directory tree copying all files and directories below the one listed as the source.

Compressing Data

Table: Methods to Perform Compression

Command	Usage
`gzip`	The most frequently used Linux compression utility
`bzip2`	Produces files significantly smaller than those produced by `gzip`
`xz`	The most space-efficient compression utility used in Linux
`zip`	Is often required to examine and decompress archives from other operating systems

In addition, the **tar** utility is often used to group files in an archive and then compress the whole archive at once.

---

gzip

Table: gzip Usage Examples

Command	Usage
`gzip *`	Compresses all files in the current directory; each file is compressed and renamed with a `.gz` extension.
`gzip -r projectX`	Compresses all files in the `projectX` directory, along with all files in all of the directories under `projectX`.
`gunzip foo`	De-compresses `foo` found in the file `foo.gz`. Under the hood, the `gunzip` command is actually the same as `gzip -d`.

---

bzip2

bzip2 has a syntax that is similar to gzip but it uses a different compression algorithm and produces significantly smaller files, at the price of taking a longer time to do its work. Thus, it is more likely to be used to compress larger files.

Table: bzip2 Usage Examples

Command	Usage
`bzip2 *`	Compresses all of the files in the current directory and replaces each file with a file renamed with a `.bz2` extension.
`bunzip2 *.bz2`	Decompresses all of the files with an extension of `.bz2` in the current directory. Under the hood, `bunzip2` is the same as calling `bzip2 -d`.

---

xz

xz is the most space-efficient compression utility, used for storing archives of the Linux kernel.

Table: xz Usage Examples

Command	Usage
`xz *`	Compresses all of the files in the current directory and replaces each file with one with a **`.xz`** extension.
`xz foo`	Compresses **`foo`** into **`foo.xz`** using the default compression level (-6), and removes **`foo`** if compression succeeds.
`xz -dk bar.xz`	Decompresses **`bar.xz`** into **`bar`** and does not remove **`bar.xz`** even if decompression is successful.
`xz -dcf a.txt b.txt.xz > abcd.txt`	Decompresses a mix of compressed and uncompressed files to standard output, using a single command.
`xz -d *.xz`	Decompresses the files compressed using **`xz`**.

---

zip

While, the **zip** program is rarely used to compress files in Linux, it may be needed to examine and decompress archives from other operating systems.

Table: zip Usage Examples

Command	Usage
`zip backup *`	Compresses all files in the current directory and places them in the **`backup.zip`**.
`zip -r backup.zip ~`	Archives your login directory (~) and all files and directories under it in **`backup.zip`**.
`unzip backup.zip`	Extracts all files in **`backup.zip`** and places them in the current directory.

---

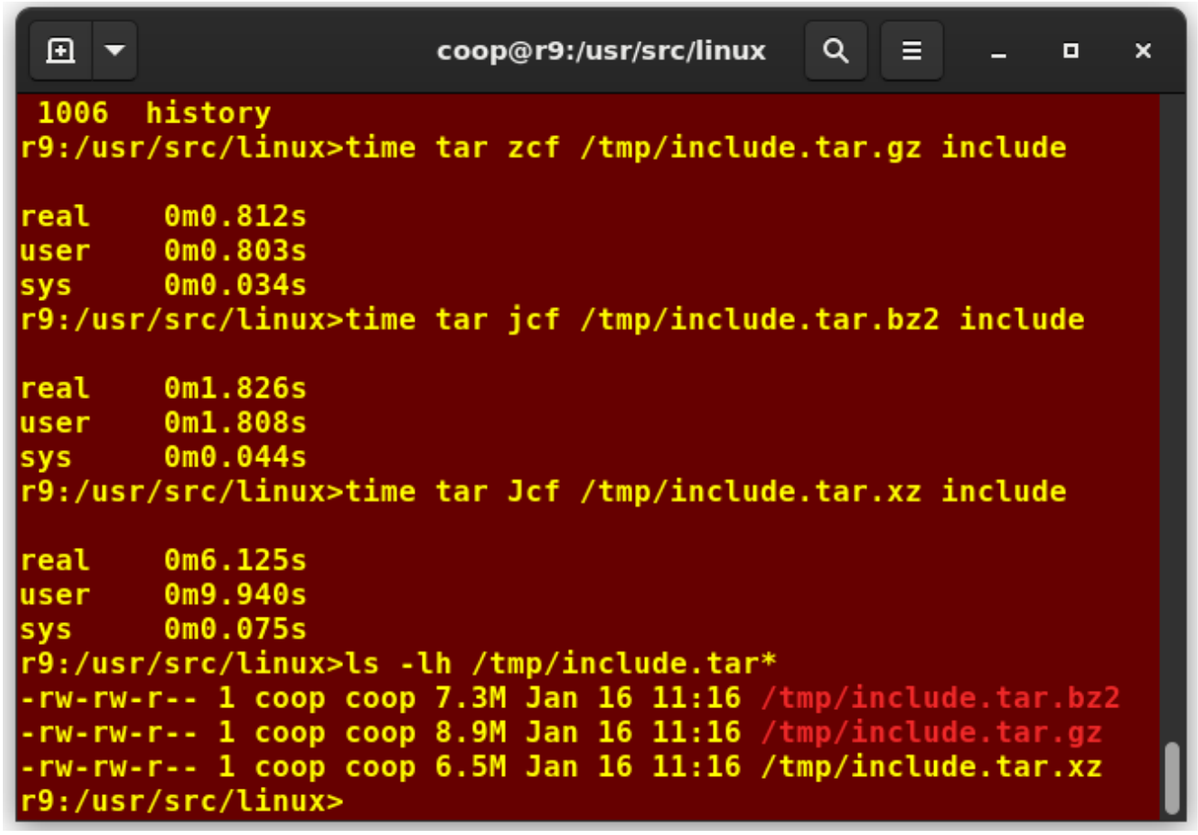
tar

Historically, **tar** stood for "**t**ape **ar**chive" and was used to archive files to a magnetic tape. It allows you to create or extract files from an archive file, often called a **tarball**. At the same time, you can optionally compress while creating the archive, and decompress while extracting its contents.

Table: tar Usage Examples

Command	Usage
`tar xvf mydir.tar`	Extract all the files in **`mydir.tar`** into the **`mydir`** directory.
`tar zcvf mydir.tar.gz mydir`	Create the archive and compress with **`gzip`**.
`tar jcvf mydir.tar.bz2 mydir`	Create the archive and compress with **`bz2`**.
`tar Jcvf mydir.tar.xz mydir`	Create the archive and compress with **`xz`**.
`tar xvf mydir.tar.gz`	Extract all the files in **`mydir.tar.gz`** into the **`mydir`** directory. ***NOTE**: You do **not** have to tell **tar** it is in **gzip** format.*

Relative Compression Times and Sizes:



---

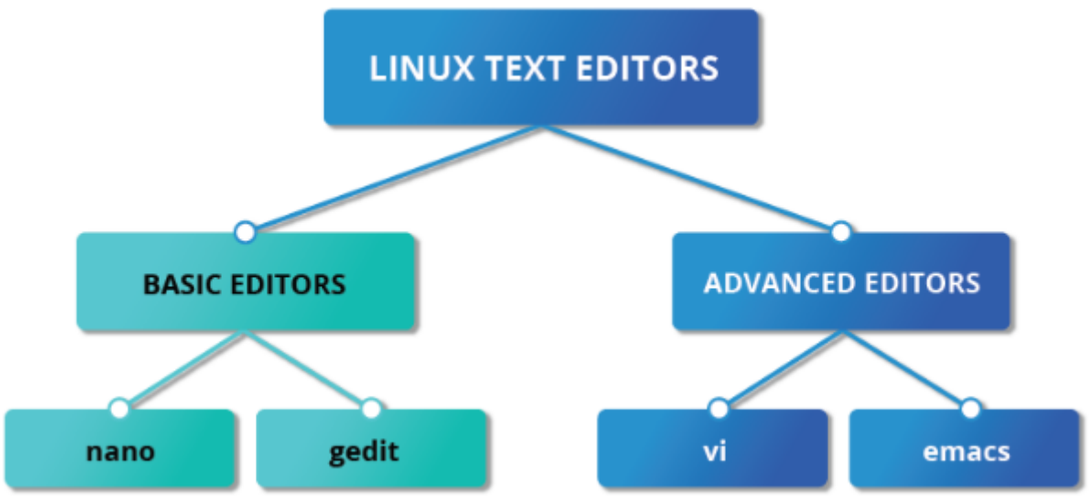
dd

The **dd** program is very useful for making copies of raw disk space.

```
dd if=/dev/sda of=/dev/sdb
```

To make a copy of one disk onto another, will delete everything that previously existed on the second disk. An exact copy of the first disk device is created on the second disk device.

Chap11 - Text Editors



nano

type **nano filename** and press **Enter** - file created, if doesn't exist.

- **CTRL-G**
  Display the help screen.
- **CTRL-O**
  Write to a file.
- **CTRL-X**
  Exit a file.
- **CTRL-R**
  Insert contents from another file to the current buffer.
- **CTRL-C**
  Show cursor position.

---

gedit

type **gedit filename** - file created if doesn't exist.
Simple-to-use graphical editor that can only be run within a Graphical Desktop environment.
It is visually quite similar to the Notepad text editor in Windows.

VS Code

type **code filename** - file created if doesn't exist.
Simple-to-use graphical editor.

---

vi

Table: Modes and Features

Mode	Feature
Command	• By default, **vi** starts in Command mode. • Each key is an editor command. • Keyboard strokes are interpreted as commands that can modify file contents.
Insert	• Type **i** to switch to Insert mode from Command mode. • Insert mode is used to enter (insert) text into a file. • Insert mode is indicated by an "**? INSERT ?**" indicator at the bottom of the screen. • Press **Esc** to exit Insert mode and return to Command mode.
Line	• Type **:** to switch to the Line mode from Command mode. Each key is an external command, including operations such as writing the file contents to disk or exiting. • Uses line editing commands inherited from older line editors. Most of these commands are actually no longer used. Some line editing commands are very powerful. • Press **Esc** to exit Line mode and return to Command mode.

Table: Start, Exit, Read and Write Files in vi

Command	Usage
`vi myfile`	Start the editor and edit `myfile`
`vi -r myfile`	Start and edit `myfile` in recovery mode from a system crash
`:r file2`	Read in `file2` and insert at current position
`:w`	Write to the file
`:w myfile`	Write out to `myfile`
`:w! file2`	Overwrite `file2`
`:x or :wq`	Exit and write out modified file
`:q`	Quit
`:q!`	Quit even though modifications have not been saved

Table: Keys and Usage Examples

Key	Usage
arrow keys	To move up, down, left and right
j or **<ret>**	To move one line down
k	To move one line up
h or Backspace	To move one character left
l or Space	To move one character right
0	To move to beginning of line
$	To move to end of line
w	To move to beginning of next word
:0 or **1G**	To move to beginning of file
:n or **nG**	To move to line n
:$ or **G**	To move to last line in file
CTRL-F or **Page Down**	To move forward one page
CTRL-B or **Page Up**	To move backward one page
^l	To refresh and center screen

Table: Commands Used to Search for Text in vi

Command	Usage
/pattern	Search forward for pattern
?pattern	Search backward for pattern

Table: Keystrokes Used to Search for Text in vi

Key	Usage
`n`	Move to next occurrence of search pattern
`N`	Move to previous occurrence of search pattern

Table: Changing, Adding, Deleting Text in vi

Key	Usage
`a`	Append text after cursor; stop upon **Escape** key
`A`	Append text at end of current line; stop upon **Escape** key
`i`	Insert text before cursor; stop upon **Escape** key
`I`	Insert text at beginning of current line; stop upon **Escape** key
`o`	Start a new line below current line, insert text there; stop upon **Escape** key
`O`	Start a new line above current line, insert text there; stop upon **Escape** key
`r`	Replace character at current position
`R`	Replace text starting with current position; stop upon **Escape** key
`x`	Delete character at current position
`Nx`	Delete N characters, starting at current position
`dw`	Delete the word at the current position
`D`	Delete the rest of the current line
`dd`	Delete the current line
`Ndd` or `dNd`	Delete N lines
`u`	Undo the previous operation
`yy`	Yank (copy) the current line and put it in buffer
`Nyy` or `yNy`	Yank (copy) N lines and put it in buffer
`p`	Paste at the current position the yanked line or lines from the buffer

---

emacs

Unlike vi, it does not work with modes. emacs is highly customizable and includes a large number of features. It was initially designed for use on a console, but was soon adapted to work with a GUI as well.

Rather than having different modes for command and insert, like vi, emacs uses the **CTRL** and Meta (**Alt** or **Esc**) keys for special commands.

Table: Starting, Exiting, Reading, and Writing Files in emacs

Key	Usage
`emacs myfile`	Start emacs and edit `myfile`
`CTRL-x i`	Insert prompted for file at current position
`CTRL-x s`	Save all files
`CTRL-x CTRL-w`	Write to the file giving a new name when prompted
`CTRL-x CTRL-s`	Saves the current file
`CTRL-x CTRL-c`	Exit after being prompted to save any modified files

Table: Keys and Usage Examples

Key	Usage
arrow keys	Use the arrow keys for up, down, left and right
`CTRL-n`	One line down
`CTRL-p`	One line up
`CTRL-f`	One character forward/right
`CTRL-b`	One character back/left
`CTRL-a`	Move to beginning of line
`CTRL-e`	Move to end of line
`Meta-f`	Move to beginning of next word
`Meta-b`	Move back to beginning of preceding word
`Meta-<`	Move to beginning of file
`Meta-g-g-n`	Move to line n (can also use '`Esc-x Goto-line n`')
`Meta->`	Move to end of file
`CTRL-v` or `Page Down`	Move forward one page
`Meta-v` or `Page Up`	Move backward one page
`CTRL-l`	Refresh and center screen

Table: Key Combinations Used to Search for Text in emacs

Key	Usage
`CTRL-s`	Search forward for prompted pattern, or for next pattern
`CTRL-r`	Search backwards for prompted pattern, or for next pattern

Table: Changing, Adding, and Deleting Text in emacs

Key	Usage
CTRL-o	Insert a blank line
CTRL-d	Delete character at current position
CTRL-k	Delete the rest of the current line
CTRL-_	Undo the previous operation
CTRL- (space or **CTRL-@**)	Mark the beginning of the selected region (the end will be at the cursor position)
CTRL-w	Delete the current marked text and write it to the buffer
CTRL-y	Insert at current cursor location whatever was most recently deleted

---

Chap12 - User Environment

Id current user - type **whoami**
Id all users currently logged-on - type **who**

Startup Files

Used to configure the user environment.
Files in the **/etc** directory define global settings for all users, while initialization files in the user's home directory can include and/or override the global settings.

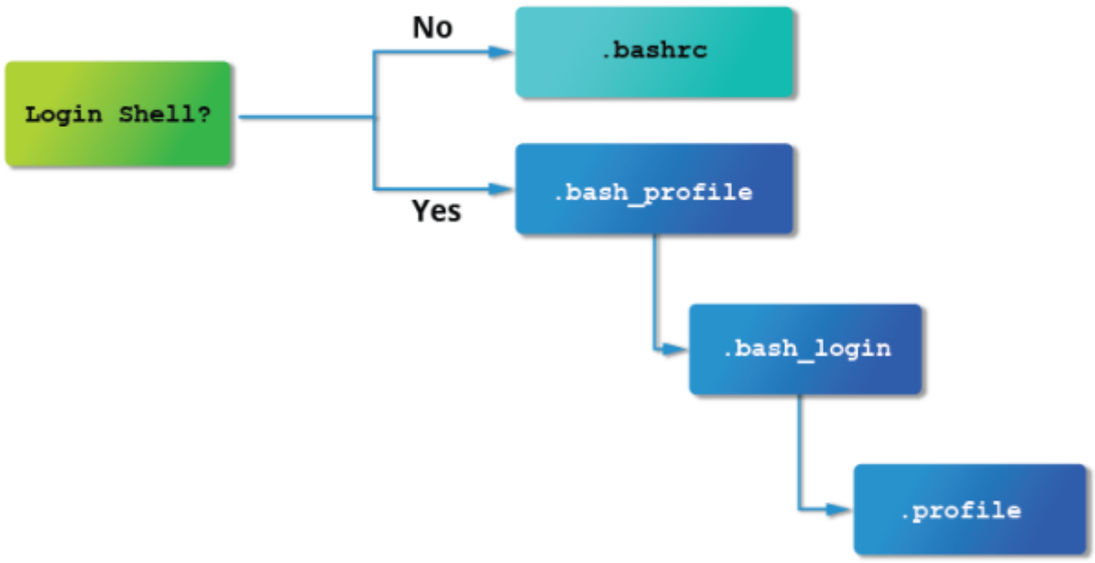
The startup files can do anything the user would like to do in every command shell, such as:

- Customizing the prompt
- Defining command line shortcuts and aliases
- Setting the default text editor
- Setting the path for where to find executable programs

The standard prescription is that when you first login to Linux, **/etc/profile** is read and evaluated, after which the following files are searched (if they exist) in the listed order:

1. `~/.bash_profile`
2. `~/.bash_login`
3. `~/.profile`

However, every time you create a new shell, or terminal window, etc., you do not perform a full system login; only a file named **~/.bashrc** file is read and evaluated.



Order of the Startup Files

Create customized commands or modify the behaviour of already existing ones by creating **aliases**. Most often, these aliases are placed in your **~/.bashrc** file so they are available to any command shells you create. **unalias** removes an alias.
typing **alias** without arguments will list currently defined aliases.

Users and Groups

All Linux users are assigned a unique user ID (**uid**), which is just an integer; normal users start with a uid of 1000 or greater.

Linux uses **groups** for organizing users. Groups are collections of accounts with certain shared permissions; they are used to establish a set of users who have common interests for the purposes of access rights, privileges, and security considerations.

Control of group membership is administered through the **/etc/group** file, which shows the list of groups and their members.

Users also have one or more group IDs (**gid**), including a default one that is the same as the user ID. These numbers are associated with names through the
files **/etc/passwd** and **/etc/group**.

Adding a new user is done with **useradd** and removing an existing user is done
with **userdel**.
Adding new group is done with **groupadd** and removing an existing group is done with **groupdel**.

Adding a user to an already existing group is done with **usermod**. These utilities update **/etc/group** as necessary. Make sure to use the **-a** option, for append, so as to avoid removing already existing groups.

groupmod can be used to change group properties, such as the Group ID (gid) with the **-g** option or its name with then **-n** option.

Typing **id** with no argument gives information about the current user.

---

su and sudo

> When assigning elevated privileges, you can use the command **su** (**s**witch or **s**ubstitute **u**ser) to launch a new shell running as another user. It is almost always a bad (dangerous for both security and stability) practice to use **su** to become root. Resulting errors can include deletion of vital files from the system and security breaches.
>
> Granting privileges using **sudo** is less dangerous and is preferred.
>
> To temporarily become the superuser for a series of commands, you can type **su** and then be prompted for the root password.
>
> To execute just one command with root privilege type **sudo command**. When the command is complete, you will return to being a normal unprivileged user.
>
> **sudo** configuration files are stored in the **/etc/sudoers** file and in the **/etc/sudoers.d/** directory. By default, the **sudoers.d** directory is empty.

Environment Variables

> **Environment variables** are quantities that have specific values which may be utilized by the command shell, such as **bash**, or other utilities and applications. Some environment variables are given preset values by the system (which can usually be overridden), while others are set directly by the user, either at the command line or within startup and other scripts.
>
> There are a number of ways to view the values of currently set environment variables; one can type **set**, **env**, or **export.**
>
> By default, variables created within a script are only available to the current shell; child processes (sub-shells) will not have access to values that have been set or modified. Allowing child processes to see the values requires use of the **export** command.

Table: Tasks and Commands

Task	Command
Show the value of a specific variable	`echo $SHELL`
Export a new variable value	`export VARIABLE=value` (or `VARIABLE=value; export VARIABLE`)
Add a variable permanently	Edit `~/.bashrc` and add the line `export VARIABLE=value` Type `source ~/.bashrc` or just `. ~/.bashrc` (*dot* `~/.bashrc`); or just start a new shell by typing `bash`

HOME variable

> **HOME** is an environment variable that represents the home (or login) directory of the user. **cd** without arguments will change the current working directory to the value of **HOME**.

> Note the tilde character (**~**) is often used as an abbreviation for **$HOME**.

Table: Commands and Explanations

Command	Explanation

`$ echo $HOME`
`/home/student`
`$ cd /bin`
	Show the value of the **HOME** environment variable, then change directory (**cd**) to **/bin**.

`$ pwd` `/bin`	Where are we? Use print (or present) working directory (**pwd**) to find out. As expected, **/bin**.
`$ cd`	Change directory without an argument...
`$ pwd` `/home/student`	...takes us back to **HOME**, as you can now see.

PATH variable

> **PATH** is an ordered list of directories (the path) which is scanned when a command is given to find the appropriate program or script to run. Each directory in the path is separated by colons (**:**). A null (empty) directory name (or ***./***) indicates the current directory at any given time.

SHELL variable

> The environment variable **SHELL** points to the user's default command shell and contains the full pathname to the shell.

PS1 variable

> Prompt Statement (**PS**) is used to customize your prompt string in your terminal windows to display the information you want.
>
> **PS1** is the primary prompt variable which controls what your command line prompt looks like. The following special characters can be included in **PS1**:
>
> **\u** - User name
> **\h** - Host name
> **\w** - Current working directory
> **\!** - History number of this command
> **\d** - Date

History

type **history** at command line to view list of previously executed commands. The list of commands is displayed with the most recent command appearing last in the list. This information is stored in **~/.bash_history**.

Several associated environment variables can be used to get information about the **history** file:

- **`HISTFILE`**
  The location of the history file.
- **`HISTFILESIZE`**
  The maximum number of lines in the history file (default 500).
- **`HISTSIZE`**
  The maximum number of commands in the history file.
- **`HISTCONTROL`**
  How commands are stored.
- **`HISTIGNORE`**
  Which command lines can be unsaved.

Table: Keys and Usage Examples

Key	Usage
`Up`/**`Down`** arrow keys	Browse through the list of commands previously executed
`!!` (Pronounced as bang-bang)	Execute the previous command
`CTRL-R`	Search previously used commands

CTRL-R - reverse intelligent search

Table: Syntax and Tasks

Syntax	Task
`!`	Start a history substitution
`!$`	Refer to the last argument in a line
`!n`	Refer to the n^{th} command line
`!string`	Refer to the most recent command starting with string

Keyboard Shortcuts

Table: Keyboard Shortcuts and Tasks

Keyboard Shortcut	Task
`CTRL-L`	Clears the screen
`CTRL-S`	Temporarily halt output to the terminal window
`CTRL-Q`	Resume output to the terminal window
`CTRL-D`	Exits the current shell
`CTRL-Z`	Puts the current process into suspended background
`CTRL-C`	Kills the current process
`CTRL-H`	Works the same as backspace
`CTRL-A`	Goes to the beginning of the line
`CTRL-W`	Deletes the word before the cursor
`CTRL-U`	Deletes from beginning of line to cursor position
`CTRL-E`	Goes to the end of the line
`Tab`	Auto-completes files, directories, and binaries

File Ownership

In Linux and other UNIX-based operating systems, every file is associated with a user who is the owner. Every file is also associated with a group (a subset of all users) which has an interest in the file and certain rights, or permissions: read, write, and execute.

Table: Commands and Usage

Command	Usage
`chown`	Used to change user ownership of a file or directory
`chgrp`	Used to change group ownership
`chmod`	Used to change the permissions on the file, which can be done separately for **owner**, **group** and the rest of the **world** (often named as **other**)

File Permission Modes

Files have three kinds of permissions: read (**r**), write (**w**), execute (**x**). These are generally represented as in **rwx**. These permissions affect three groups of owners: user/owner (**u**), group (**g**), and others (**o**).

- **4** if read permission is desired
- **2** if write permission is desired
- **1** if execute permission is desired

Thus, **7** means read/write/execute, **6** means read/write, and **5** means read/execute.

Chap13 - Manipulating Text

cat

Short for **concatenate**. It is often used to read and print files, as well as for simply viewing file contents. The main purpose of **cat** is often to combine (concatenate) multiple files together.

The **tac** command (**cat** spelled backwards) prints the lines of a file in reverse order. Each line remains the same, but the order of lines is inverted.

Table: Commands and Usage

Command	Usage
`cat file1 file2`	Concatenate multiple files and display the output; i.e. the entire content of the first file is followed by that of the second file
`cat file1 file2 > newfile`	Combine multiple files and save the output into a new file
`cat file >> existingfile`	Append a file to the end of an existing file
`cat > file`	Any subsequent lines typed will go into the file, until **CTRL-D** is typed
`cat >> file`	Any subsequent lines are appended to the file, until **CTRL-D** is typed

---

echo

echo simply displays (echoes) text. **echo** can be used to display a string on standard output (i.e. the terminal) or to place in a new file (using the **>** operator) or append to an already existing file (using the **>>** operator).

Table: Commands and Usage

Command	Usage
`echo string > newfile`	The specified string is placed in a new file
`echo string >> existingfile`	The specified string is appended to the end of an already existing file
`echo $variable`	The contents of the specified environment variable are displayed

---

less - to view the contents of such a large file, scrolling up and down page by page, without the system having to place the entire file in memory before starting.

Compressed Files

When working with compressed files, many standard commands cannot be used directly. For many commonly-used file and text manipulation programs, there is also a version especially designed to work directly with compressed files. These associated utilities often have the letter "z" prefixed to their name.

Table: z Family Commands and Descriptions

Command	Description
`$ zcat compressed-file.txt.gz`	To view a compressed file

`$ zless somefile.gz`
or
`$ zmore somefile.gz`
	To page through a compressed file

`$ zgrep -i less somefile.gz`	To search inside a compressed file
`$ zdiff file1.txt.gz file2.txt.gz`	To compare two compressed files

sed

sed is a powerful text processing tool. It is used to modify the contents of a file or input stream, usually placing the contents into a new file or output stream. Its name is an abbreviation for stream editor.

Data from an input source/file (or stream) is taken and moved to a working space. The entire list of operations/modifications is applied over the data in the working space and the final contents are moved to the standard output space (or stream).

Table: Commands and Usage Examples

Command	Usage
`sed -e command <filename>`	Specify editing commands at the command line, process input from a file, and put the output on standard out (e.g., the terminal)
`sed -f scriptfile <filename>`	Specify a script file containing **sed** commands, operate on file, and put output on standard out
`echo "I hate you" | sed s/hate/love/`	Use **sed** to filter standard input, putting output on standard out

Table: Commands and Usage Examples

Command	Usage
`sed s/pattern/replace_string/ file`	Substitute first string occurrence in every line
`sed s/pattern/replace_string/g file`	Substitute all string occurrences in every line
`sed 1,3s/pattern/replace_string/g file`	Substitute all string occurrences in a range of lines
`sed -i s/pattern/replace_string/g file`	Save changes for string substitution in the same file

---

awk

awk is used to extract and then print specific contents of a file and is often used to construct reports.

Table: Commands and Usage Examples

Command	Usage
`awk 'command' file`	Specify a command directly at the command line
`awk -f scriptfile file`	Specify a file that contains the script to be executed

The input file is read one line at a time, and, for each line, **awk** matches the given pattern in the given order and performs the requested action. The **-F** option allows you to specify a particular **field separator** character.

Table: Commands and Usage Examples

Command	Usage
`awk '{ print $0 }' /etc/passwd`	Print entire file
`awk -F: '{ print $1 }' /etc/passwd`	Print first field (column) of every line, separated by a colon
`awk -F: '{ print $1 $7 }' /etc/passwd`	Print first and seventh field of every line

---

sort

sort is used to rearrange the lines of a text file, in either ascending or descending order according to a sort key. When used with the **-u** option, **sort** checks for unique values after

sorting the records (lines).

Table: Using sort

Syntax	Usage
`sort <filename>`	Sort the lines in the specified file, according to the characters at the beginning of each line
`cat file1 file2 | sort`	Combine the two files, then sort the lines and display the output on the terminal
`sort -r <filename>`	Sort the lines in reverse order
`sort -k 3 <filename>`	Sort the lines by the 3rd field on each line instead of the beginning

---

uniq

uniq removes duplicate consecutive lines in a text file and is useful for simplifying the text display.
Because **uniq** requires that the duplicate entries must be consecutive, one often runs sort first and then pipes the output into **uniq**; if sort is used with the **-u** option, it can do all this in one step.

---

paste

paste can be used to create a single file containing all columns. The different columns are identified based on delimiters.

paste accepts the following options:

- **`-d`** delimiters, which specify a list of delimiters to be used instead of tabs for separating consecutive values on a single line. Each delimiter is used in turn; when the list has been exhausted, **`paste`** begins again at the first delimiter.
- **`-s`**, which causes paste to append the data in series rather than in parallel; that is, in a horizontal rather than vertical fashion.

paste can be used to combine fields (such as name or phone number) from different files, as well as combine lines from multiple files.

---

join

Essentially, an enhanced version of **paste**. It first checks whether the files share common fields, such as names or phone numbers, and then joins the lines in two files based on a common field.

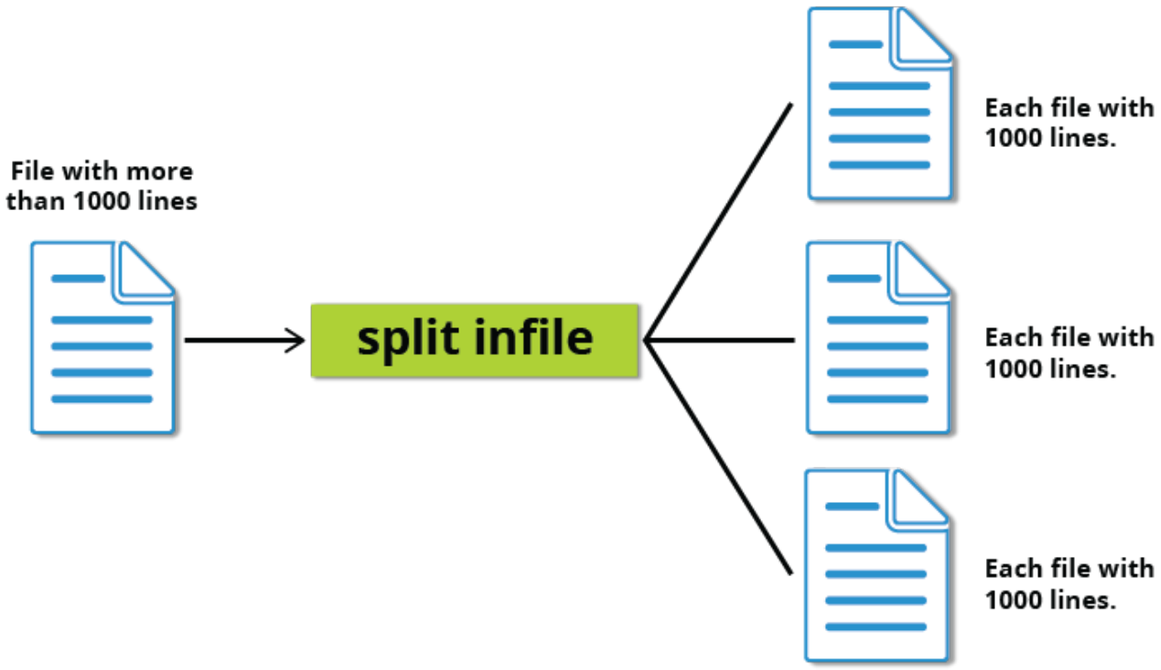
---

split

split is used to break up (or split) a file into equal-sized segments for easier viewing and manipulation, and is generally used only on relatively large files.By default, **split** breaks up a file into 1000-line segments.

The original file remains unchanged, and a set of new files with the same name plus an added prefix is created. By default, the **x** prefix is added.

To split a file into segments, use the command **split infile**.



RegEx and Search Patterns

Regular expressions are text strings used for matching a specific pattern, or to search for a specific location, such as the start or end of a line or a word.

Table: Search Patterns and Usage Examples

Search Patterns	Usage
`.(dot)`	Match any single character
`a|z`	Match a or z
`$`	Match end of a line
`^`	Match beginning of a line
`*`	Match preceding item 0 or more times

---

grep

grep is extensively used as a primary text searching tool. It scans files for specified patterns and can be used with regular expressions, as well as simple strings.

Table: Commands and Usage Examples

Command	Usage
`grep [pattern] <filename>`	Search for a pattern in a file and print all matching lines
`grep -v [pattern] <filename>`	Print all lines that do **not** match the pattern
`grep [0-9] <filename>`	Print the lines that contain the numbers **0** through **9**
`grep -C 3 [pattern] <filename>`	Print context of lines (specified number of lines above and below the pattern) for matching the pattern. Here, the number of lines is specified as 3

---

strings

> **strings** is used to extract all printable character strings found in the file or files given as arguments. It is useful in locating human-readable content embedded in binary files.

---

tr

> The **tr** utility is used to translate specified characters into other characters or to delete them.

Table: Command and Usage Example

Command	Usage
`tr abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ`	Convert lower case to upper case
`tr '{}' '()' < inputfile > outputfile`	Translate braces into parenthesis
`echo "This is for testing" | tr [:space:] '\t'`	Translate white-space to tabs
`echo "This is for testing" | tr -s [:space:]`	Squeeze repetition of characters using **-s**
`echo "the geek stuff" | tr -d 't'`	Delete specified characters using **-d** option
`echo "my username is 432234" | tr -cd [:digit:]`	Complement the sets using **-c** option
`tr -cd [:print:] < file.txt`	Remove all non-printable character from a file
`tr -s '\n' ' ' < file.txt`	Join all the lines in a file into a single line

---

tee

> **tee** takes the output from any command, and, while sending it to standard output, it also saves it to a file.

---

wc

> **wc** (**w**ord **c**ount) counts the number of lines, words, and characters in a file or list of files.
>
> **Table: Options and Descriptions**
>
Option	Description
`-l`	Displays the number of lines
`-c`	Displays the number of bytes
`-w`	Displays the number of words

---

cut

> **cut** is used for manipulating column-based files and is designed to extract specific columns. The default column separator is the **TAB** character.

Chap14 - Network Operations

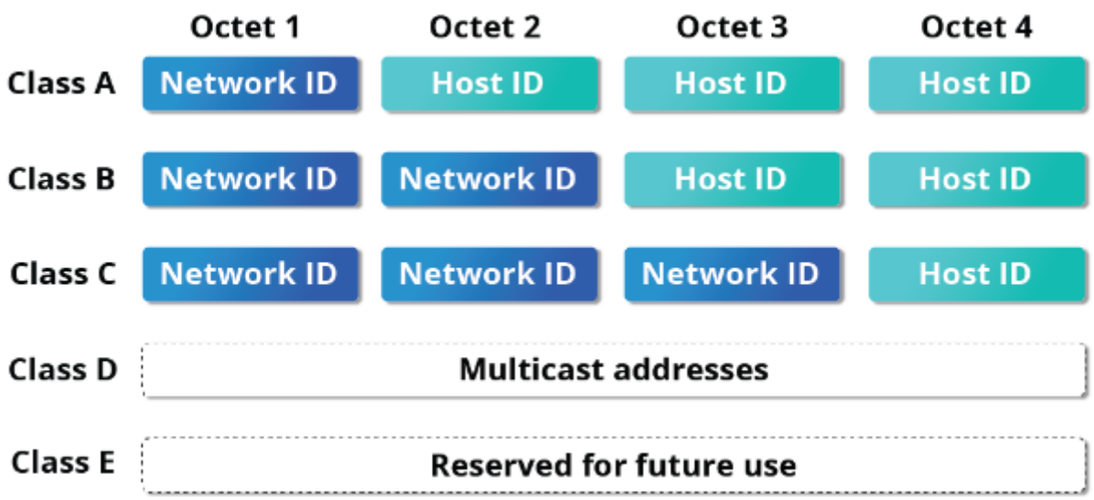
Networks

> A network is a group of computers and computing devices connected together through communication channels, such as cables or wireless media. The connected devices are often termed **nodes**.
>
> The **Internet** is the largest network in the world and can be called **the network of networks**.

IP Addresses

> Devices attached to a network must have at least one unique network address identifier known as the **IP** (Internet Protocol) address. This address is essential for routing packets of information through the network.
>
> Exchanging information across the network requires using streams of small packets, each of which contains a piece of the information going from one machine to another. These packets contain data buffers, together with headers which contain information about where the packet is going to and coming from and where it fits in the sequence of packets that constitute the stream.
>
> IPv4 uses 32-bits for addresses; there are only 4.3 billion unique addresses available. Furthermore, many addresses are allotted and reserved, but not actually used. IPv4 is considered inadequate for meeting future needs because the number of devices available on the global network has increased enormously in recent years.
>
> IPv6 uses 128-bits for addresses; this allows for 3.4 X 1038 unique addresses. If you have a larger network of computers and want to add more, you might want to move to IPv6 because it provides more unique addresses.

> One reason IPv4 has not disappeared is there are widely-used ways to effectively make many more addresses available by methods such as NAT (Network Address Translation). NAT enables sharing one IP address among many locally connected computers, each of which has a unique address only seen on the local network. While this is used in organizational settings, it is also used in simple home networks.
>
> 32-bit IPv4 address- divided into four 8-bit sections called **octets**. Network addresses are divided into five classes: A, B, C, D and E. Classes A, B and C are classified into two parts: Network addresses (Net ID) and Host address (Host ID). The Net ID is used to identify the network, while the Host ID is used to identify a host in the network.
>
> Class D - Special multicast applications, Class E - Reserved for future use.



Class A

> Class A addresses use the first octet of an IP address as their Net ID and use the other three octets as the Host ID**.** The first bit of the first octet is always set to zero. So you can use only 7-bits for unique network numbers. As a result, there are a maximum of 126 Class A networks available (the addresses 0000000 and 1111111 are reserved).

Class B

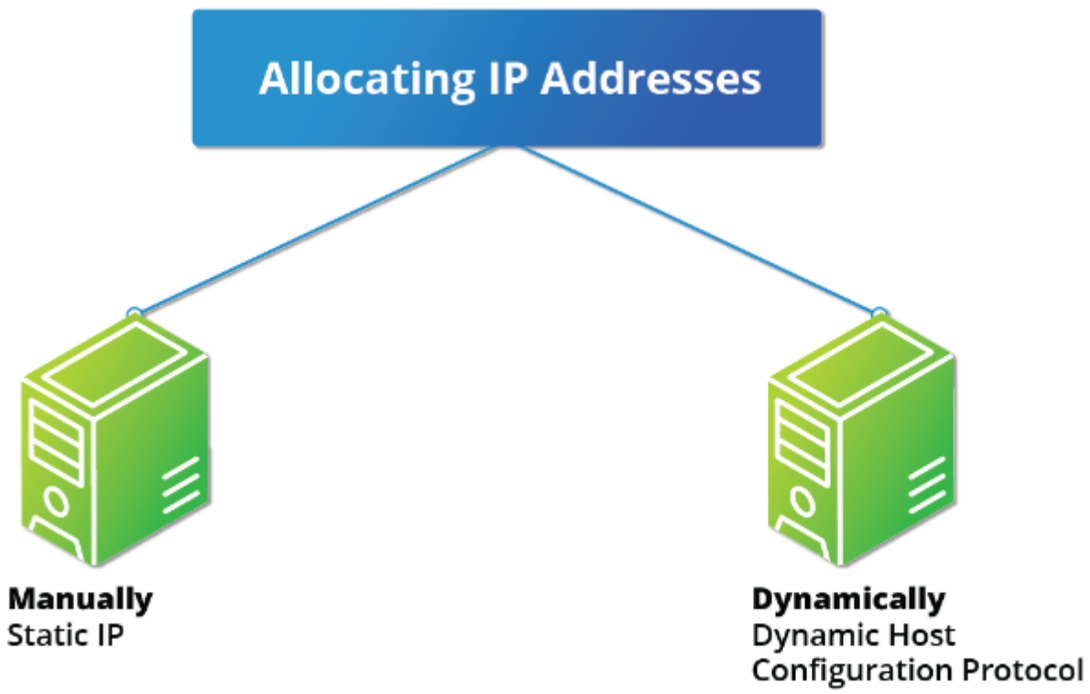
> Class B addresses use the first two octets of the IP address as their Net ID and the last two octets as the Host ID. The first two bits of the first octet are always set to binary 10, so there are a maximum of 16,384 (14-bits) Class B networks. The first octet of a Class B address has values from 128 to 191.

Class C

> Class C addresses use the first three octets of the IP address as their Net ID and the last octet as their Host ID. The first three bits of the first octet are set to binary 110, so almost 2.1 million (21-bits) Class C networks are available. The first octet of a Class C address has values from 192 to 223.

IP Address Allocation

> Typically, a range of IP addresses are requested from your Internet Service Provider (ISP) by your organization's network administrator. Often, your choice of which class of IP address you are given depends on the size of your network and expected growth needs.
>
> You can assign IP addresses to computers over a network either manually or dynamically. Manual assignment adds static (never changing) addresses to the network. Dynamically assigned addresses can change every time you reboot or even more often; the Dynamic Host Configuration Protocol (DHCP) is used to assign IP addresses.



Name Resolution

> Name Resolution is used to convert numerical IP address values into a human-readable format known as the hostname.
>
> The special hostname **localhost** is associated with the IP address **127.0.0.1** and describes the machine you are currently on.

Network Configuration Files

> Network configuration files are essential to ensure that interfaces function correctly. They are located in the ***/*etc** directory tree.

Network Interfaces

> Network interfaces are a connection channel between a device and a network. Physically, network interfaces can proceed through a network interface card (NIC) or can be more abstractly implemented as software.

Information about a particular network interface or all network interfaces can be reported by the **ip** and **ifconfig** utilities.

To view the IP address:

```
$ /sbin/ip addr show
```

To view the routing information:

```
$ /sbin/ip route show
```

---

ping

ping is used to check whether or not a machine attached to the network can receive and send data; i.e. it confirms that the remote host is online and is responding.

ping is frequently used for network testing and management; however, its usage can increase network load unacceptably. Hence, you can abort the execution of **ping** by typing **CTRL-C**, or by using the **-c** option, which limits the number of packets that **ping** will send before it quits. When execution stops, a summary is displayed.

---

route

A network requires the connection of many nodes. Data moves from source to destination by passing through a series of routers and potentially across multiple networks. Servers maintain routing tables containing the addresses of each node in the network. The IP routing protocols enable routers to build up a forwarding table that correlates final destinations with the next hop addresses.

Use **route** utility - view or change the IP routing table to add, delete, or modify specific (static) routes to specific hosts or networks.

Table: Tasks and Commands

Task	Command
Show current routing table	`$ route -n` or `ip route`
Add static route	`$ route add -net address` or `ip route add`
Delete static route	`$ route del -net address` or `ip route del`

---

traceroute

traceroute is used to inspect the route which the data packet takes to reach the destination host, which makes it quite useful for troubleshooting network delays and errors. By using **traceroute**,
you can isolate connectivity issues between hops, which helps resolve them faster.

Table: Networking Tools and Their Descriptions

Networking Tools	Description
`ethtool`	Queries network interfaces and can also set various parameters such as the speed
`netstat`	Displays all active connections and routing tables; useful for monitoring performance and troubleshooting
`nmap`	Scans open ports on a network; important for security analysis
`tcpdump`	Dumps network traffic for analysis
`iptraf`	Monitors network traffic in text mode
`mtr`	Combines functionality of ping and traceroute and gives a continuously updated display
`dig`	Tests DNS workings; a good replacement for host and nslookup

Browsers

Browsers are used to retrieve, transmit, and explore information resources, usually on the World Wide Web.

The common graphical browsers used in Linux are: Firefox, Google Chrome, Chromium, Konqueror, Opera

Table: Non-graphical Browsers and Their Descriptions

Non-Graphical Browsers	Description
lynx	Configurable text-based web browser; the earliest such browser and still in use
elinks	Based on Lynx; it can display tables and frames
w3m	Another text-based web browser with many features

wget

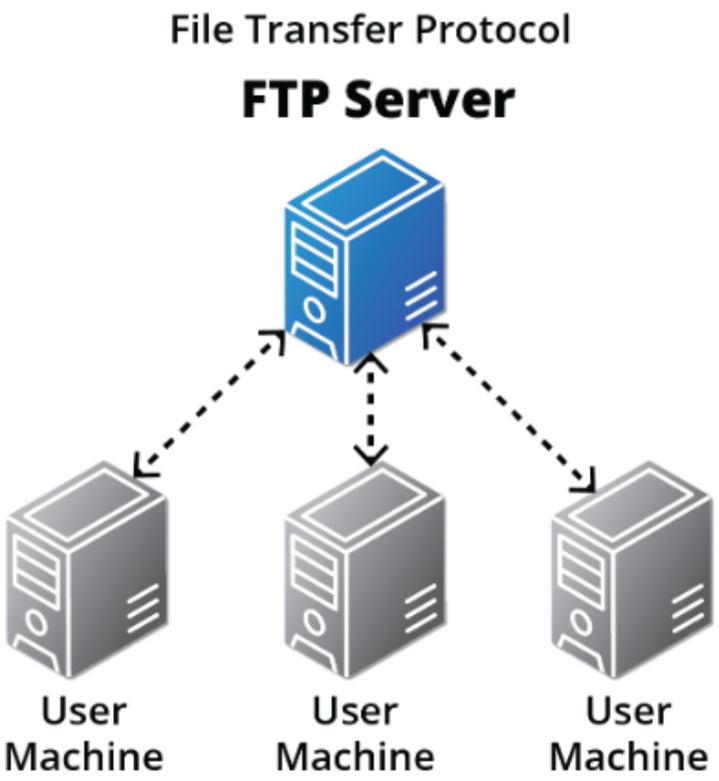
wget is a command line utility that can capably handle the following types of downloads:

- Large file downloads
- Recursive downloads, where a web page refers to other web pages and all are downloaded at once
- Password-required downloads
- Multiple file downloads.

curl

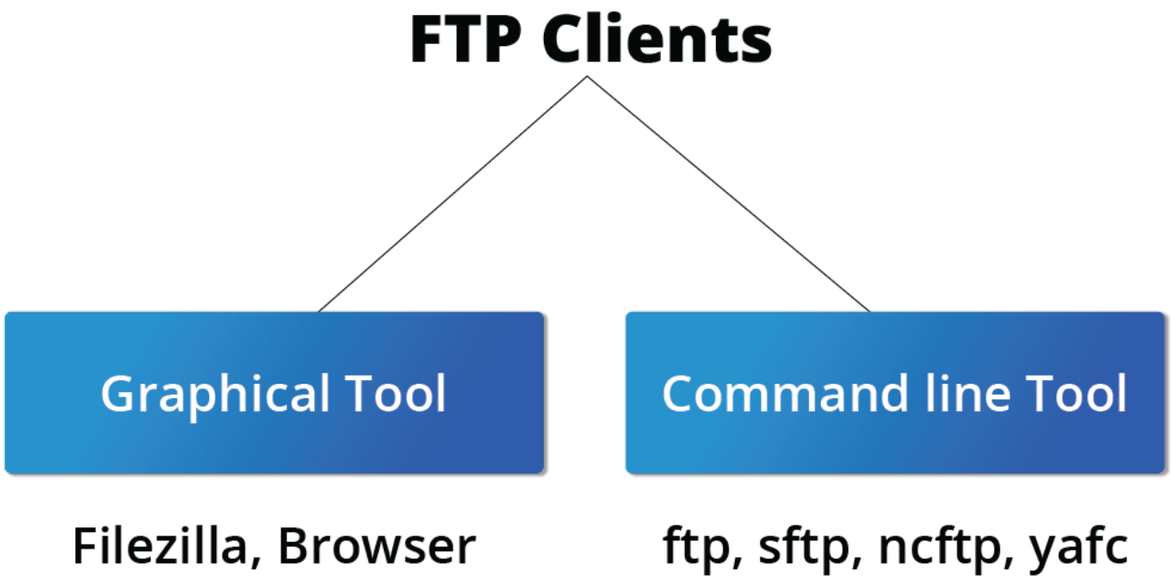
> Besides downloading, you may want to obtain information about a URL, such as the source code being used. **curl** can be used from the command line or a script to read such information. **curl** also allows you to save the contents of a web page to a file.
>
> To read URL : **curl URL**
> To get contents of webpage and store in a file: **curl -o filename URL**

FTP

> When you are connected to a network, you may need to transfer files from one machine to another. File Transfer Protocol (FTP) is a well-known and popular method for transferring files between computers using the Internet. This method is built on a client-server model.
>
> 
>

FTP Clients

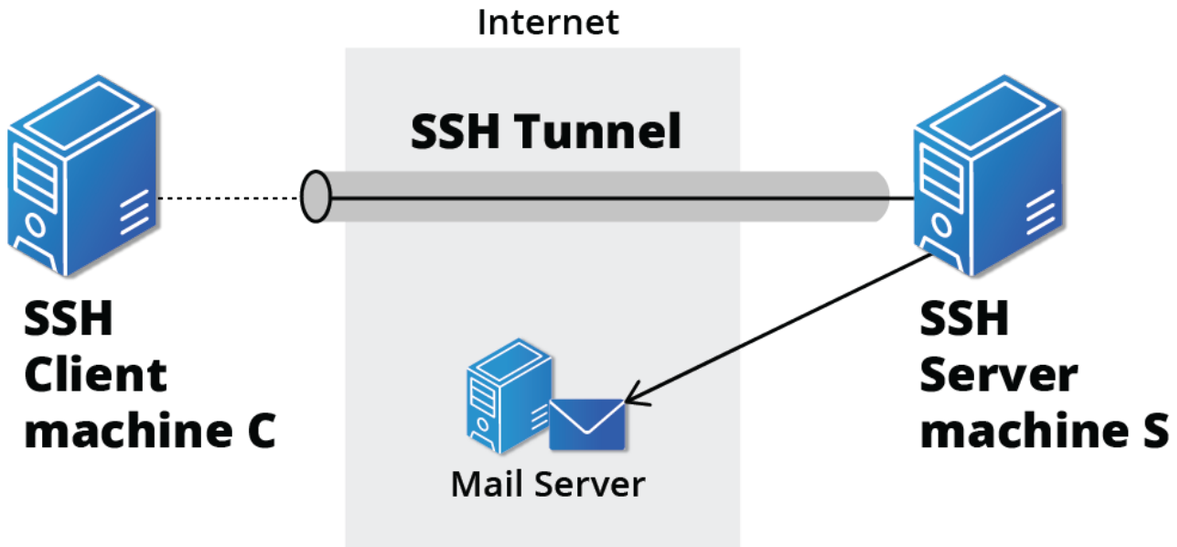
> FTP clients enable you to transfer files with remote computers using the FTP protocol. These clients can be either graphical or command line tools.



The reason FTP has fallen into disfavor on modern systems is that it is intrinsically insecure; passwords are user credentials that can be transmitted without encryption and are thus prone to interception. Thus, it was removed in favor of using rsync and web browser https access for example. As an alternative, **sftp** is a very secure mode of connection, which uses the Secure Shell (**ssh**) protocol. **sftp** encrypts its data and thus sensitive information is transmitted more securely.

SSH

Secure Shell (SSH) is a cryptographic network protocol used for secure data communication. It is also used for remote services and other secure services between devices on the network and is very useful for administering systems which are not easily available to physically work on, but to which you have remote access.



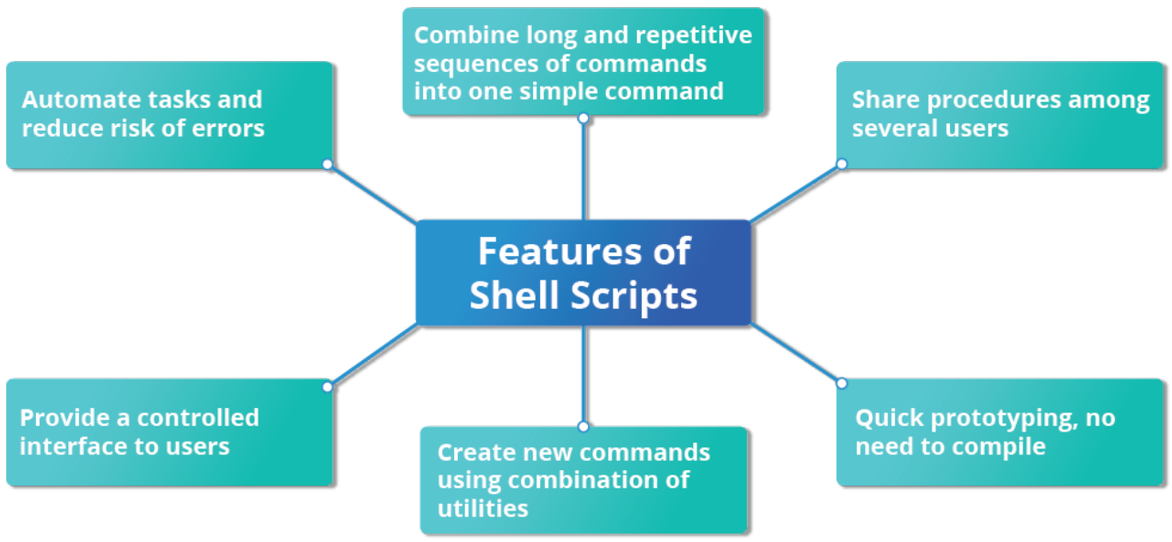
To login to a remote system using your same user name you can just type **ssh some_system** and press **Enter**.
If you want to run as another user, you can do either **ssh -l someone some_system** or **ssh someone@some_system**.
To run a command on a remote system via SSH, at the command prompt, you can type **ssh some_system my_command**.

---

scp

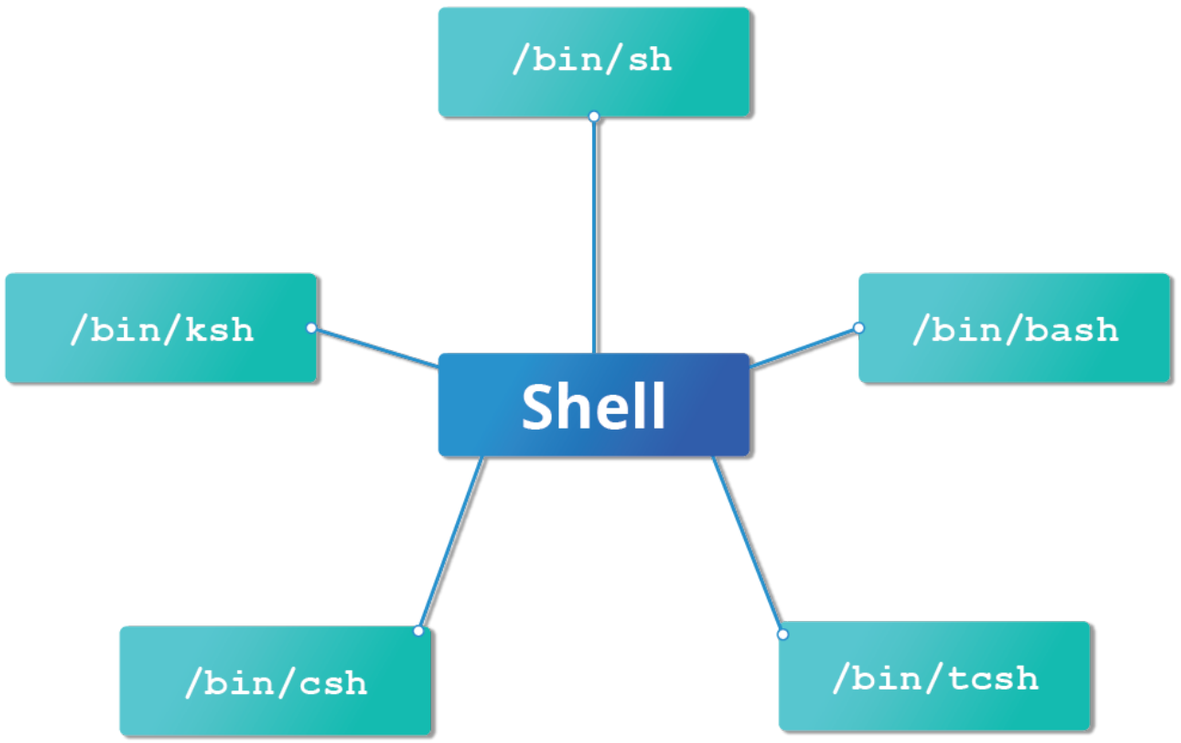
> We can also move files securely using Secure Copy (scp) between two networked hosts. scp uses the SSH protocol for transferring data.
> To copy a local file to a remote system, at the command prompt, type **scp localfile user@remotesystem:/home/user/** and press **Enter**.

Chap15 - Bash Shell and Scripting



Command Shells

> Linux provides a wide choice of shells; exactly what is available on the system is listed in **/etc/shells**.



Return Values

> All shell scripts generate a return value upon finishing execution, which can be explicitly set with the **exit** statement. Return values permit a process to monitor the exit state of another process, often in a parent-child relationship.

> By convention, success is returned as zero (**0**), and failure is returned as any non-zero value.
>
> For non-existing file - returns 2, for existing file - returns 0.

Basic Syntax

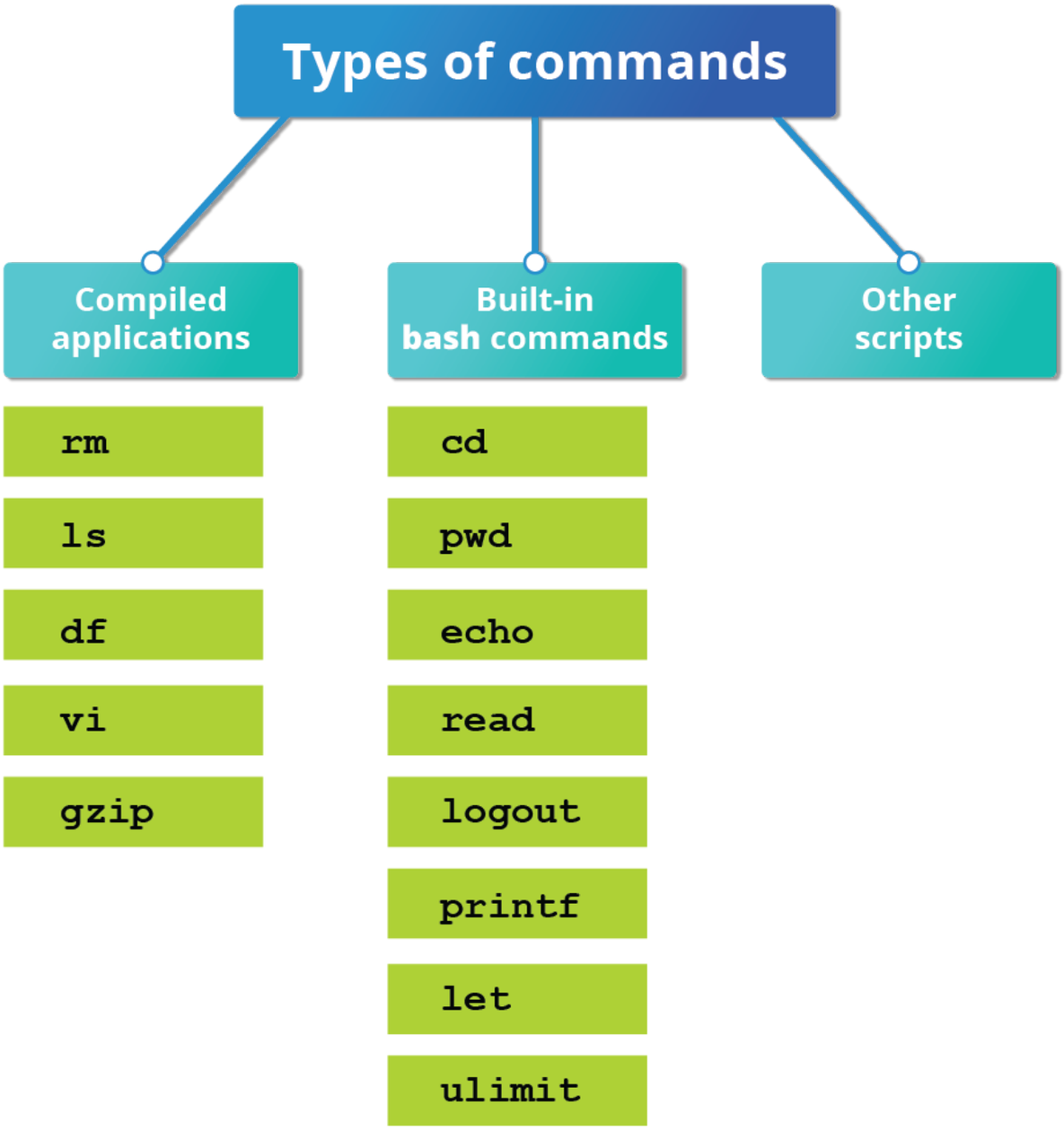
> **Table: Character Usages and Their Descriptions**
>
Character	Description
#	Used to add a comment, except when used as \#, or as #! when starting a script
\	Used at the end of a line to indicate continuation on to the next line, or to indicate that the next character is to be interpreted literally, as in \$
;	Used to interpret what follows as a new command to be executed after completion of the current command
$	Indicates what follows is an environment variable
>	Redirect output
>>	Append output
<	Redirect input
|	Used to pipe the result into the next command

> backslash - concatenation operator.
> semicolon ; - put multiple commands on the same line - chaining of commands.
> and && - if first command fails, all other commands aborted.
> or || - continue till one of the commands succeeds.

Built-in Shell Commands

> Shell scripts execute sequences of commands and other types of statements. These commands can be:
>
> - Compiled applications
> - Built-in bash commands
> - Shell scripts or scripts from other interpreted languages, such as perl and Python.
>
> Compiled applications are binary executable files, usually residing on the filesystem in well-known directories such as **/usr/bin**. Shell scripts always have access to applications in the default path, such as **rm**, **ls**, **df**, **vi**, and **gzip**, which are programs compiled from lower-level programming languages such as **C**.

In addition, bash has many **built-in** commands, which can only be used to display the output within a terminal shell or shell script.These built-in commands include **cd**, **pwd**, **echo**, **read**, **logout**, **printf**, **let**, **time**, and **ulimit**. All commands listed using **help**.



Script Parameters

Table: Parameters and Their Meanings

Parameter	Meaning
$0	Script name
$1	First parameter
$2, **$3**, etc.	Second, third parameter, etc.
$*	All parameters
$#	Number of arguments

Command substitution

> At times, you may need to substitute the result of a command as a portion of another command. It can be done in two ways:
>
> - By enclosing the inner command in **$( )**
> - By enclosing the inner command with backticks **(`)**

---

if Statement

```
if condition
then
        statements
else
        statements
fi
```

> When an **if** statement is used, the ensuing actions depend on the evaluation of specified conditions, such as:
>
> - Numerical or string comparisons
> - Return value of a command (0 for success)
> - File existence or permissions

---

elif Statement

```
if [ sometest ] ; then
    echo Passed test1
elif [ someothertest ] ; then
    echo Passed test2
fi
```

Testing for Files

Table: Conditions and Their Meanings

Condition	Meaning
`-e file`	Checks if the file exists
`-d file`	Checks if the file is a directory
`-f file`	Checks if the file is a regular file (i.e., not a symbolic link, device node, directory, etc.)
`-s file`	Checks if the file is of non-zero size
`-g file`	Checks if the file has **`sgid`** set
`-u file`	Checks if the file has **`suid`** set
`-r file`	Checks if the file is readable
`-w file`	Checks if the file is writable
`-x file`	Checks if the file is executable

Numerical Tests

Table: Operator and Meaning

Operator	Meaning
`-eq`	Equal to
`-ne`	Not equal to
`-gt`	Greater than
`-lt`	Less than
`-ge`	Greater than or equal to
`-le`	Less than or equal to

-op - used to compare numbers

Arithmetic Expressions

> Arithmetic expressions can be evaluated in the following three ways (spaces are important!
>
> - Using the **expr** utility
>   **expr** is a standard but somewhat deprecated program. The syntax is as follows:
>   `expr 8 + 8`
>   `echo $(expr 8 + 8)`
>
> - Using the **$((...))** syntax
>   This is the built-in shell format. The syntax is as follows:
>   `echo $((x+1))`
>
> - Using the built-in shell command **let**. The syntax is as follows:
>   `let x=( 1 + 2 ); echo $x`

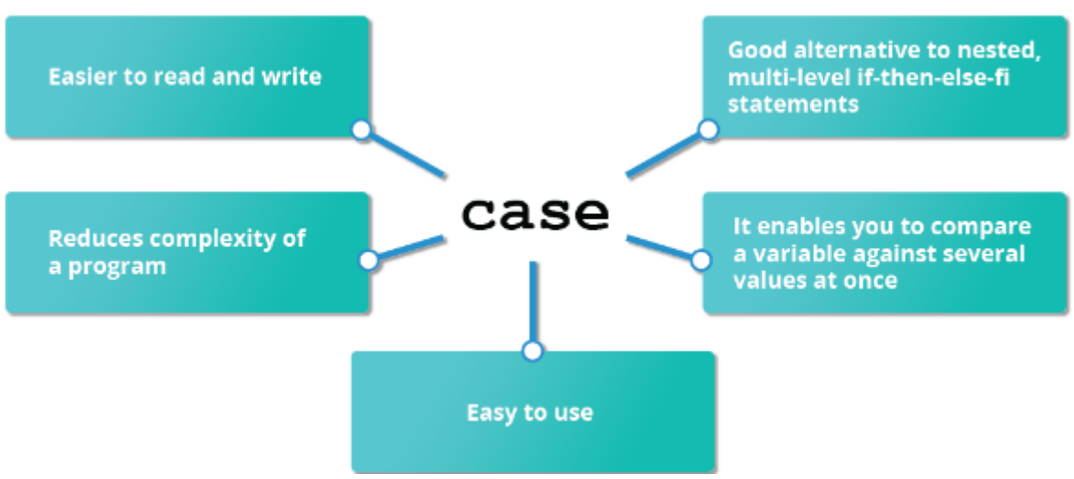
Chap16 - More on Bash Scripting

String Manipulation

> A string variable contains a sequence of text characters. It can include letters, numbers, symbols and punctuation marks.
>
> **Table: Operator and Its Meaning**
>
Operator	Meaning
`[[ string1 > string2 ]]`	Compares the sorting order of **string1** and **string2**
`[[ string1 == string2 ]]`	Compares the characters in **string1** with the characters in **string2**
`myLen1=${#string1}`	Saves the length of **string1** in the variable **myLen1**
>
> To extract the first **n** characters of a string, we can specify: **${string:0:n}**. Here, **0** is the offset in the string (i.e., which character to begin from) where the extraction needs to start, and **n** is the number of characters to be extracted.

---

case Statement

> The **case** statement is used in scenarios where the actual value of a variable can lead to different execution paths. **case** statements are often used to handle command-line options.
>
> Below are some of the advantages of using the **case** statement:
>
> 
>

Strcuture:

```
case expression in
   pattern1) execute commands;;
   pattern2) execute commands;;
   pattern3) execute commands;;
   pattern4) execute commands;;
   * )       execute some default commands or nothing ;;
esac
```

---

for Loop

Operates on each element in a list of items.

```
for variable-name in list
do
    execute one iteration for each item in the list until the list is finishe
done
```

---

while Loop

The **while** loop repeats a set of statements as long as the control command returns true.

```
while condition is true
do
    Commands for execution
    ----
done
```

---

until Loop

The **until** loop repeats a set of statements as long as the control command is false.

```
until condition is false
do
    Commands for execution
    ----
done
```

Script Debug Mode

You can run a bash script in debug mode either by doing **bash –x ./script_file**, or bracketing parts of the script with **set -x** and **set +x**.

Debug mode helps identify errors because:

- It traces and prefixes each command with the **+** character.
- It displays each command before executing it.
- It can debug only selected parts of a script (if desired) with:

```
set -x    # turns on debugging
...
set +x    # turns off debugging
```

Creating Temporary Files and Directories

Temporary files (and directories) are meant to store data for a short time. Usually, one arranges it so that these files disappear when the program using them terminates.

The best practice is to create random and unpredictable filenames for temporary storage. One way to do this is with the **mktemp** utility. The **XXXXXXXX** is replaced by **mktemp** with random characters to ensure the name of the temporary file cannot be easily predicted and is only known within your program.

Table: Commands and Usage Examples

Command	Usage
`TEMP=$(mktemp /tmp/tempfile.XXXXXXXX)`	To create a temporary file
`TEMPDIR=$(mktemp -d /tmp/tempdir.XXXXXXXX)`	To create a temporary directory

/dev/null

Certain commands (such as **find**) are quite capable of spewing voluminous and overwhelming amounts of output. To avoid this, we can redirect the large output to a special file (a device node) called **/dev/null**. This pseudofile is also called the bit bucket or black hole.

All data written to **/dev/null** is discarded. Furthermore, write operations never return failure conditions. Both **stdout** and **stderr** will be dumped into **/dev/null**.

Random Numbers and Data

Random numbers can be generated by using the **$RANDOM** environment variable, which is derived from the Linux kernel's built-in random number generator, or by the OpenSSL library function, which uses the FIPS140 (Federal Information Processing Standard) algorithm to generate random numbers for encryption.

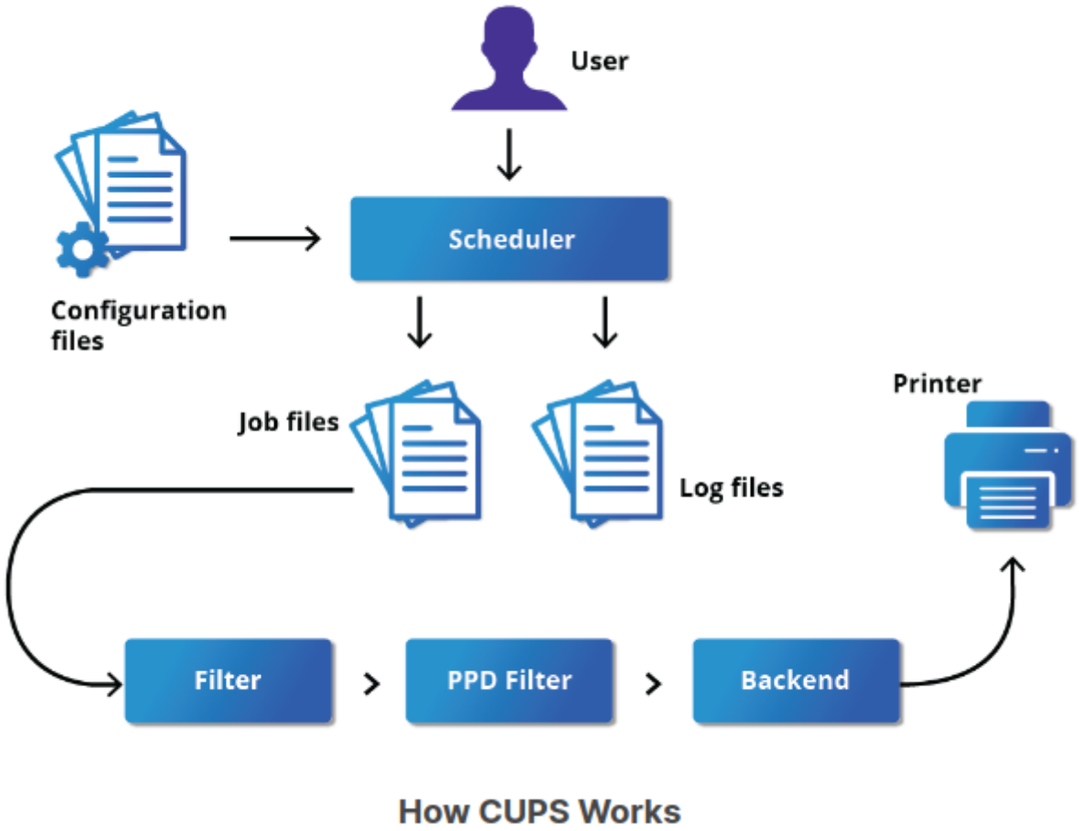
System maintains a so-called **entropy pool** of these digital numbers/random bits. Random numbers are created from this entropy pool.

> The Linux kernel offers the **/dev/random** and **/dev/urandom** device nodes, which draw on the entropy pool to provide random numbers which are drawn from the estimated number of bits of noise in the entropy pool.
> **/dev/random** is used where very high-quality randomness is required, such as a one-time pad or key generation, but it is relatively slow to provide values. **/dev/urandom** is faster and suitable (good enough) for most cryptographic purposes.
> Furthermore, when the entropy pool is empty, **/dev/random** is blocked and does not generate any number until additional environmental noise (network traffic, mouse movement, etc.) is gathered, whereas **/dev/urandom** reuses the internal pool to produce more pseudo-random bits.

Chap17 - Printing

CUPS

> Linux standard for printing - Common UNIX Printing System (CUPS).
>
> It interprets page descriptions produced by your application and then sends the information to the printer. It acts as a print server for both local and network printers.
>
> CUPS uses a modular printing system that accommodates a wide variety of printers and also processes various data formats. This makes the printing process simpler; you can concentrate more on printing and less on how to print.
>
> CUPS carries out the printing process with the help of its various components:
>
> - Configuration files
> - Scheduler
> - Job files
> - Log files
> - Filter
> - Printer drivers
> - Backend.



How CUPS Works

Scheduler

CUPS is designed around a print scheduler that manages print jobs, handles administrative commands, allows users to query the printer status, and manages the flow of data through all CUPS components.

Configuration Files

The print scheduler reads server settings from several configuration files, the two most important of which are **cupsd.conf** and **printers.conf**. These and all other CUPS-related configuration files are stored under the **/etc/cups/** directory.

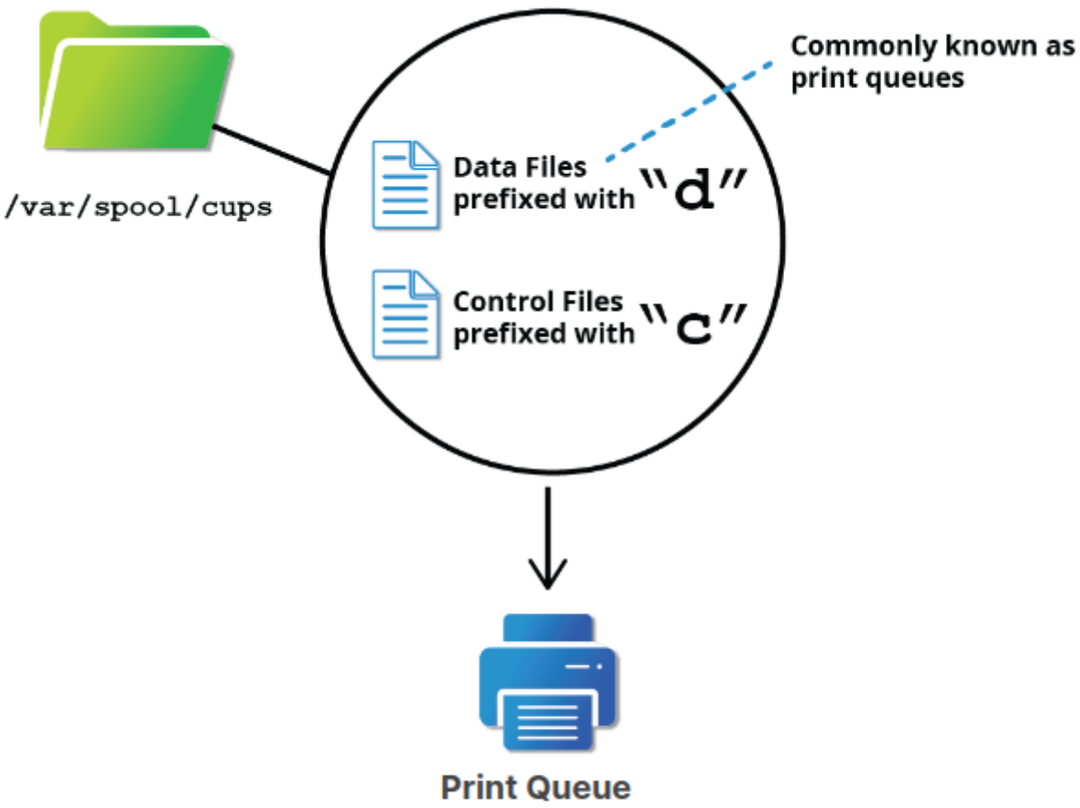
cupsd.conf is where most system-wide settings are located; it does not contain any printer-specific details. Most of the settings available in this file relate to network security, i.e., which systems can access CUPS network capabilities, how printers are advertised on the local network, what management features are offered, and so on.

printers.conf is where you will find the printer-specific settings. For every printer connected to the system, a corresponding section describes the printer's status and capabilities. This file is generated or modified only after adding a printer to the system and should not be modified by hand.

Job Files

CUPS stores print requests as files under the **/var/spool/cups** directory. Data files are prefixed with the letter **d**, while control files are prefixed with the letter **c**.

After a printer successfully handles a job, data files are automatically removed. These data files belong to what is commonly known as the **print queue**.



Log Files

Log files are placed in **/var/log/cups** and are used by the scheduler to record activities that have taken place. These files include access, error, and page records.

Filters, Printer Drivers and Backends

CUPS uses **filters** to convert job file formats to printable formats.
Printer drivers contain descriptions for currently connected and configured printers, and are usually stored under **/etc/cups/ppd/**.
The print data is then sent to the printer through a filter, and via a **backend** that helps to locate devices connected to the system.

Managing CUPS

```
$ systemctl status cups

$ sudo systemctl [enable|disable] cups

$ sudo systemctl [start|stop|restart] cups
```

> Many graphical applications allow users to access printing features using the **CTRL-P** shortcut.
>
> Use either **lp** (System V) or **lpr** (BSD) to print from Command-Line Interface.

---

lp

> **Table: Commands and Usage Examples**
>
Command	Usage
`lp <filename>`	To print the file to default printer
`lp -d printer <filename>`	To print to a specific printer (useful if multiple printers are available)
`program | lp` `echo string | lp`	To print the output of a program
`lp -n number <filename>`	To print multiple copies
`lpoptions -d printer`	To set the default printer
`lpq -a`	To show the queue status
`lpadmin`	To configure printer queues

Managing Print Jobs

> In Linux, command line print job management commands allow you to monitor the job state as well as managing the listing of all printers and checking their status, and canceling or moving print jobs to another printer.
>
> **Table: Commands and Usage Examples**
>
Command	Usage
`lpstat -p -d`	To get a list of available printers, along with their status
`lpstat -a`	To check the status of all connected printers, including job numbers

`cancel job-id`
or
`lprm job-id`
	To cancel a print job

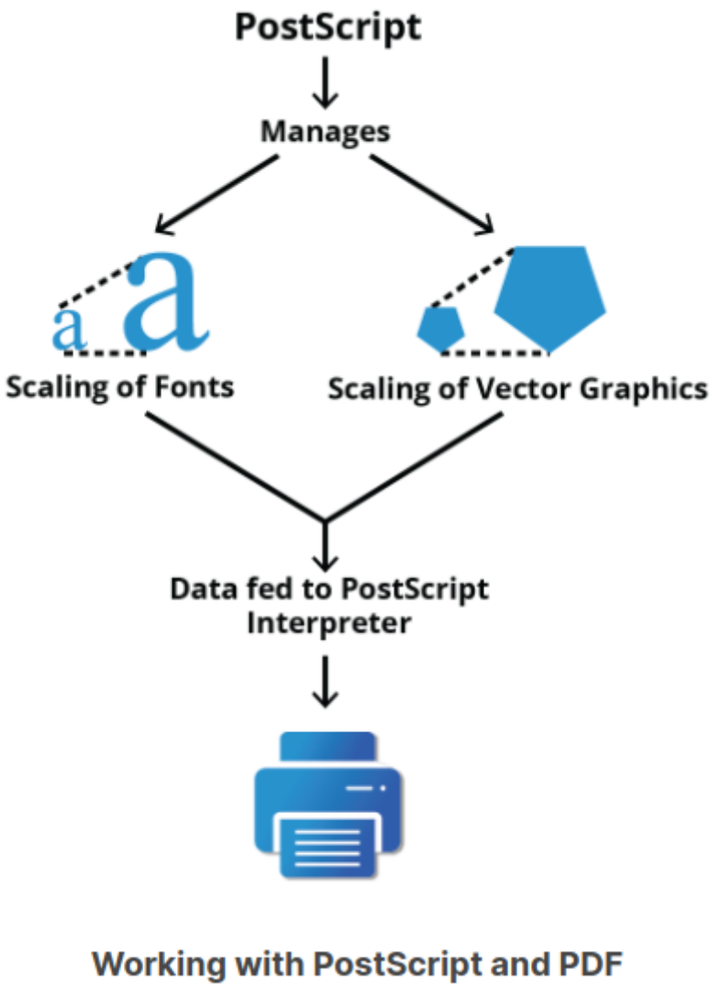
`lpmove job-id newprinter`	To move a print job to new printer

PostScript

> PostScript is a standard page description language. It effectively manages scaling of fonts and vector graphics to provide quality printouts. It is purely a text format that contains the data fed to a PostScript interpreter.

Features:

- It can be used on any printer that is PostScript-compatible, i.e., any modern printer.
- Any program that understands the PostScript specification can print to it.
- Information about page appearance, etc., is embedded in the page.



Working with PostScript and PDF

---

enscript

enscript is a tool that is used to convert a text file to PostScript and other formats. It also supports Rich Text Format (RTF) and HyperText Markup Language (HTML).

Table: Commands and Usage Examples

Command	Usage
`enscript -p psfile.ps textfile.txt`	Convert a text file to PostScript (saved to **psfile.ps**)
`enscript -n -p psfile.ps textfile.txt`	Convert a text file to n columns where n=1-9 (saved in **psfile.ps**)
`enscript textfile.txt`	Print a text file directly to the default printer

Converting between PostScript and PDF

Table: Commands and Usage Examples

Command	Usage
`pdf2ps file.pdf`	Converts **file.pdf** to **file.ps**
`ps2pdf file.ps`	Converts **file.ps** to **file.pdf**
`pstopdf input.ps output.pdf`	Converts **input.ps** to **output.pdf**
`pdftops input.pdf output.ps`	Converts **input.pdf** to **output.ps**
`convert input.ps output.pdf`	Converts **input.ps** to **output.pdf**
`convert input.pdf output.ps`	Converts **input.pdf** to **output.ps**

---

qpdf, pdftk, ghostscript - programs to manipulate PDFs.

---

qpdf

Table: Commands and Usage Examples

Command	Usage
`qpdf --empty --pages 1.pdf 2.pdf -- 12.pdf`	Merge the two documents **1.pdf** and **2.pdf**. The output will be saved to **12.pdf**.
`qpdf --empty --pages 1.pdf 1-2 -- new.pdf`	Write only pages 1 and 2 of **1.pdf**. The output will be saved to **new.pdf**.

`qpdf --rotate=+90:1 1.pdf 1r.pdf`
 	
`qpdf --rotate=+90:1-z 1.pdf 1r-all.pdf`
	Rotate page 1 of **1.pdf** 90 degrees clockwise and save to **1r.pdf**.
	
	Rotate all pages of **1.pdf** 90 degrees clockwise and save to **1r-all.pdf**

`qpdf --encrypt mypw mypw 128 -- public.pdf private.pdf`	Encrypt with 128 bits **public.pdf** using as the passwd **mypw** with output as **private.pdf**.
`qpdf --decrypt --password=mypw private.pdf file-decrypted.pdf`	Decrypt **private.pdf** with output as **file-decrypted.pdf**.

---

pdftk

If you’re working with PDF files that contain confidential information and you want to ensure that only certain people can view the PDF file, you can apply a password to it using the **user_pw** option.

```
$ pdftk public.pdf output private.pdf user_pw PROMPT
```

When you run this command, you will receive a prompt to set the required password, which can have a maximum of 32 characters. A new file, **private.pdf**, will be created with the identical content as **public.pdf**, but anyone will need to type the password to be able to

view it.

Table: Commands and Usage Examples

Command	Usage
`pdftk 1.pdf 2.pdf cat output 12.pdf`	Merge the two documents **`1.pdf`** and **`2.pdf`**. The output will be saved to **`12.pdf`**.
`pdftk A=1.pdf cat A1-2 output new.pdf`	Write only pages 1 and 2 of **`1.pdf`**. The output will be saved to **`new.pdf`**.
`pdftk A=1.pdf cat A1-endright output new.pdf`	Rotate all pages of **`1.pdf`** 90 degrees clockwise and save result in **`new.pdf`**.

---

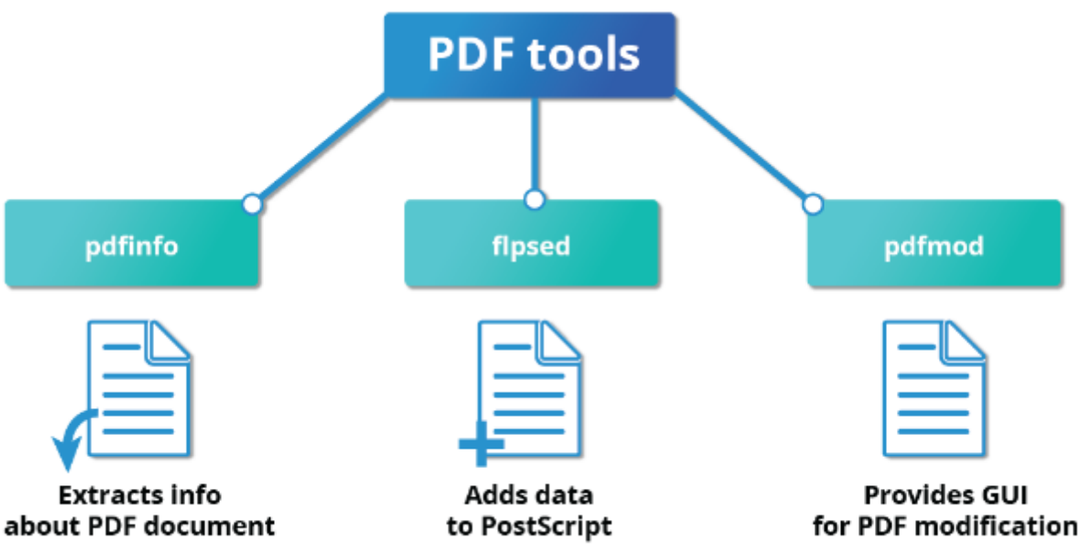
ghostscript

Ghostscript is widely available as an interpreter for the Postscript and PDF languages. The executable program associated with it is abbreviated to gs.

- Combine three PDF files into one:
  ```
  $ gs -dBATCH -dNOPAUSE -q -sDEVICE=pdfwrite  -sOutputFile=all.pdf file1.pdf file2.pdf file3.pdf
  ```

- Split pages 10 to 20 out of a PDF file:
  ```
  $ gs -sDEVICE=pdfwrite -dNOPAUSE -dBATCH -dDOPDFMARKS=false -dFirstPage=10 -dLastPage=20\
  -sOutputFile=split.pdf file.pdf
  ```

Additional Tools



Chap18 - Local Security Principles

User Accounts

The Linux kernel allows properly authenticated users to access files and applications. While each user is identified by a unique integer (the user id or UID), a separate database associates a username with each UID. Upon account creation, new user information

is added to the user database and the user's home directory must be created and populated with some essential files.

Command line programs such as **useradd** and **userdel** as well as GUI tools are used for creating and removing accounts.

Table: Fields Maintained in the `/etc/passwd` File

Field Name	Details	Remarks
Username	User login name	Should be between 1 and 32 characters long
Password	User password (or the character **x** if the password is stored in the `/etc/shadow` file) in encrypted format	Is never shown in Linux when it is being typed; this stops prying eyes
User ID (UID)	Every user must have a user id (UID)	• UID 0 is reserved for root user • UID's ranging from 1-99 are reserved for other predefined accounts • UID's ranging from 100-999 are reserved for system accounts and groups • Normal users have UID's of 1000 or greater
Group ID (GID)	The primary Group ID (GID); Group Identification Number stored in the `/etc/group` file	Is covered in detail in the chapter on *Processes*
User Info	This field is optional and allows insertion of extra information about the user such as their name	For example: **`Rufus T. Firefly`**
Home Directory	The absolute path location of user's home directory	For example: `/home/rtfirefly`
Shell	The absolute location of a user's default shell	For example: `/bin/bash`

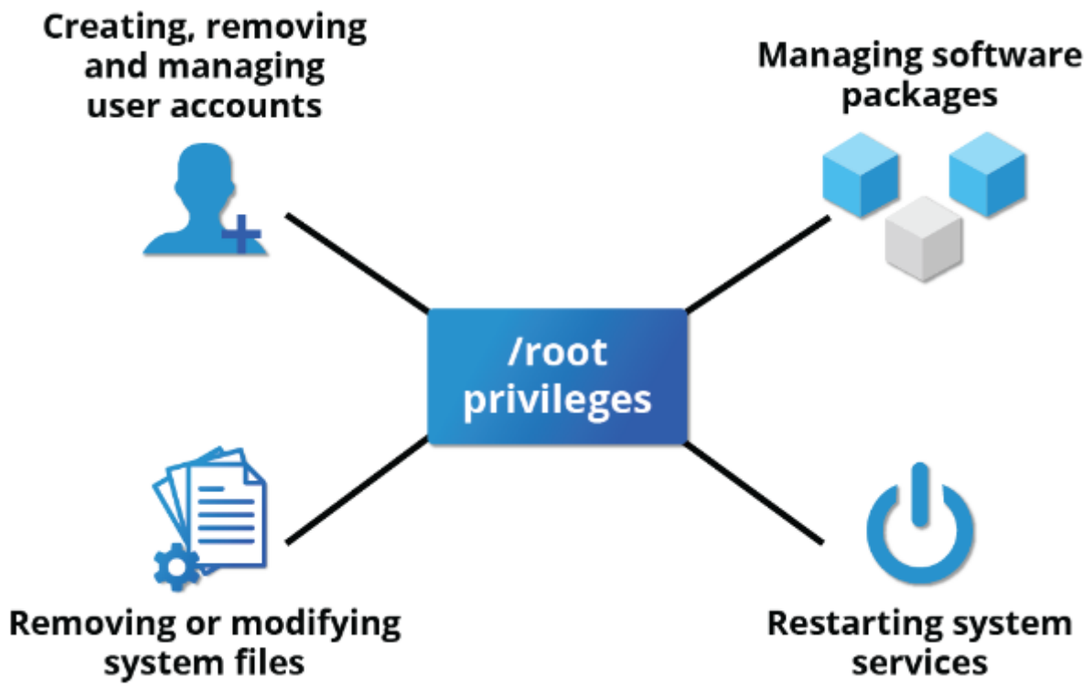
Linux - four type of accounts - root, System, Normal, Network.

---

root Account

root is the most privileged account on a Linux/UNIX system. This account has the ability to carry out all facets of system administration, including adding accounts, changing user passwords, examining log files, installing software, etc.

When you are signed in as, or acting as root, the shell prompt displays '#'.



Operations Not Requiring root Privileges

SUID (**S**et owner **U**ser **ID** upon execution - similar to the Windows "run as" feature) is a special kind of file permission given to a file. Use of SUID provides temporary permissions to a user to run a program with the permissions of the file **owner** (which may be root) instead of the permissions held by the **user**.

Table: Operations Which Do Not Require root Privileges

Operations that do not require root privilege	Examples of this operation
Running a network client	Sharing a file over the network
Using devices such as printers	Printing over the network
Operations on files that the user has proper permissions to access	Accessing files that you have access to or sharing data over the network
Running SUID-root applications	Executing programs such as **passwd**

sudo and su

Table: su and sudo Comparison

su	sudo
When elevating privilege, you need to enter the root password. Giving the root password to a normal user should never, ever be done.	When elevating privilege, you need to enter the user's password and not the root password.
Once a user elevates to the root account using **su**, the user can do anything that the root user can do for as long as the user wants, without being asked again for a password.	Offers more features and is considered more secure and more configurable. Exactly what the user is allowed to do can be precisely controlled and limited. By default the user will either always have to keep giving their password to do further operations with **sudo**, or can avoid doing so for a configurable time interval.
The command has limited logging features.	The command has detailed logging features.

sudo Features

> **sudo** has the ability to keep track of unsuccessful attempts at gaining root access. Users' authorization for using **sudo** is based on configuration information stored in the **/etc/sudoers** file and in the **/etc/sudoers.d** directory.

---

sudoers File

> Whenever **sudo** is invoked, a trigger will look at **/etc/sudoers** and the files in **/etc/sudoers.d** to determine if the user has the right to use **sudo** and what the scope of their privilege is. Unknown user requests and requests to do operations not allowed to the user even with **sudo** are reported.
> Basic structure:
>
> ```
> who where = (as_whom) what
> ```
>
> **/etc/sudoers** contains a lot of documentation in it about how to customize. Most Linux distributions now prefer you add a file in the directory **/etc/sudoers.d** with a name the same as the user. This file contains the individual user's **sudo** configuration, and one should leave the main configuration file untouched except for changes that affect all users.
>
> You should edit any of these configuration files by using **visudo**, which ensures that only one person is editing the file at a time, has the proper permissions, and refuses to write out the file and exit if there are syntax errors in the changes made.

Command logging

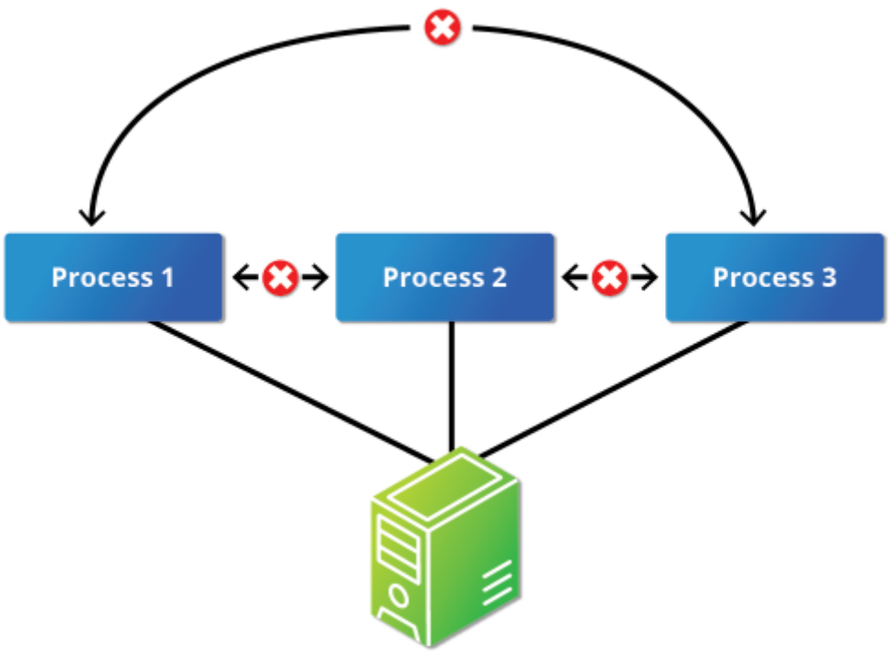
> By default, **sudo** commands and any failures are logged in **/var/log/auth.log** under the Debian distribution family, and in **/var/log/messages** and/or **/var/log/secure** on other systems.
> This is an important safeguard to allow for tracking and accountability of **sudo** use.
> A typical entry of the message contains:
>
> - Calling username
> - Terminal info
> - Working directory
> - User account invoked
> - Command with arguments

Process Isolation

> Linux is considered to be more secure than many other operating systems because processes are naturally isolated from each other. One process normally cannot access the resources of another process, even when that process is running with the same user privileges. Linux thus makes it difficult, for viruses and security exploits to access and attack random resources on a system.

More recent additional security mechanisms that limit risks even further include:

- Control Groups (cgroups)
  Allows system administrators to group processes and associate finite resources to each cgroup.
- Containers
  Makes it possible to run multiple isolated Linux systems (containers) on a single system by relying on cgroups.
- Virtualization
  Hardware is emulated in such a way that not only can processes be isolated, but entire systems are run simultaneously as isolated and insulated guests (virtual machines) on one physical host.



Hardware Device Access

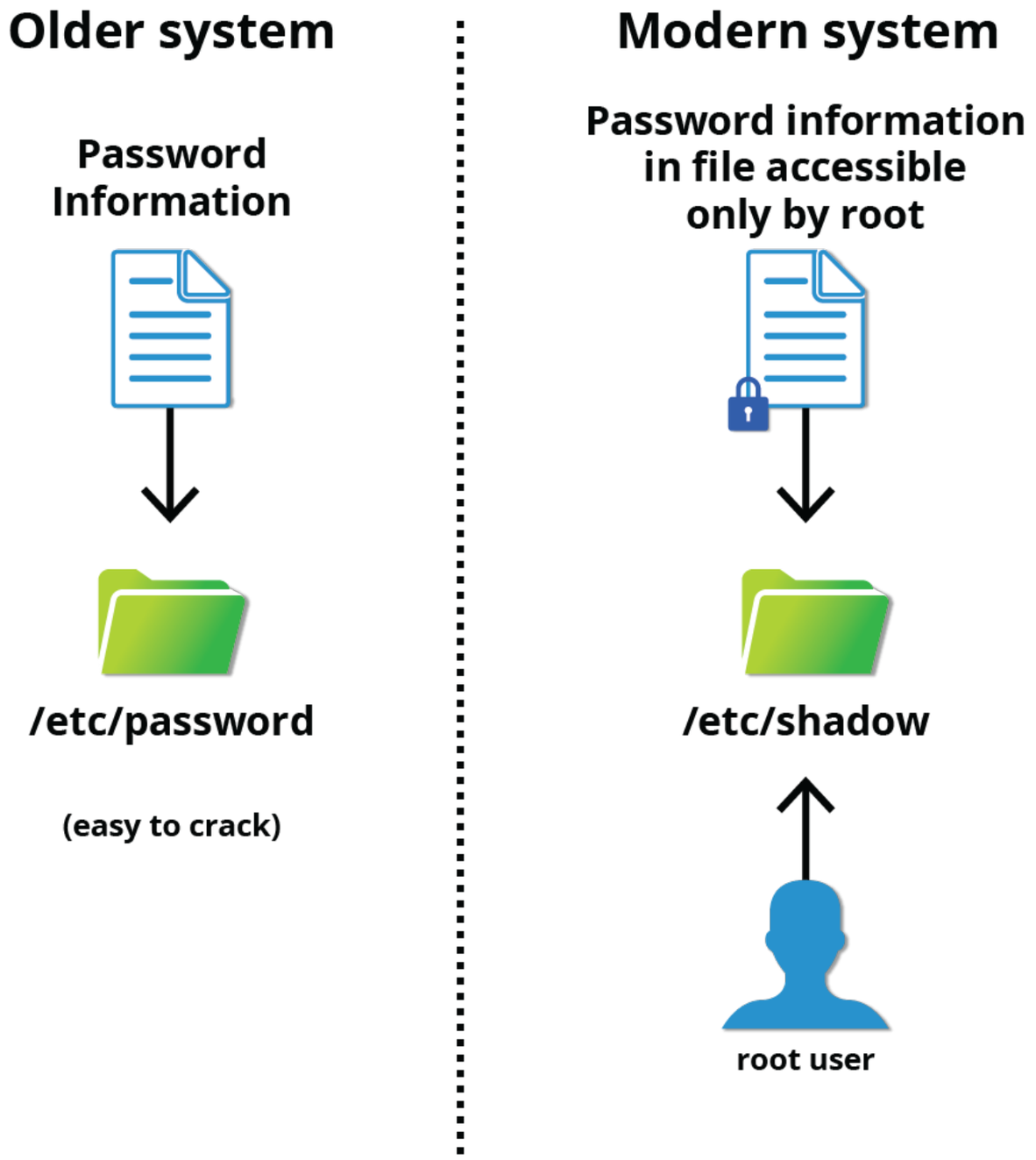
Linux limits user access to non-networking hardware devices in a manner that is extremely similar to regular file access. Applications interact by engaging the filesystem layer.
This layer will then open a device special file (often called a device node) under the **/dev** directory that corresponds to the device being accessed. Each device special file has standard owner, group and world permission fields. Security is naturally enforced just as it is when standard files are accessed.
The normal reading and writing of files on the hard disk by applications is done at a higher level through the filesystem and never through direct access to the device node.

Keeping Current

When security problems in either the Linux kernel or applications and libraries are discovered, Linux distributions have a good record of reacting quickly and pushing out fixes to all systems by updating their software repositories and sending notifications to update immediately. The same thing is true with bug fixes and performance improvements that are not security related.

Passwords

The system verifies authenticity and identity using user credentials.



Password Encryption

Protecting passwords has become a crucial element of security. Most Linux distributions rely on a modern password encryption algorithm called SHA-512 (Secure Hashing Algorithm 512 bits).
The SHA-512 algorithm is widely used for security applications and protocols. These security applications and protocols include TLS, SSL, PHP, SSH, S/MIME and IPSec.

Good Password Practices

- Password aging is a method to ensure that users get prompts that remind them to create a new password after a specific period. This can ensure that passwords, if cracked, will only be usable for a limited amount of time. This feature is implemented using **chage**, which configures the password expiry information for a user.
- Another method is to force users to set strong passwords using **P**luggable **A**uthentication **M**odules (**PAM**). PAM can be configured to automatically verify that a password created or modified using the **passwd** utility is sufficiently strong. PAM configuration is implemented using a library called **`pam_cracklib.so`**, which can also be replaced by **`pam_passwdqc.so`** to take advantage of more options.
- One can also install password cracking programs, such as John The Ripper, to secure the password file and detect weak password entries. It is recommended that written authorization be obtained before installing such tools on any system that you do not own.

Boot Loader Passwords

You can secure the boot process with a secure password to prevent someone from bypassing the user authentication step. This can work in conjunction with password protection for the BIOS.

For the older GRUB 1 boot method, it was relatively easy to set a password for **grub**. However, for the GRUB 2 version, things became more complicated. However, you have more flexibility, and can take advantage of more advanced features, such as user-specific passwords.
Furthermore, you never edit **grub.cfg** directly; instead, you can modify the configuration files in **/etc/grub.d** and **/etc/defaults/grub**, and then run **update-grub**, or **grub2-mkconfig** and save the new configuration file.

Hardware Vulnerability

When hardware is physically accessible, security can be compromised by:

- Key logging
  Recording the real-time activity of a computer user, including the keys they press. The captured data can either be stored locally or transmitted to remote machines.
- Network sniffing
  Capturing and viewing the network packet level data on your network.
- Booting with a live or rescue disk
- Remounting and modifying disk content.

The guidelines of security are:

- Lock down workstations and servers.
- Protect your network links such that it cannot be accessed by people you do not trust.
- Protect your keyboards where passwords are entered to ensure the keyboards cannot be tampered with.
- Ensure a password protects the BIOS in such a way that the system cannot be booted with a live or rescue DVD or USB key.
