

LFS101X

LinuxFoundationX: LFS101x Introduction to Linux edx course notes

Last Updated: August 02, 2018 by [Pepe Sandoval](#)



WANT TO SHOW SUPPORT?

If you find the information in this page useful and want to show your support, you can make a donation

Use PayPal

[Donate](#)



or

[Donate with Crypto](#)

This will help me to create more stuff and fix the existent content... or probably your money will be used to buy beer 😊

INDEX

[CHAPTER 1: THE LINUX FOUNDATION AND INTRODUCTION](#)

[CHAPTER 2: LINUX PHILOSOPHY AND CONCEPTS](#)

[CHAPTER 3: LINUX BASICS AND SYSTEM STARTUP](#)

[CHAPTER 4-5: GRAPHICAL INTERFACE & SYSTEM CONFIGURATION FROM THE GRAPHICAL INTERFACE](#)

[CHAPTER 6: PACKAGE MANAGEMENT AND COMMON APPLICATIONS](#)

[CHAPTER 7: COMMAND LINE OPERATIONS](#)

[CHAPTER 8: LINUX DOCUMENTATION](#)

[CHAPTER 9: PROCESSES](#)

[CHAPTER 10: FILE OPERATIONS AND FILESYSTEMS](#)

[CHAPTER 11: TEXT EDITORS](#)

[CHAPTER 12: USER ENVIROMENT](#)

[CHAPTER 13: MANIPULATING TEXT](#)

[CHAPTER 14: NETWORK OPERATIONS](#)

[CHAPTER 15-16: BASH SHELL SCRIPTING](#)



[CHAPTER 17: PRINTING](#)[CHAPTER 18: LOCAL SECURITY PRINCIPLES](#)

CHAPTER 1: THE LINUX FOUNDATION AND INTRODUCTION

INTRO

The kernel is just the core of the OS, it talks to the HW, makes the HW works, and it makes it able for you to run a program

LINUX FOUNDATION

It's a non-profit organization that sponsors Linux

- Supported by individual members and large companies (Intel, TI, NXP, Oracle, etc.)
- Organize technical events and develop training programs
- Manages kernel.org where the official versions of the Linux kernel are released.
- Runs Linux web page linux.com

LINUX DISTRIBUTIONS & FAMILIES

FEDORA

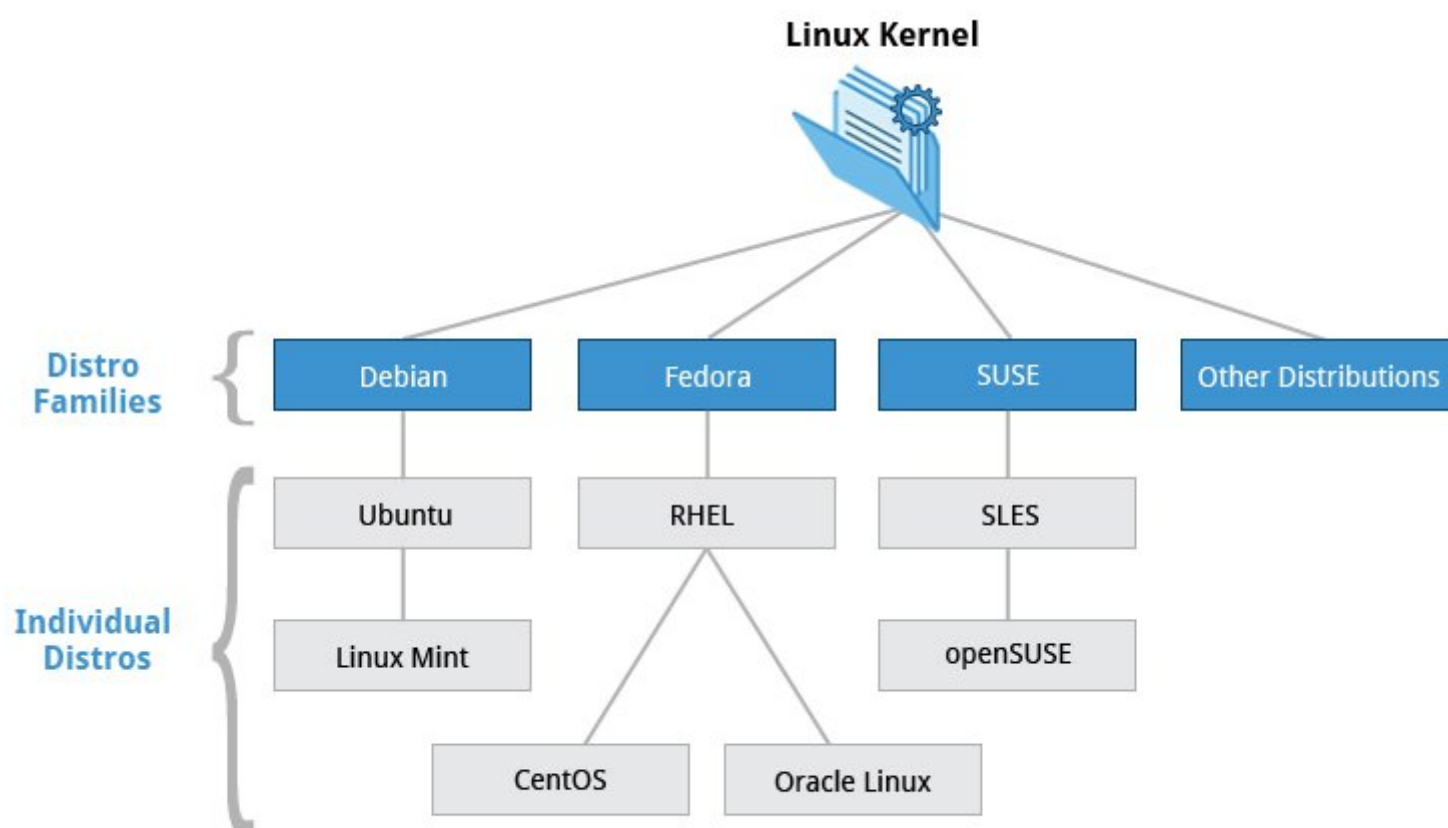
- Package manager: **yum** or/and **dnf**
- Directly associated with Red Hat widely used by enterprises which host their own systems.
- Examples: Fedora, RHEL (Red Hat Enterprise Linux), Centos

DEBIAN

- Widley used for servers and desktop
- Focuses on stability
- Package manager: DPKG-based **apt-get**
- Ubuntu uses **Unity** GUI based on **GNOME** and has been widely used for cloud deployments.
- Examples: Ubuntu

SUSE

- Package manager: **zypper**
- Widely used in the retail/sales sector (Ex clothing stores, supermarkets).
- Examples: SUSE, SUSE Linux Enterprise Server (SLES), OpenSUSE



CHAPTER 2: LINUX PHILOSOPHY AND CONCEPTS

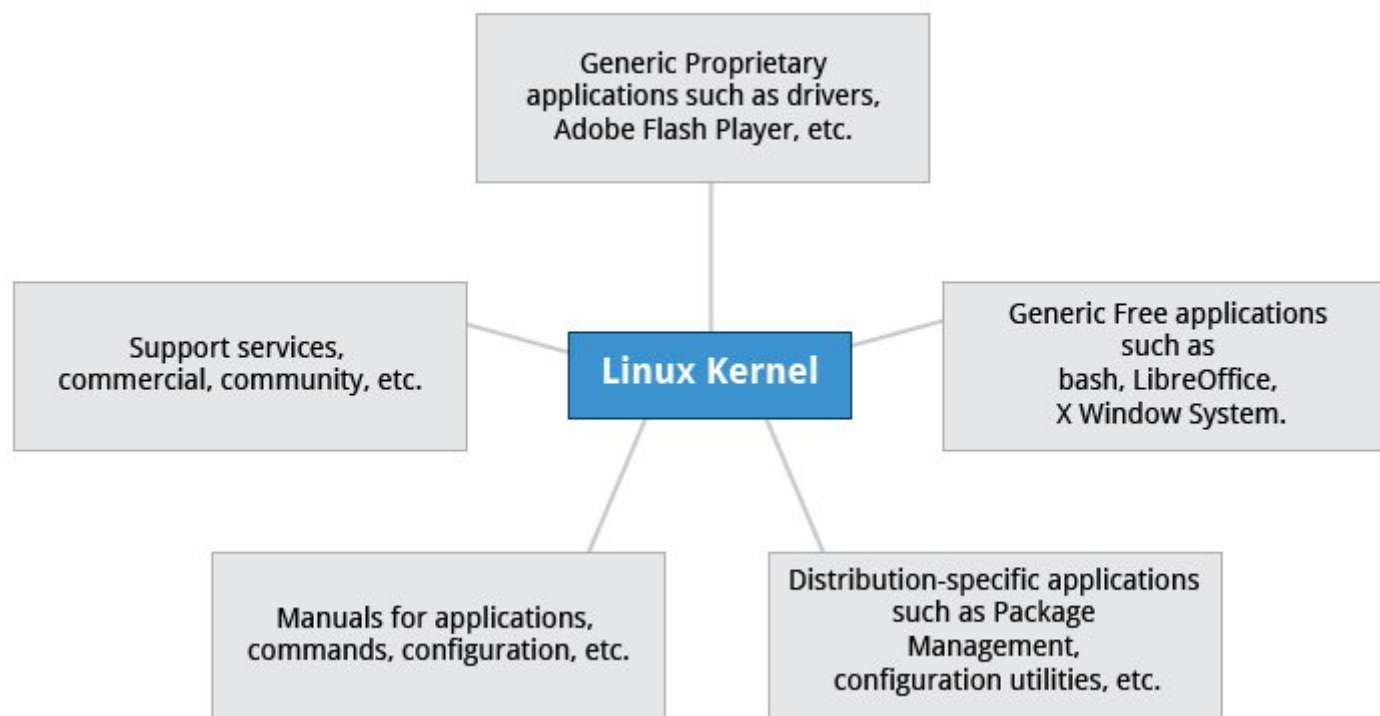
INTRO

- Linux was written to be a free and open source version of UNIX.
- **"All is a file"**: Linux makes its components available via files or objects that look like files. so it can be used with the same utilities used for regular files.
 - Processes, devices, and network sockets are all represented by file-like objects
 - In Linux all files are stored in one united directory tree, starting at **"/"**, the system's **root** directory
- **Kernel**: The core of the OS, it controls the HW and makes the HW interact with the Apps.
- **Boot Loader**: Program that boots/loads the OS (which is just another program). Bootloader Examples: **GRUB** and ISOLINUX.
- **Service**: Program that runs as a background process known as daemons.
- **File System**: Method for storing and organizing files (Ex. FAT, NTFS, ext3, ext4, etc).
- **X Window System**: Provides the toolkit and protocol to build GUIs
- **The desktop enviroment**: is a GUI on top of the OS. Ex. GNOME, Xfce, KDE...
- **Shell**: is a command line interpreter

DISTRIBUTION

A Linux Distro is combination of the kernel Linux with other system components or programs to make a Linux based OS. The programs are used for:

- File-related operations
- User management
- Software package management.



CHAPTER 3: LINUX STRUCTURE AND INSTALLATION

FILESYSTEM

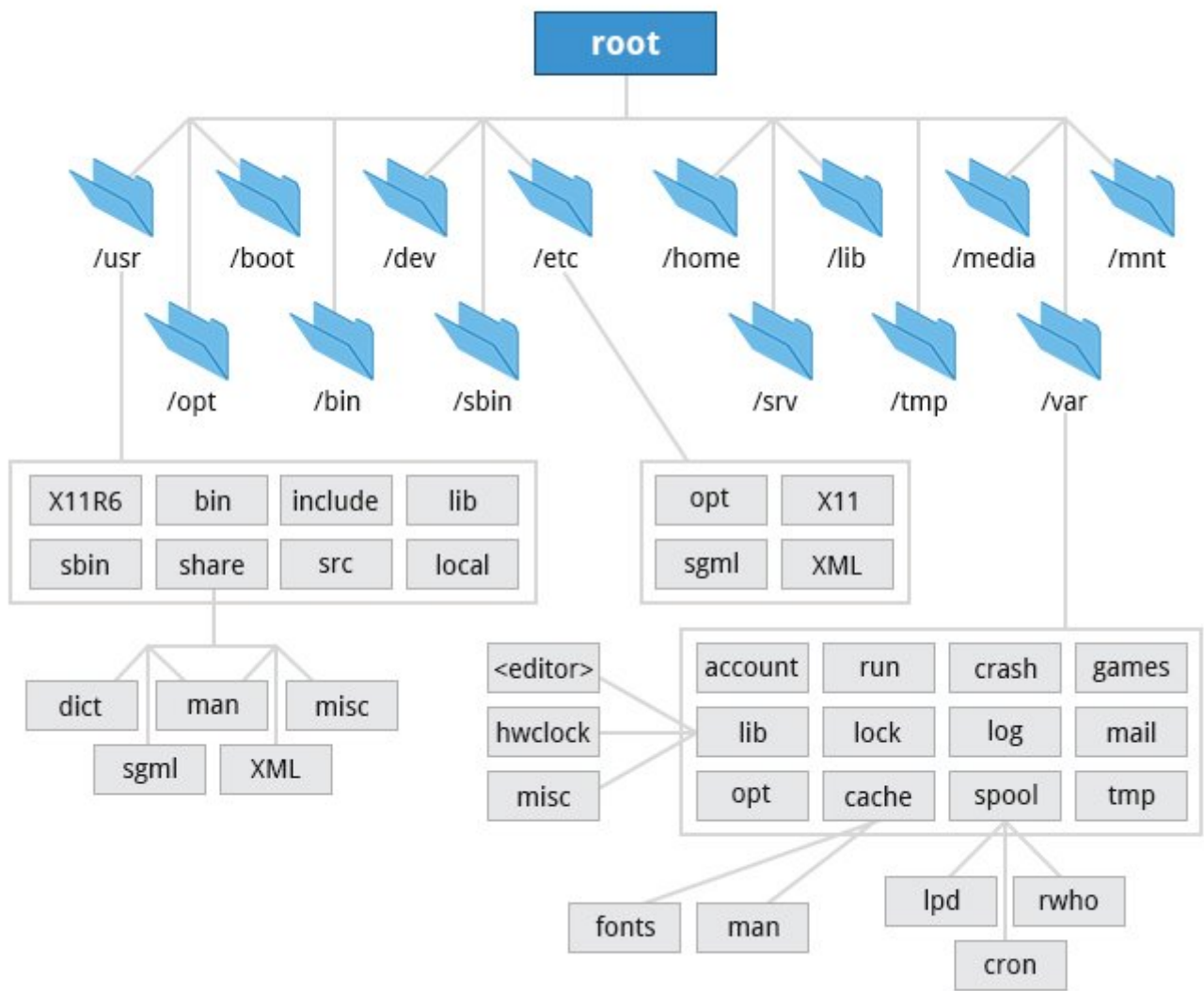
PARTITIONS AND FILESYSTEMS

- The filesystem is the method used for storing and organizing arbitrary collections of data in a usable form:
 - Conventional disk filesystem: ext3, ext4, NTFS.
 - Flash Storage filesystem: ubifs, JFFS2.
 - Database filesystem
 - Special puporse filesystem: sysfs, tmpfs, debugfs
- Partition: is **logical (NOT physical)** part of the Disk and the filesystem is a method for storing/finding files on that hard disk (usually in a partition).

THE FILESYSTEM HIERARCHY STANDARD

- Linux stores it's important files according to a standard ([FHS: Filesystem Hierachy Standard](#)), this ensures a consistency, of how the system is organized, between distros.
- New drives are mounted as directories in the single filesystem.
- Many distributions distinguish between core utilities needed for proper system operation and other programs. This is the reason for the **/usr**





THE BOOT PROCESS

It is the procedure for initializing the system. which consist of the following steps:

1. BIOS (BASIC INPUT/OUTPUT SYSTEM):

- The BIOS is a SW that's stored in ROM on the motherboard and is usually completely HW specific/dependent.
- Initializes the HW (Screen, keyboard, etc.)
- Tests the main memory (POST: Power On Self Test).

2. THE BOOTLOADER :

- The program responsible for loading into memory the Kernel
- It also loads into memory the initial RAM-based filesystem or **_initramfs_** so it can be used by the kernel
- Common bootloaders for Linux:
 - **GRUB** (for **GR**and **U*nified *B**oot loader)
 - ISOLINUX (for booting from removable media)
- Stored on one of the partitions of one of the Hard disks in the system. It can be stored in:
 - The boot sector in MBR (Master Boot Records) Systems.
 - UEFI or EFI (Unified Extensible Firmware Interface partition) systems.

STAGE 1:

MBR (MASTER BOOT RECORD) SYSTEMS

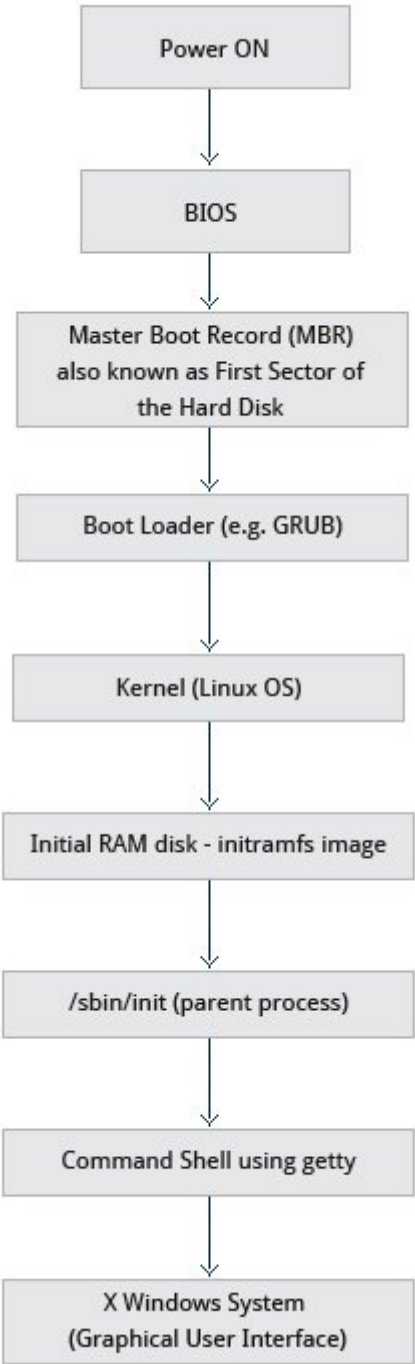
- The Bootloder resides in first sector of the hard disk known as the MBR which size is 512 bytes.
- The bootloader examines the partition table and find the bootable partion.
- Searches the second stage bootloader (e.g. GRUB) and loads it into RAM.

UEFI/EFI (UNIFIED EXTENDED FIRMWARE INTERFACE) SYSTEMS

- Reads the BOOT Manager data to determine which UEFI application will be launched and from which disk and partition the EFI partition can be found.
- Launches the found application (e.g. GRUB) and loads it into RAM.

STAGE 2:

- The second stage boot loader resides under **/boot**
- A screen to select an OS is presented to select the OS



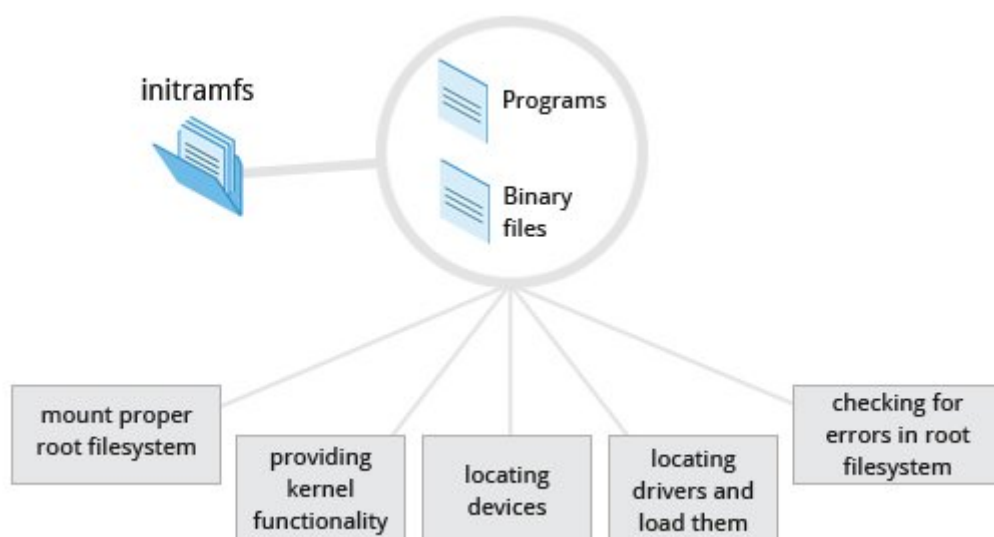
- The selected OS Kernel is uncompressed (Usually is compressed)
- Check and analyze the system HW and initialize any HW device drivers built into the kernel.
- The Kernel is loaded into RAM and the bootloader passes control to it.

3. THE KERNEL:

- After the Kernel is loaded it initializes and configures the memory and all the HW attached to the system (processor, I/O subsystems, storage devices, etc.)
- Loads some necessary user space apps

4. THE INITIAL RAM DISK:

- The initramfs contains programs and binary files that perform all actions needed to mount the proper root filesystem
- Essential programs of initramfs:
 - udev:
 - provides kernel functionality and device drivers for mass storage controllers
 - responsible for figuring out the present devices and find drivers they need to operate properly
 - after a root filesystem is found it is checked for error and mounted
 - mount:
 - instructs the OS that a filesystem is ready for use and associates it with a particular point in the overall hierarchy of the filesystem (mount point)?
- If the essential programs are successful the initramfs is cleared from RAM and the **init** (/sbin/init) program on the root filesystem is executed



5. /SBIN/INIT AND SERVICE

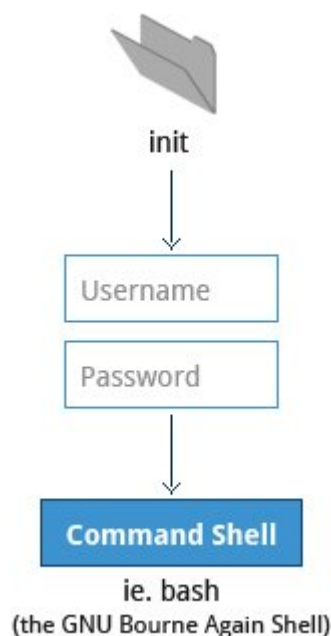
- Once the kernel has set up all its hardware and mounted the root filesystem, the kernel runs the `/sbin/init` program
- This **init** program becomes the initial process and starts other processes to get the system running.

Kernel processes are an exception because they are started directly by the kernel for managing internal OS system calls

- The init program is responsible for keeping the system running and for shutting it down cleanly
- Acts as a manager for all the **non-kernel** processes, cleaning up after them and controlling and restarting users log in and out, it does the same for other background system services.
- Handles the mounting and pivoting over the final real root filesystem?

6. COMMAND SHELL

- init is responsible for starting a number of text-mode login prompts (usually done by the **getty** program) which allow you to eventually get a command shell if the login is successful

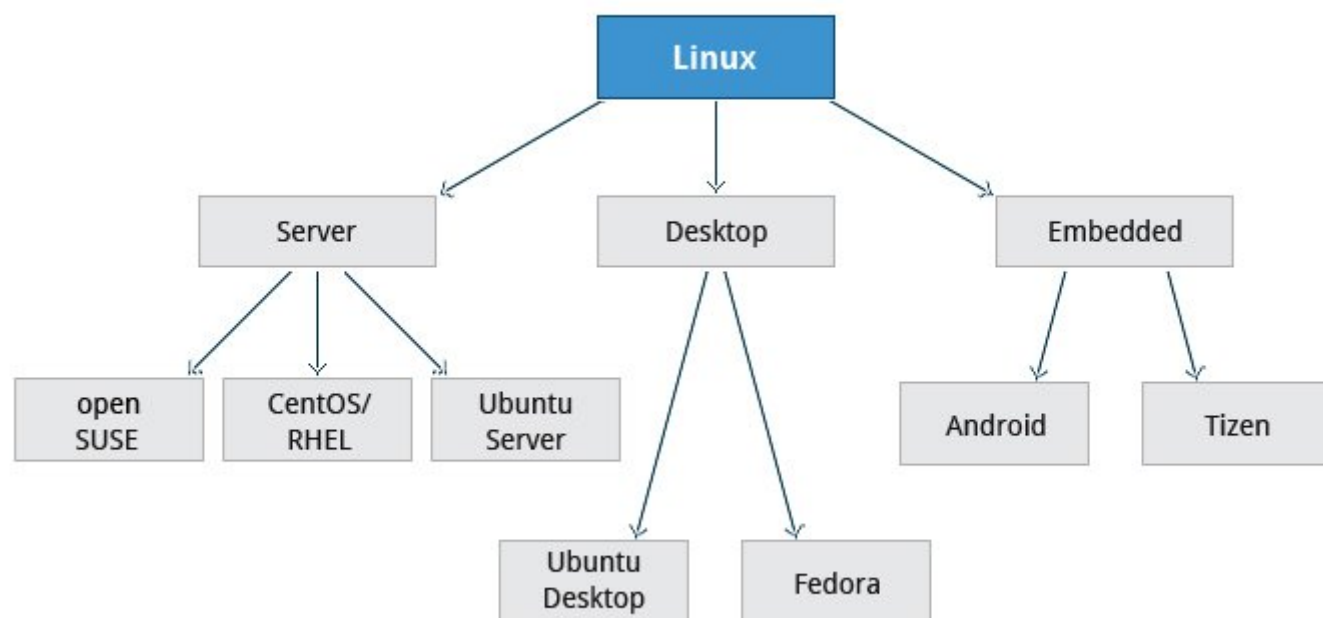


7. X WINDOWS SYSTEM OR X DISPLAY MANAGER

- Usually as the final boot process step the [X Windows System](#) is loaded which is the one that calls [display_manager](#) service

SELECTING A DISTRO

- Installing a Linux distro can be done completely automatically using a config file to specify installation options (preseeds file for the Debian-based systems like Ubuntu).
- What packages are important
- main function of the system (server, desktop, embedded)
- Available disk space and HW (x86, ARM, PPC, etc.)
- Support, updates and stability for release



SYSTEM STARTUP (SYSTEMD)

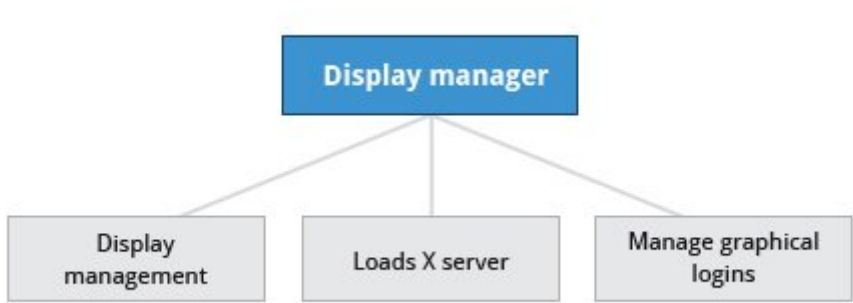
- It is the startup system used on most recent Linux distros
- It uses configuration files, which enumerate what has to be done before a service is started, how to execute service startup, and what conditions the service should indicate have been accomplished when startup is finished
- It uses the `systemctl` for most basic tasks
 - start/stop/restart: `sudo systemctl {start|stop|restart} <name_of_service>`
 - enable/disable: `sudo systemctl {enable|disable} <name_of_service>`

CHAPTER 4-5: GRAPHICAL INTERFACE & SYSTEM CONFIGURATION FROM THE GRAPHICAL INTERFACE

DISPLAY MANAGER / X DISPLAY MANAGER / X WINDOWS SYSTEM

- After the X Window System is loaded it usually calls the display manager service
- The Display Manager is the one responsible for:
 - Display Management (Keep track of the number of displays being provided)
 - Loading the X Server

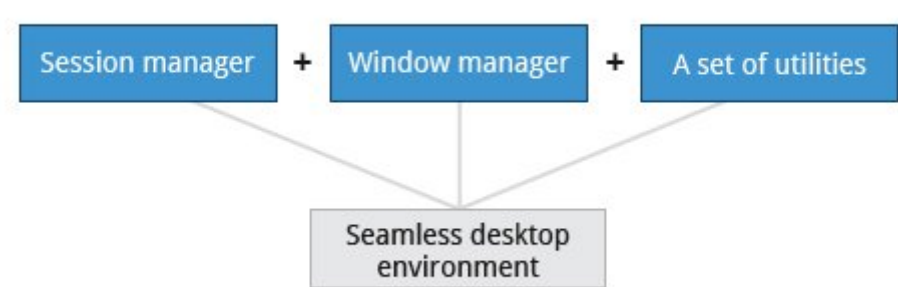
- Manage/handle graphical logins and starts appropriate desktop environment after log in
- Ubuntu uses **lightgdm** display manager
- The display manager is run as a service, you can stop the GUI desktop with the systemctl utility
 - `sudo systemctl {stop|start} gdm` (or `sudo telinit 3`)



DESKTOP ENVIROMENT

Consist of:

- **Session manager:** Starts and maintains graphical components
- **Window Manager:** Controls the placement and movements of windows, windows title bars and controls
- Usually a set of utilities, session manager, and window manager are used together as a unit to provide a desktop environment.



GNOME is an example of a Desktop enviroment which uses **gdm** display manager

SYSTEM CONFIGURATION MISCELLANEOUS INFORMATION

- **Nautilus:** File manager browser
- The X server handled by the windows X and which actually provides the GUI, uses the `/etc/X11/xorg.conf` file as its configuration file if it exists
- The **Network Time Protocol (NTP)** is the most popular and reliable protocol for setting the local time via Internet servers (the `/etc/ntp.conf` file contains configurations for this functionality).
- Linux uses **Coordinated Universal Time (UTC)** for its own internal time-keeping.
- A deleted file is moved to the trash usually at `/home/user/.local/share/Trash/files`
- Find out the current screen resolution: `xrandr | grep dim`
- the `init.d` directory is no longer used much in systemd-based systems (systemd), but is kept for backwards compatibility reasons

CHAPTER 6: PACKAGE MANAGEMENT AND COMMON APPLICATIONS

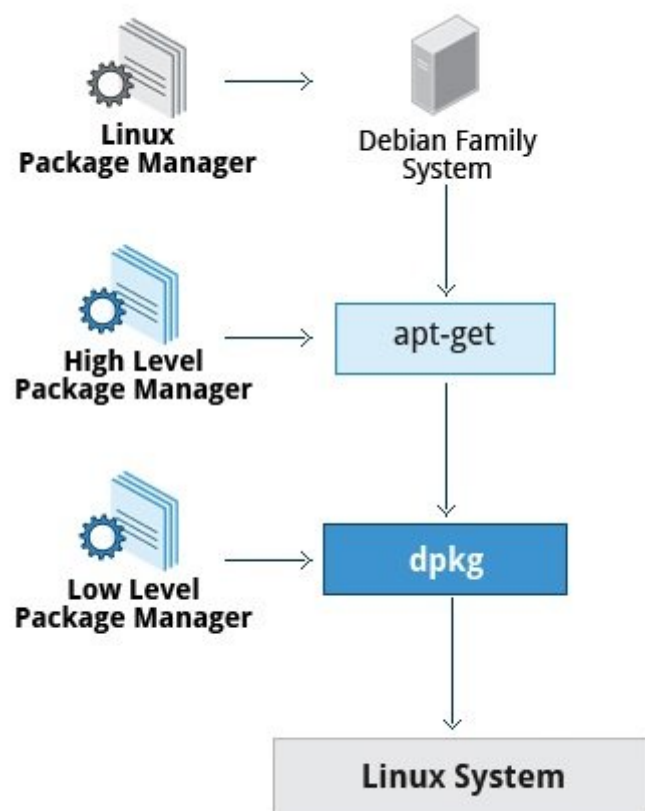
PACKAGES AND SW UPDATES

- A package is a piece of software that provides a piece of the system. Examples of packages:
 - Linux Kernel
 - The C compiler
 - The Firefox web browser
 - The USB library to interact with USB devices
- Packages have dependencies from other packages that must also be downloaded and installed before installing the package with dependencies.
- A package contains the files and other instructions needed to make one SW component work on the system

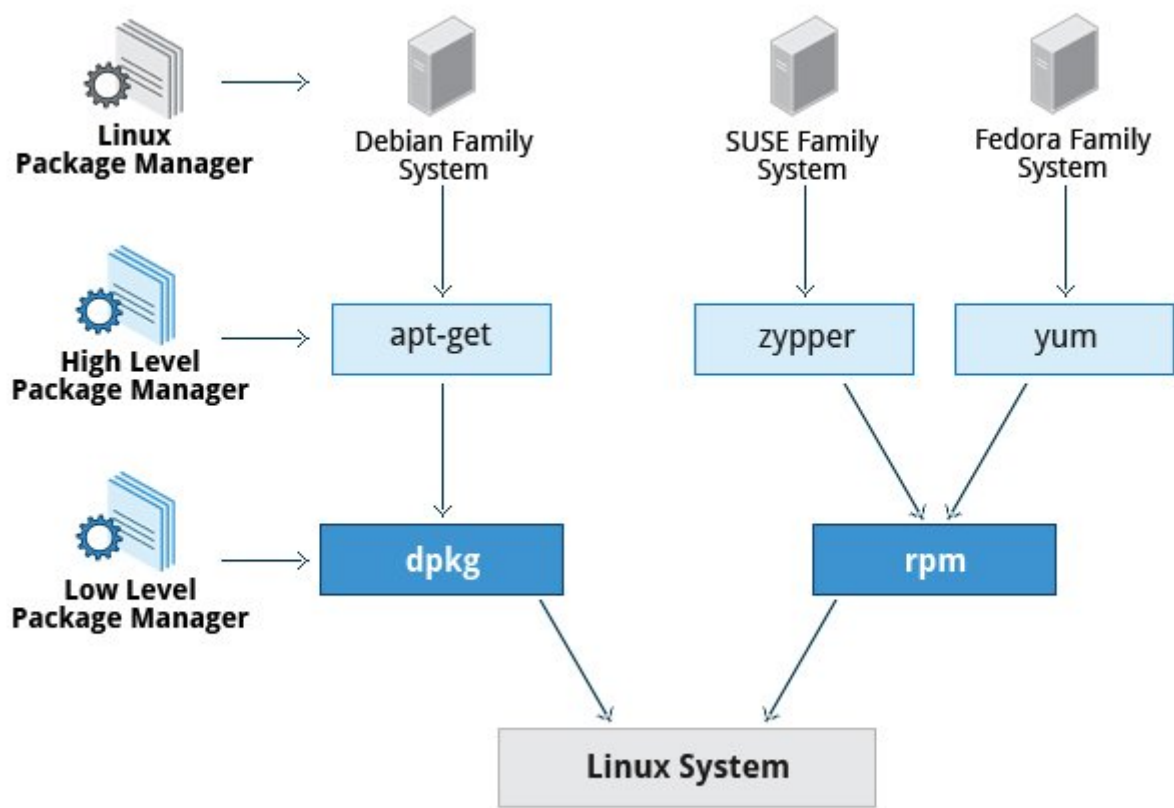
PACKAGE MANAGEMENT SYSTEM

- The packet manager is the utility that handles the downloading, unpacking and putting the unpacked pieces in the right places.

- Consists of two entities: 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-get, yum, or zypper) works with groups of packages, downloads packages from the vendor, and figures out dependencies
- The core parts of a linux distro and most of its add-on SW are installed via a Package Management System
- Debian Family System
 - **dpkg:**
 - low-level or underlying packet manager
 - Unpacks, installs, removes and build packages
 - It can't download or resolve dependencies
 - **Advance Package Tool (apt) :**
 - Build on top of **dpkg** (depends on it)
 - works with groups of packages
 - It can automatically download and install packages to figure out and resolve dependencies.
 - Tts native user interface is through the **apt-get** and **apt-cache** commands
 - Usually a user interface is created on top of it for an specific distro (SW update GUIs are an example of this)



- Fedora Family System
 - **Red Hat Package Manager (rpm):**
 - low-level or underlying packet manager
 - **Yellowdog Updater (yum):**
 - The high-level package manager differs between distribution but yum is commonly used
 - **Dandified Yum (dnf):**
 - It is the next-generation version of yum



PACKAGING COMMANDS

- low-level packaging commands operates on individual packages
- high-level packaging commands operates on multiple packages and deal with dependencies

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

COMMON 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
Ekiga	To connect to Voice over Internet Protocol (VoIP) networks
Hexchat	To access Internet Relay Chat (IRC) networks
GIMP	GNU Image Manipulation Program s a feature-rich image retouching and editing tool similar to Photoshop

CHAPTER 7: COMMAND LINE OPERATIONS

THE COMMAND LINE



- The most basic operation that can be done in a terminal is to execute a **Command** which is the name of a small program you are executing, they can have:
 - Options: Dictates how the program will behave
 - Arguments: What the command operates on
- Programs like **gnome-terminal** (default Terminal emulator on GNOME) or **xterm** are applications that emulates/simulates a standalone terminal with a window, they behave essentially as if you were logging into the machine at a pure text terminal with no running graphical interface
- **Virtual Terminals (VT)** are console sessions that use the entire display and keyboard outside of a graphical environment. These are not the same as the command line terminal window
 - **Set variables:** creating variables that can be used as arguments of a command
- The desktop manager is simply a service that can be stopped (`sudo service gdm stop` or `sudo service lightdm stop`)
- The `/etc/sudoers.d/<user_name>` file is the configuration file that enables/disables user account to use sudo.
 - `<user_name> ALL=(ALL) ALL`
 - `chmod 440 /etc/sudoers.d/<user_name>`

COMMANDS DICTIONARY

BASIC OPERATIONS

SSH

- ssh [username@remote-server.com](#)

SHUTDOWN

Shutdown system operations:

- **halt** and **poweroff** are equivalent to **shutdown -h** which shutdown the system (sudo)
- **reboot** is equivalent to **shutdown -r** which reboot system (sudo)

WHICH AND WHEREIS

- Executables programs usually live in:
 - `/bin`
 - `/usr/bin`
 - `/sbin`
 - `/usr/sbin`
 - `/opt`
- The following commands can be used to locate programs
 - **which:** Locates a program
 - **whereis:** Search for a program in broader range of system directories

CD AND DIRECTORY NAVIGATION

- **pwd:** Displays current directory
- **cd ~ /cd:** Change to Home directory
- **cd ..:** Change to parent directory
- **cd -:** Change to previous directory
- **cd /:** Change to root directory
- **ls:** List content of current directory
- **ls -a:** List ALL content of current directory (include hidden files)
- **ls -l:** Use long listing format
- **ls -li:** display [inode](#)
- **tree:** Display tree view of the filesystem (include files in the tree view)
- **tree -d:** Display folders in a tree view of the filesystem
- **pushd:** pushes the current directory to the history stack
- **popd:** pops the latest added directory of the history stack
- **dirs:** Displays the directories in the stack

SYMBOLIC LINKS

- **inode:** Unique file identifier
- **Hard links:** N/A
- **Soft links:** can point to objects even on different filesystems (or partitions) which may or may not be currently available c even exist



- In **<existent_file> <name_of_link_file>**: Creates hard link
- In **-s <existent_file> <name_of_link_file>**: Creates soft link

FILE STREAMS

- In Linux all open files are represented by **file streams/descriptors** which are just numbers used as identifiers
- File streams opened by default:
 - **stdin (0)**
 - **stdout (1)**
 - **stderr (2)**
- If other files are opened in addition to the default opened streams, they will start at file descriptor 3 and increase from there.

I/O REDIRECTION

- we can redirect the input or output of a stream using the **>***, ***<** and **&** (File descriptor identifier) operators. Examples:
 - `do_something > output-file`
 - `do_something < input-file`
 - `do_something <file-descriptor-number\> > output-file`
 - `do_something > all-output-file 2>&1` equivalent to `do_something >& all-output-file` which redirects file descriptor 2 to 1

PIPES (|)

- Used to have one program take as input the output of another.
- Allows to combine the actions of several commands into one using the pipe symbol |
 - `command1 | command2 | command3`

FILES

SEARCHING FOR FILES

LOCATE

- Search through a previously constructed DB of files and dirs
 - the DB is created by the **updatedb** program/command, usually automatically run by Linux (SUDO)
- Match entries with a specified string

WILDCARDS

Used to locate a filename containing a specific character(s)

- **?:** matches any **single** character
- ***:** matches any **string** character
- **[set] and [!set]:** matches any character in the set of or not (!) in the set of characters

FIND

- Search for files
 - `find <dir-to-search\> <options\> <input-string-no-quotes\>`
- arguments do not need quotes ""
- with no arguments it lists all files in the current dir and all of its subdirs. Common options:
 - name: name
 - iname: name ignoring case sensitive
 - type: dir (d), file (f), symbolic link (l)
- find is able to run commands on the found files using the **-exec** or **-ok** the ok option asks before performing the action
 - `find -name <in-string> -exec <command-to-exec> {} ';'`
 - `find -name <in-string> -exec <command-to-exec> {} \;`
 - find based on time (N is the number of days with +/- modifier for greater/less than that number):
 - **-ctime <N>**: last changed
 - **-atime <N>**: last accessed/read
 - **-mtime <N>**: last modified/written
 - check **-cmin**, **-amin** & **-mmin** for same functionality number of minutes
 - find based on size (with +/- modifier for greater/less):
 - **-size <size\>**: c(bytes), k(kilobytes) and M(megabytes) & G(gigabytes)

FILE OPERATIONS



VIEWING

- **cat:** Print content of a file no-scroll
- **tac:** Print content of a file no-scroll starting backwards
- **less:** pages and shows content of a file (/: search forward, ?: search backward)
- **tail:** print last lines of file
- **head:** print first lines of file

CREATE/REMOVE

- **mkdir/rmdir:** creates and removes EMPTY dirs.
- **rm:** removes dir or file
 - **rm -rf:** removes dir and all its content.
 - **rm -i:** interactively removes dir/file (asks/prompts before removing)
- **touch <filename>:** create a file
 - **touch -t <YYDDMMHHHH> <filename0> <filenameN>:** change timestamp of file
- **mv <old-name> <new-name>:** rename files

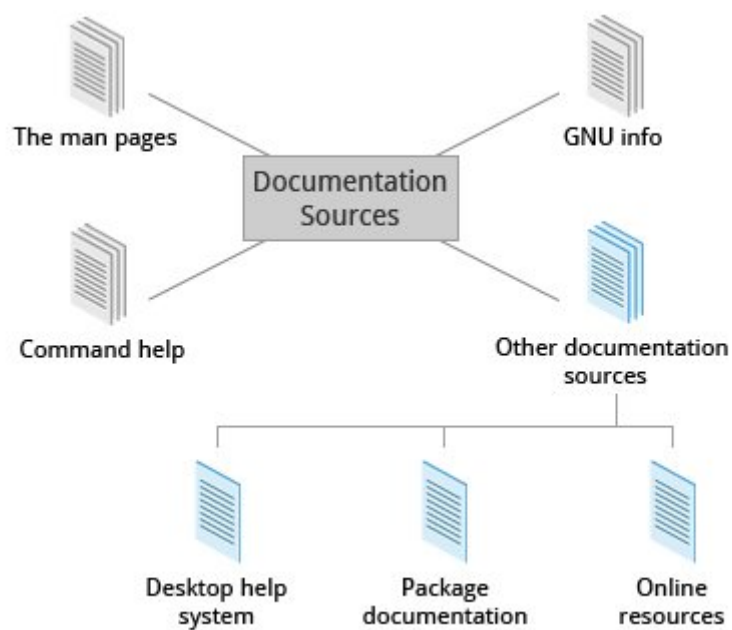
COMMAND LINE PROMPT STRING

- The **PS1** (**\$PS1**) variable is the character string that is displayed as the prompt on the command line.

CHAPTER 8: LINUX DOCUMENTATION

DOCUMENTATION SOURCES

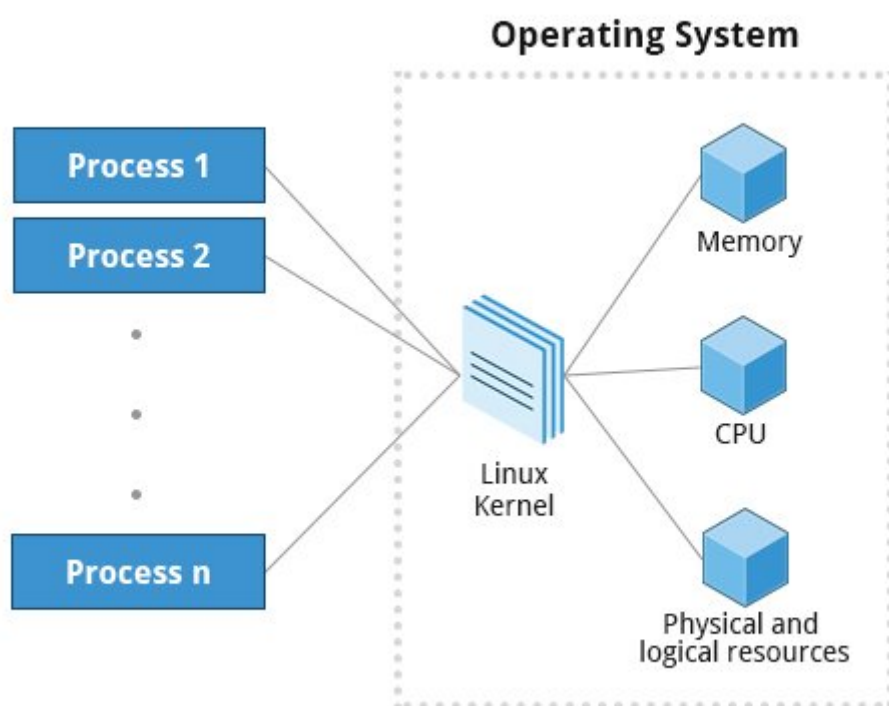
- **The man pages:**
 - Manual pages that provide in-depth doc ([man pages web version](#))
 - The man pages are divided into nine numbered chapters
 - Accessed through the **man** command/program that receives a topic name as an argument, searches and retrieves the information stored in the topic's man pages:
 - **man <string_topic>:** Returns the man page of the topic
 - **whatis** or **man -f <string_topic>:** Returns the man pages containing the given string in their name
 - **apropos** or **man -k:** Returns the man pages that discuss the given subject
 - **man <N> <string_topic>:** Display the page from a particular chapter
 - **man -a <string_topic>:** Display all pages with the given name in all chapters,
- **GNU Info:**
 - This is the GNU project's standard documentation format
 - Can be accessed through the info command.
 - **info <topic>:** Shows the info page for the specified topic
 - The n (next), p (prev) and u (up index) are used to move between nodes (info pages)
- **The help command and --help option:**
 - Option to use with other command to display a quick reference information
 - **man --help**
- **Other Sources:**
 - **gnome-help:** GUI based Linux Help System
 - The **/usr/share/doc** contains docs of the installed packages
 - web pages like:
 - help.ubuntu.com
 - LinuxCommand.org



CHAPTER 9: PROCESSES

INTRO

- All the application programs we are running are processes. Some processes are independent of each other and others are related
- **Process:** It's an instance of one or more related tasks (threads) executing on your computer. A process uses system's resources (memory, CPU, peripherals, etc) through the OS specially the Kernel which is responsible for allocating a proper share of these resources to each process and ensuring overall optimum utilization
- **Program:** or a **command**, it's NOT the same as a process but it can start several process simultaneously



PROCESS TYPES

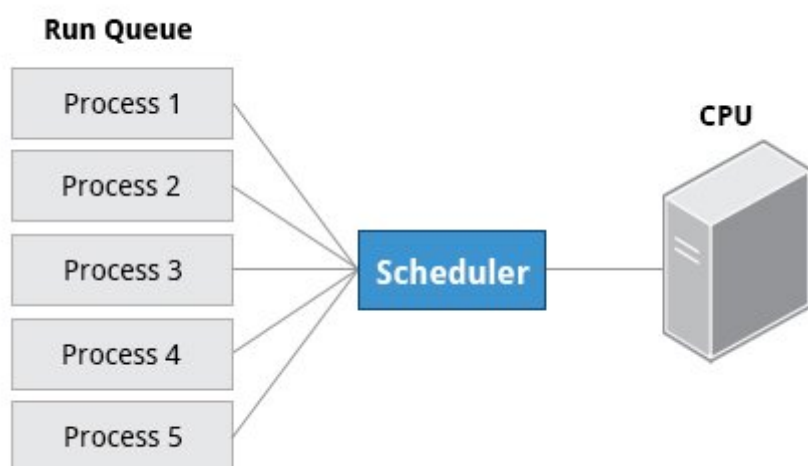
Process can be of different types according to the task being performed

- **Interactive process:**
 - Need to be started by an user.
 - Examples: bash, firefox, top...
- **Batch process:**
 - Automatic processes which are scheduled from and then disconnected from the terminal. Work on a FIFO scheme.
 - Examples: updatedb
- **Daemons:**
 - Server process that run continuously usually in the background
 - Many are launched during system startup and wait for requests of their services.
 - Examples: httpd, xinetd, sshd...
- **Threads:**
 - Lightweight processes
 - These are tasks that run under the umbrella of a main process, sharing memory and other resources
 - Usually started from another main process to process data in parallel
 - Examples: firefox, gnome-terminal-server...
- **Kernel Threads:**

- Not started nor terminated by the user. Perform kernel actions like moving a thread from one CPU to another or making sure I/O operations are completed
- Examples: kthreadd, migration, ksoftirqd...

SCHEDULING PROCESS

- A critical kernel function called the scheduler constantly shares the CPU between processes
- Process Status:
 - **running**: means it is either currently executing instructions on a CPU, or is waiting to be granted a share of time. On a run queue
 - **sleep**: generally when they are waiting for something to happen before they can resume. On a wait queue
 - **zombie**: if a child process completes, but its parent process has not asked about its state
- 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. A nice value of -20 represents the highest priority and 19 represents the lowest.



IDENTIFIERS

- **PID (Process ID)**: Unique ID that identifies the process, used to track process state, CPU usage, memory usage, where resources are located in memory, etc.
 - PID=1 denotes the **/sbin/init** process
- **PPID (Parent Process ID)**: Process (Parent) that started this process
- **TID (Thread ID)**:
 - The same as PID for single-threaded processes
 - For multi-threaded process each shares same PID but has unique TID
- **RUID (Real User ID)**: Identifies the user who starts the process
- **EUID (Effective User ID)**: Identifies the user who determines the rights for a user
- **RGID (Real Group ID)**: Identifies the group that starts the process
- **EGID (Effective Group ID)**: Identifies the Group that determines the rights for a group

PROCESS COMMANDS

- **ps** is the most common command to get information about the running processes
 - **-u**: option to display information of process for a specified user name
 - **-ef**: option display in detail all process in full detail
 - **-eLf**: displays on line of info for every thread (a process can have multiple threads)
 - **ps aux**: Displays all processes of all users
 - **ps axo <options>**: specify which attributes you want to view like: stat, priority, pid, pcpu, comm
- **top** for monitoring live updating info. It also provides interactive keys like: **A** (sort top consumers), **k** (kill), **o** (new sort option), ...
- The **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, threads are displayed {} and repeated process are not displayed
- To kill a process we can use the **kill** command but you can only do this to your own process those belonging to another user are offlimit unless you are root
 - **kill -SIGKILL <pid>**
- **Load average**: Determines how busy the system is, can be obtained with **w**, **top** or **uptime** and presents in 3 numbers average over the past 1 minute, over the past 5 minutes and over the past 15 minutes:
 - **=1*NUM_CPUs**: Fully subscribed but not overloaded
 - **>1*NUM_CPUs**: Overloaded, process are competing for CPU time
 - **<1*NUM_CPUs**: Under-loaded
- the **at** utility program is used to execute any non-interactive command at a specified time. You can see queued jobs with **atq** Ex:

```
at now + 2 days
commands
...
```

CRON

- it is a time-based scheduling utility program. It can launch routine background jobs at specific times and/or days on an on-going basis
- cron is driven by a configuration file `/etc/crontab` (cron table) which contains the various shell commands that need to be run at the properly scheduled times
- 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 (6 fields in total).
- `/etc/crontab` is the system wide crontab
- `crontab -e` edits the user crontab, these are crontab per user files stores in `/var/spool/cron/crontabs/<username>`
 - use `sudo crontab -e` to edit the root crontab

Field	Description	Values
MIN	Minutes	0 to 59
HOUR	Hour	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

```
30 08 10 06 * /home/user/script.sh # run the script.sh at 8.30am, 10-June, irrespective of the day of the week.
```

BACKGROUND AND FOREGROUND PROCESSES

- A **job** is just a command launched from a terminal window
- **Foreground** jobs run directly from the shell/GUI, and when one foreground job is running, other jobs need to wait for shell/GUI access
- **Background** jobs run with a lower priority which does no lock the shell/GUI
- by default all jobs are executed in the foreground. Use the `&` operator to put a job in the background or the `bg` and `fg` commands to run a process in the background and foreground,
- background jobs are connected to the terminal window if you log off, the jobs utility will not show the ones started from that window.
- The `jobs -l` utility can be use to view all the jobs running in the background

CHAPTER 10: FILE OPERATIONS AND FILESYSTEMS

FILES AND FILESYSTEMS

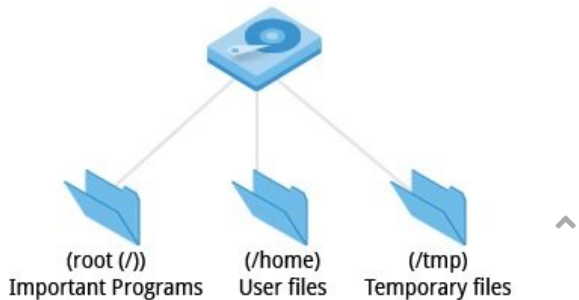
- In Linux (and all UNIX-like OS) everything is treated as a file (it doesn't mean it actually is a file)
- Whether you are dealing with normal data files and documents, or with devices such as sound cards and printers, you interact with them through the same kind of Interfaces and/or Input/Output (I/O) operations.
- The Linux filesystem is a tree that starts at the **trunk** or **root (/)**

FILESYSTEM HIERARCHY STANDARD (FHS)

- Provides a standadrd directory structure for the FS which provides consistency between systems and distros. Check [FHS](#) for more info.
- Linux support various filesystems:
 - Native Linux filesystems: ext3, ext4, btrfs, xfs
 - filesystems from other operating systems: vfat, ntfs, hfs

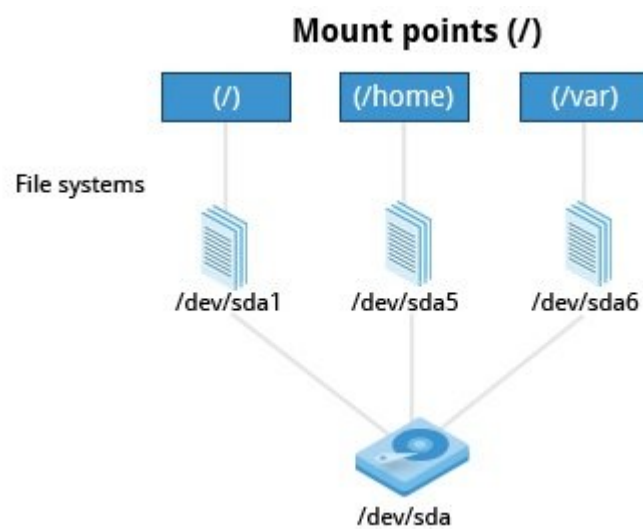
PARTITIONS

- Each filesystem resides on a hard disk partition
- The partitions help organize data according to the kind of data contained and how it is used. For example:
 - important data in a root partition
 - user files in home partition
 - temp files in a tmp partition.



MOUNT AND MOUNT POINTS

- **Mount/Mounting** is the operation that must be performed to start using a filesystem. It must be done at a **mount point** which simply is a directory where the filesystem will be attached (mounted)
- device-node: reference to partition with a filesystem in a device
- The mounting and retrieving partitions info:
 - **mount <device-node> <mount-point>**: mounts a filesystem (SUDO)
 - **mount** : Shows all presently mounted filesystems
 - **df -Th**: will display information about mounted filesystems
- The `/etc/fstab` file contains the filesystem table that contains the filesystems that will be mounted and configured automatically by the system.
- The `mount` command without options and arguments will list all of the currently mounted filesystems, along with information that will indicate whether they are mounted as read-only or writable



warning: If you mount a filesystem on a non-empty directory, the former contents of that directory are covered-up and not accessible until the filesystem is unmounted. Thus mount points are usually empty directories.

NETWORK FILESYSTEM (NFS)

- Mounting a home local directory on a server allows an user to have access to shared files or resources because other users home directories can also be mounted on the server

NFS ON THE SERVER

- the `etc/exports` file contains the directories and permissions that a host is willing to share with other systems over NFS. Entry file format:
 - `*.<domain>(<permissions>)`. Ecample `/projects *.example.com(rw)`
- To share a directory using NFS on the server:
 - starts NFS services: `sudo systemctl start nfs`
 - modify `etc/exports` file appropriately
 - notify update: `exportfs -av` or `sudo systemctl restart nfs`

NFS ON THE CLIENT

- To share a directory using NFS on the client:
 - mount the NFS:
 - modify `/etc/fstab` file:
 - `servername:/projects /mnt/nfs/projects nfs defaults 0 0`
 - manually mount the NFS:
 - `sudo mount servername:/projects /mnt/nfs/projects`

PROC FILESYSTEM

- It's a pseudo filesystem because it has no permanent presence on disk
- Mounted on `/proc`
- Contains virtual files (exists only in volatile memory) that permit viewing constantly varying kernel data
- Contains files and dirs that mimic kernel data structs and config info so it can be viewed.
- Contains runtime system info like: system memory, devices mounted, hardware configuration, etc
- It does **NOT** contain real files, but runtime system information
- There is a directory for every process (`/proc/<Process-ID-#>`)
- The information on proc is never stored permanently on disk
- Important files:
 - `/proc/cpuinfo`
 - `/proc/interrupts`
 - `/proc/meminfo`
 - `/proc/mounts`
 - `/proc/partitions`
 - `/proc/version`
 - `/proc/sys`

FILESYSTEM ARCHITECTURE

- Each user has a home directory under /home
- the /root dir is just the root user's home directory
- The /home dir is usually mounted as a separated FS on its or shared remotely though NFS

LINUX DIRECTORIES

/BIN AND /SBIN

- **/bin**: Contains executable binaries and essential commands required by all system users
- **/sbin**: Contains essential binaries related to system administration
- **/usr/bin**: Contains non-essential commands used in single-user mode
- **/usr/sbin**: Less essential system admin programs

On the newest Linus distributions, /usr/bin and /bin are actually just symbolically linked together, as are /usr/sbin and /sbin.

/DEV

- Contains **device nodes** which are pseudo files used by most HW and SW devices except for network devices.
- Contains entries created by the **udev** system which creates and manages device nodes on Linux when devices are found.
eg:
 - /dev/sda1 (first partition on the first hard disk)
 - /dev/lp1 (second printer)
 - /dev/dvd1 (first DVD drive)

/VAR AND /ETC

- **/var**: Contains files expected to change in size and content as the system is running.
 - System log files: **/var/log**
 - Packages and database files: **/var/lib**
 - Print queues: **/var/spool**
 - Temp files: **/var/tmp**
- **/etc**: Home for the system configuration files. It does not contain binary programs
 - **resolv.conf** tells the system where to go on the network to obtain host name to IP address mappings (DNS)
 - **passwd**, **shadow** and **group** are for managing user accounts
 - System run level scripts are found in subdirectories

/BOOT

- Contains 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

Each of the boot files has a kernel version appended to its name.

- It also contains the **Grand Unified Bootloader (GRUB)** files (such as /boot/grub/grub.conf or /boot/grub2/grub2.cfg)

/LIB AND /MEDIA, /RUN, /MNT

- **/lib**:
 - Contains libraries for the essential programs in **/bin** and **/sbin**.
 - Contains the dynamically loaded libraries (a.k.a. shared libraries or Shared Objects (SO)).
 - Kernel modules (kernel code, often device drivers, that can be loaded and unloaded without re-starting the system) are located in **/lib/modules**.
- **/media**: The directory where removable media, such as CDs, DVDs and USB drives are mounted
 - Newer Linux distributions place these mount points under the **/run** or **/run/media** directory.
 - **/mnt**: Used for temporarily mounting filesystems like network filesystems with NFS, not normally mounted or temporary partitions.

OTHER DIRS

- **/opt**: optional app SW packages.
- **/sys**: virtual filesystem that gives information about the system and HW (for param HW config and debugging).



- **/srv:** site specific data used by the system.
- **/tmp:** temporary files.
- **/usr:** Multi-user applications utilities, non-essential (not needed to boot) programs and scripts.
 - **/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.
 - **/usr/share:** Shared data used by applications, generally architecture-independent.
 - **/usr/src:** Source code, usually for the Linux kernel.
 - **/usr/X11R6:** X Window configuration files; generally obsolete.
 - **/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 commands on the system.

whats the difference between /sys and /proc/sys

MANAGING FILES AND DIRS

DIFF AND DIFF3

- **diff :** used to compare files.
- **diff -c <filename1> <filename2>:** compare files including context lines.
- **diff -r <filename1> <filename2>:** compare dirs and files recursively.
- **diff -i <filename1> <filename2>:** ignore case of letters
- **diff -w <filename1> <filename2>:** ignore differences in tabs and spaces
- **diff3 <my-file> <common-file> <others-file>:** compare 3 files at once using one as a base

PATCH & PATCHES

- a **patch** is a file that contains **deltas** which are changes required to update a file.
 - **diff -Nur <original_file> <newfile> <patchfile_name>:** Creates a patch
 - **patch -p1 < patchfile:** Apply patch to the entire directory
 - **patch originalfile patchfile:** Apply patch to specific file

FILE UTILITY

- The extension of a file does not determine the real nature of a file (a .txt can actually be an executable). So the **file** utility is used for this reason

It is a good practice to use the **-dry-run** option with the **patch** and **rsync** commands to test

BACKING UP AND COMPRESSING DATA

BACKING-UP

- **rsync <source> <destination>:** Used to synchronize entire directory trees, usually used with **-r** option (Basically a copy/backup command). the source and destination can be on a local machine or in a network machine

COMPRESSING

In Linux there are several methods for compressing data:

- **gzip:** Most used and very fast, produces **.gz** compressed files:
 - **gzip * <uncompressed-file>:** compress all files in the given dir (current dir by default), each file is compressed and renamed with **.gz** extension. use **-r** option to include the files inside directories in the compression.
 - **gzip -d <compressed-file>** or **gunzip <compressed-file>:** decompress
- **bzip2:** More space efficient than gzip but slower, same syntax and produces **.bz2** compressed files.
- **xz:** Most space efficient but also very slow, usually removes the input dir/file if operation succeeds and produces **.xz** compressed files:
 - **xz <uncompressed-file>:** compress the given dir/file (default compression level = -6).
 - use **-k** option to avoid removing dir/file if the operation succeeds
 - use **-d** option for decompression
- **zip:** Used for compatibility with other OS used by Linux.
 - **zip <zip-name>.zip <dir>:** compress the given dir/file and places its content in the .zip with the given name. if the .zip already exists by default it appends new content and replaces content with the same name.
 - **unzip <zip-name>.zip -d <extract-dir>:** extracts contents of .zip file and places them in the given directory



- **tar ("tape archive")**: used to group files in an archive called **tarball**. It can optionally compress while creating the archive or decompress while extracting:
 - **tar -xvf <input>.tar/.gz**: Extracts files in the given tar and places into a dir with the same name. It can be a **.tar** or a **.tar.gz/tgz**
 - **tar -cvf <output>.tar <input-dir>**: Creates and archive
 - **tar -zcvf <output>.tar.gz/tgz <input-dir>**: Creates and archive and compress it with **gzip**
 - **tar -jcvf <output>.tar.bz2/tbz <input-dir>**: Creates and archive and compress it with **bz2**
 - **tar -Jcvf <output>.tar.xz/txz <input-dir>**: Creates and archive and compress it with **xz**

DISK-TO-DISK COPYING (DD)

- The **dd** program is very useful for copying raw data (disk space).
- Use carefully because raw copy basically erases and replaces the data
 - **dd if= of= bs= count=<num-blocks>**

CHAPTER 11: TEXT EDITORS

CREATING FILES WITHOUT USING AN EDITOR

ECHO

Use **echo** to redirect the output to file, creating it at the same time, then appending lines to it.

```
echo line one > myfile
echo line two >> myfile
echo line three >> myfile
```

CAT

use **cat** and redirection

```
cat << EOF > myfile
> line one
> line two
> line three
> EOF
```

BASIC TEXT EDITORS: NANO AND GEDIT

NANO

- Text-terminal based editor that displays all the help you need at the bottom
- use the **nano** command to create a file and starts editing
 - **nano <filename>**

GEDIT

- Graphical editor that needs a Graphical Desktop environment to run
- use the **gedit** command to create a file and starts the **gedit** graphical interface
 - **gedit <filename>**

ADVANCED TEXT EDITORS

vi and **emacs** are more advanced text editors and have a basic purely text-based form that can run in a non-graphical environment.

VI

- Usually the actual program installed on the system is **vim** (vi Improved)
- **gvim** is a extended graphical interface version of **vi**
- All commands are entered through the keyboard

VI MODES

1. Command:

- It's the starting/default mode
- Each key is and editor command that can modify file contents

2. Insert:

- Type **i** to enter to *Insert* mode and **Esc** to exit. This mode is indicated by an **? INSERT ?** at the bottom of the screen.
- Used to insert text into a file

3. Line:

- Type **:** to enter to *Line* mode and **Esc** to exit
- Each key is an external command that includes operations such as writing file contents and exiting

vi useful commands:

- **vi myfile**: Start the vi editor and edit the myfile file
- **vi -r <file-name>**: Start vi and edit myfile in recovery mode from a system crash
- **:r <file-name>**: Read in the given file and insert at current position
- **:w! <file-name>**: Overwrite given file
- **:x** or **:wq**: Exit vi and write out modified file
- **:q**: Quit vi
- **:q!**: Quit vi even though modifications have not been saved
- **:0**: To move to beginning of file
- **:n**: To move to line n
- **:\$** or **G**: To move to last line in file
- **^L**: To refresh and center screen
- **CTRL-F** or **Page Down**: To move forward one page
- **CTRL-B** or **Page Up**: To move backward one page
- **j**: 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
- **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
- **x**: Delete N characters, starting at current position
- **w**: Delete the word at the current position
- **D**: Delete the rest of the current line
- **d**: 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.
- **!**: Executes a command from within vim
- **%**: Represents the file currently being edited.
- Search:
 - **/pattern**: Search forward for pattern
 - **?pattern**: Search backward for pattern
 - **n**: Move to next occurrence of search pattern
 - **N**: Move to previous occurrence of search pattern

EMACS

- Uses the **CTRL** and **Meta (Alt or Esc)** keys for special

vi useful commands:

- **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
- **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
- **CTRL-s**: Search forward for prompted pattern, or for next pattern
- **CTRL-r**: Search backwards for prompted pattern, or for next pattern
- **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

CHAPTER 12: USER ENVIRONMENT

USERS AND GROUPS

- The **whoami**, **who** and **who -a** commands are used to identify the current user and the currently logged-on users
- **id** can be used to get information about the current user
- **groups**: Collections of accounts with certain shared permissions, rights or privileges used by Linux to organize users.
 - the **/etc/group** has a list of groups, members of the groups and who controls the group
- Groups are collections of accounts with certain shared permissions.
- Every user has a unique user ID (**uid**) and a group ID (**gid**), the **/etc/passwd** and **/etc/group** files associated theses IDs with names.
- Permissions of files and dirs can be modified at group level

ADD & REMOVE USERS

- **useradd**: Add user, by default sets the home directory of the user, populates with basic files (copied from **/etc/skel**) and adds line to the **/etc/passwd** file.
 - **sudo useradd <user-name>**
- **userdel**: Removes a user but leaves the home directory of the user intact, **-r** option must be used to remove this dir.
 - **sudo userdel <user-name>**

ADD & REMOVE GROUPS

- **groupadd**: Creates a new group.
 - **sudo /usr/sbin/groupadd <new-group-name>**
- **groupdel**: Deletes a group.
 - **sudo /usr/sbin/groupdel <new-group-name>**
- **groups**: Checks at what groups the user already belongs to
 - **groups <user-name>**
- **usermod**: Adds a user to an already existing group
 - **sudo /usr/sbin/usermod -a -G <group-name> <user-name>**
- **groupmod**: Changes groups properties
- **usermod**: Modifies the groups that a user belongs to, it can be used to add a user to an already existing group or to delete a user from a group
 - **sudo /usr/sbin/usermod -G <group-name0>,<group-name1>,... <user-name>**



THE ROOT ACCOUNT

- The account that has full access to the system (a.k.a. **superuser**)

SU

- Used to fully become root
- It can be used to launch another shell as another user (type the password of the user you're becoming)

SUDO

- Executes **ONE** command with root privileges
- **sudo** configuration files are stored in the `/etc/sudoers` file and in the `/etc/sudoers.d/` dir.

STARTUP FILES

- Used to configure the user's environment, customize prompt, setting the default text editor, setting the path to find executable programs and define shortcuts and aliases.
 - init files in the `/etc` dir define global config settings for all users
 - init files in the `/home` dir include and/or override global config settings
- **Order of the Startup Files:** Evaluates whatever startup file that it comes across first and ignores the rest. These files are read and executed only when the user first logs onto the system in the listed order:
 1. `~/.bash_profile`
 2. `~/.bash_login`
 3. `~/.profile`
- Every time you create a new shell, or terminal window, you do not perform a full system login; only the `~/.bashrc` is read and evaluated in this case, usually this file is added to the `~/.bash_profile` file

ALIASES

- Used to create customized commands or modify the behavior of already existing ones.
- Usually aliases are placed in the `~/.bashrc` file
 - The `alias` command can be used to view the currently defined aliases
- To define an alias the `alias` command must be used, there should not be spaces in the = sign and quotes must be used:
 - Example: `alias ll='ls -aLF'`

ENVIROMENT VARIABLES

- Simply named quantities that have specific values and are understood by the command shell
- Some are pre-set by the system
- Character strings that contains information used by one or more apps
- To view currently set environment variables use: `set`, `env` and `export`.
 - **\$HOME:** environment variable that represents the home (or login) directory of the user
 - **\$PATH:** Contains an ordered lists of dirs which is scanned when a command is given to find the appropriate program. Each dir in the path is separated by colons(:).
 - `export PATH=<path-to-dir>:$PATH`
 - **\$PS1:** Prompt Statement (PS) environment variable is used to customize your prompt string. it can be used with single quotes and:
 - `\u` - User name
 - `\h` - Host name
 - `\w` - Current working directory
 - `!` - History number of this command
 - `\d` - Date
 - Example:

OLD_PS1=\$PS1
export PS1='\u@\h:\w\$ '

PS1=\$OLD_PS1
 - **\$SHELL:** points to the user's default command shell, contains the full pathname to the shell

SETTING ENVIRONMENT VARIABLES

- Variables created within a script or shell are only available to the current shell, to allow other shells or child-processes use **export** command or edit the `.bashrc` file
 - `export VARIABLE=value` (or `VARIABLE=value; export VARIABLE`)
 - Add the line `export VARIABLE=value` to the `~/.bashrc` file and run the file or starts a new shell

COMMANDS HISTORY

- Bash keeps track of previously entered commands and statements in a history stored in the `~/.bash_history` file. The command `history` can also be used
 - **!! (bang-bang):** Executes previous commands.
 - **CTRL-R:** Starts a reverse search for previously used commands.
 - **!?:** Refer to the last argument in a line
 - **!n:** Refer to the nth command line
 - **!string:** Refer to the most recent command starting with string
- To get information about the history file we can use the following env. variables:
 - **HISTFILE:** stores location of the history file.
 - **HISTFILESIZE:** stores the max number of lines in the history file.
 - **HISTSIZE:** stores the maximum number of lines in the history file for the current session.

KEYBOARD SHORTCUTS

- CTRL-L: Clears the screen
- 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 ONLY word before the cursor
- CTRL-U: Deletes from beginning of line to current cursor position
- CTRL-E: Goes to the end of the line

FILE PERMISSIONS

- In Linux every file is associated with a **user** who is the **owner**, also every files is associated with a **group** (a subset of all users)
 - **chown:** Used to change user ownership of a file or directory
 - `sudo chown <new-owner> <file>`
 - **chgrp:**Used to change group ownership
 - `sudo chgrp <new-group-owner> <file>`
 - **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 permissions:**
 - read (r): 4 value
 - write (w): 2 value
 - execute (x): 1 value
 - **Group permissions:** File permissions affect 3 groups of owners:
 - user/owner (u)
 - group (g)
 - others (o)
 - **Examples:**
 - `chmod <groups-letters> + <permissions-letters> <file>`
 - `chmod u+rx,g+rx,o+rx test1`
 - `chmod 755 test1`

CHAPTER 13: MANIPULATING TEXT

CAT

Short for concatenate usually used to view files contents but it can also:

- `cat file1 file2:` Concatenate and show output.
- `cat file1 file2 > newfile:` Combine and save output into a new file
- `cat cat file >> existingfile:` Append a file to the new of another file
- `cat > file:` It will create a file and any subsequent lines typed will go into the file until CTRL-D is pressed
- `cat >> file:` Any subsequent lines are appended to the an existing file until CTRL-D is pressed.
- `tac:` (cat spelled backwards) works as `cat` but prints the lines of a file in reverse order.

ECHO

Usually used to display texts but can also be used to create new files and append contents:

- `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.
- `-e` option enables the use of special characters like `\n`, `\t`, etc

SED

- **sed (stream editor)** used to modify (search and replace) the contents of a file usually placing the contents into a new file.
- Allows you to apply a **sed** command to a file to modify it and then move the final content to a desired output
 - `sed -e <command1> <filename>`: Execute sed commands on the file and outputs the result on the terminal.
 - `sed -f scriptfile <filename>`: Apply a script file with sed commands and outputs the result on the terminal.
 - use the `-i` option to store the result in the same file
- **sed** uses both `/` and `:` as delimitiers
 - `sed s:pattern:replace_string:g file1 > file2`
- Usually the result is stored in a new file and if everything is ok the first file is replaced
 - `sed s/pattern/replace_string/g file1 > file2`
 - `mv file2 file1`

Used to create and repeatedly edit and/or extract contents

AWK

- It is used to extract and print specific contents of a file
- 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
 - `awk '<command>' var=value file`: Apply the awk command to the file
 - `awk -F: '{ print $1 }' /etc/passwd`: Print first and seventh field of every line `-F`: option is used to specify a particular field separator
 - `awk -f scriptfile var=value file`: Apply a awk script file

FILE MANIPULATION UTILITIES

- **sort**: sorts the **lines** of a file. `sort <filename>`
- **uniq**: used to remove duplicate, consecutive lines in a file
 - `sort <file> | uniq` sort file and remove duplicates produces the same result as `sort -u file`
- **paste**: can be used paste horizontally or vertically, to combine or append contents of files
 - adds delimiters between paste `-d`
 - append contents (horizontal) `-s`
- **join**: is an enhanced version of paste to combine contents of files without repeating common fields or data.
- **split**: is used to break up/split a file into equal sized segments, creates a new set of files.


GREP

- Used to search strings within files
- `grep <pattern> <file>`
 - The pattern can be a combination of string and regexp
 - `-e` for explicit use of regular expression
 - `-v` option print all lines that do **NOT** match the pattern
 - `-C N` specify the number `N` of context lines.

EXTRA TEST UTILITIES

- **tr**: translates or replaces a set of characters to another set
 - `tr [options] 'set1' 'set2'` check [tr link](#)
 - `cat input_file | tr '\t' [:space:]` : replace tabs for spaces
- **tee**: takes the output of any command and saves it to a file besides display it on the standard output
 - `ls -l | tee newfile`
- **wc**: counts the number of lines, words and chars in a file, use `-l` for lines and `-w` for words
- **cut**: used for manipulating column-based files and extract specific columns.
- **tail**: prints the last few lines of each named file
 - `tail -f <file>`: To continually monitor new output in a growing file

LARGE FILES UTILITIES

- The **less** command is equivalent to the cat command but useful for large files because it does not need to place everything in memory and allows scrolling 

- `strings` command is used to extract the printable characters string found in a file
 - `strings book1.xls | grep my_string`
- the **z-type** command families is used to work with compressed files **zcat**, **zless**, **zdiff**, and **zgrep**.

REGULAR EXPRESSIONS

- Regular expressions are different from the wildcards
- Regular expressions are text strings used for matching a specific pattern they use **meta-characters**
- Search patterns:
 - `.(dot)` : Match any single character
 - `a|z` : Match a or z
 - `$` : Match end of string
 - `*` : Match preceding item 0 or more times
 - `.*` : Match string

CHAPTER 14: NETWORK OPERATIONS

INTRO CONCEPTS

- A network is a group of computers/devices connected together through communication channels such as cables or wireless
- Internet "the network of networks"

IP ADDRESSES

- The Internet Protocol (**IP**) address is a logical network unique address. The address is essential for routing packets.

IPV4

- Uses 32-bit addresses (4.3 billion)

- Addresses are divided in 4 octets and five classes

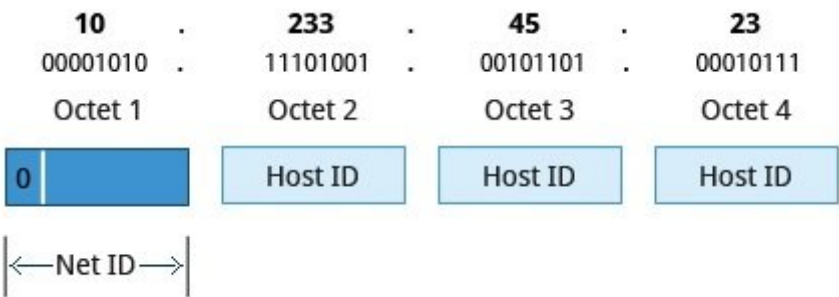
	Octet 1	Octet 2	Octet 3	Octet 4
Class A	Network ID	Host ID	Host ID	Host ID
Class B	Network ID	Network ID	Host ID	Host ID
Class C	Network ID	Network ID	Network ID	Host ID
Class D	Multicast addresses			
Class E	Reserved for future use			

- Network addresses (Net ID): used to identify the network
- Host address (Host ID): used to identify the Host in the network
- Multicast addr: Used to broadcast to multiple devices simultaneously

CLASS A NETWORK ADDRESSES

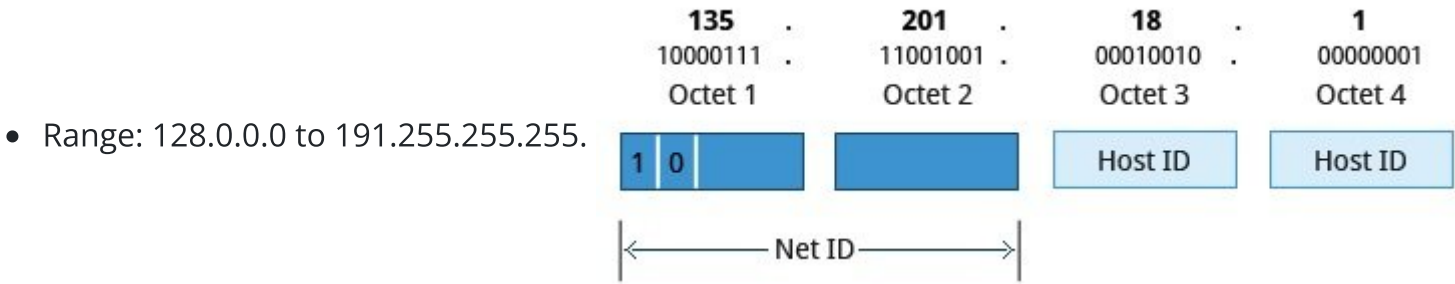
- First bit of Net ID = 0b
- First octet of an IP address as their Net ID and use the other three octets as the Host ID
- Maximum of 126 Class A networks available
- The 0000000 and 1111111 are reserved
- 2^{24} = 16.7 million unique hosts on its network.

- Range: 1.0.0.0 to 127.255.255.255.



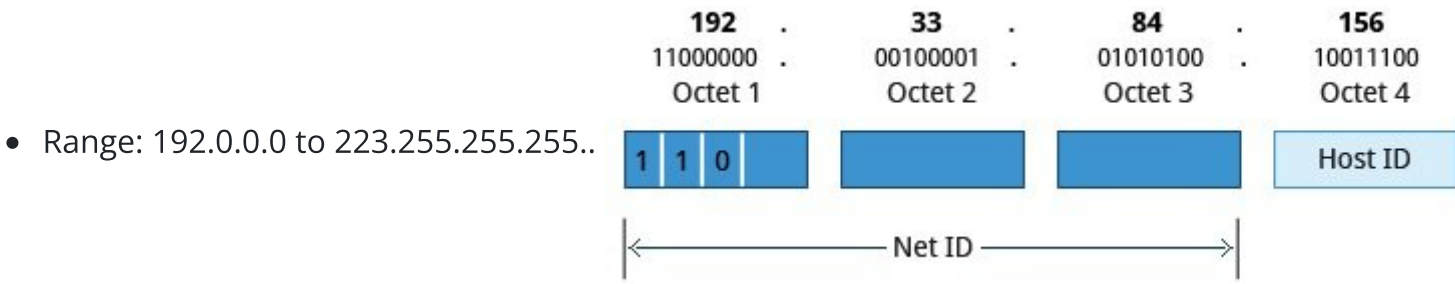
CLASS B NETWORK ADDRESSES

- First two bits of Net ID = 10b
- First two octets of the IP address as their Net ID and the last two octets as the Host ID
- Maximum of 2^{14} =16384 Class B networks available
- 2^{16} = of 65,536 unique hosts on its network.



CLASS C NETWORK ADDRESSES

- First three bits of Net ID = 110b
- First three octets of the IP address as their Net ID and the last octet as their Host ID
- Maximum of 2^{21} = ~2.1 million Class C networks available
- 2^8 = 256 of unique hosts on its network.



ALLOCATING IP ADDRESS

- A range of IP Addresses is requested from your ISP and the choice of which class of IP address depends on the size of the network.
- To assign IP Address:
 - Manully = static = never changing
 - `ipcalc <oct3.oct2.oct1.oct0/num-bits-network>`
 - For a class C: `ipcalc 192.168.0.0/24`
 - Dynamically = could change over time or every time the system reboots. **Dynamic Host Configuration Protocol (DHCP)** protocol is used for dynamically assigning of IP address

NAME RESOLUTION

- The Domain Name System (DNS) is used to convert numerical IP address into human-readable format (hostname)
 - `hostname` with no args is used to view the system's hostname. -

The special hostname **localhost** is associated with the IP address **127.0.0.1**, and describes the **machine you are currently on**

NETWORK INTERFACES

- They are connections channels between a device and a network
- Network Config files:** used to ensure that interfaces functions correctly
 - Basic network configuration file: `/etc/network/interfaces`
 - To starts network config: `etc/init.d/networking start`
 - View IP address: `/sbin/ip addr show`
 - View routing info `/sbin/ip route show`
 - Send a ping to a machine attached to the network `ping <hostname>`.

ROUTING

- A network requires the connection of many nodes.
- Servers maintain routing tables containing the addresses of each node in the network IP Routing protocols enable routers to build up a forwarding table that correlates final destinations with the next hop addresses.

ROUTE

- The `route` command is used to view and change the IP routing table
 - `route -n`: Show current routing table
 - `route add -net address`: Add static route
 - `route del -net address`: Delete static route

TRACEROUTE

- The `traceroute` command is used to nspect the route which the data packet takes to reach the destination host
 - `traceroute <domain>`: print the route taken by the packet to reach the network host

NETWORKING TOOLS

- `ethtool eth0`: Queries network interfaces
- `netstat -r`: Displays all active connections and routing tables.
- `nmap -sp <oct3.oct2.oct1.oct0/num-bits-network>`: Scans open ports on a network.
- `tcpdump`: Dumps network traffic for analysis.
- `iptraf`: Monitors network traffic in text mode.

IPV6

- Uses 128-bit addresses (3.4×10^{38})

IPv4 and IPv6 do **NOT** inter-operate, there are separated protocols and migrating from one to another requires significant effort

HOST

- A host is a computer with a Web server that handles all the services requested through the pages of a website.
- A host can also be the company that provides this service of having a computer as a web server

GRAPHICAL, NON-GRAPHICAL BROWSERS AND BROWSING TOOLS

NON-GRAPHICAL BROWSERS

- Used to retrieve, transmit and explore information resources usually on the WWW
 - **lynx**: Configurable text-based web browser
 - **links** or **elinks**: Based on lynx. It can display tables and frames.
 - **w3m**: Newer text-based web browser with many features.

WGET

- Used to handled download operations such as:
 - Download large files
 - Recursive downloads
 - Password required downloads
 - Multiple file downloads
 - Download a page `wget <url>`

CURL

- Used to obtain information about an URL
 - `curl <URL>`
 - Download a page: `curl -o saved.html <url>.`

FILE TRANSFER PROTOCOL (FTP)

- Protocol/Method for transferring files between computers using the Internet
- Built on a client-server model
- Can be used within a browser or by standalone client programs

FTP CLIENTS

- Enable you to transfer files with remote computers. Usually requires user name and password. Command line FTP clients:
 - ftp, ncftp, yafc
 - sftp: It is secure mode conection which uses Secure Shell protocol (**ssh**)

SSH

- Secure Shell is a cryptographic network protocol used for secure data communication
- Used for administering remote system
- Run a command on the remote system: `ssh <remotesystem> <command>`
- It needs the remote password
- **Secure Copy (scp)**: Program that uses **ssh** to securely move/copy files between networked hosts
- `scp <user@remotesystem>:/home/user/`

CHAPTER 15 & 16: BASH SHELL SCRIPTING

INTRO



- a **shell** is a command line **interpreter** which provides the user interface for terminal windows
- The first line of a script file starts with `#!/`, contains the full path of the command interpreter (Usually `/bin/bash`)
- All shell scripts generate a return value upon finishing execution; the value can be set with the `exit` statement
- By default return value is stored in the environment variable represented by `$?`
- To execute a shell script its executable permission must be set or the bash command can be used: `bash script.sh`
- The available shells for the syste, are listed in `/etc/shells`

SYNTAX

SPECIAL CHARACTERS

- `#`: Used to add a comment, except when used as `\#`, or as `#!` when starting a script
- `\`: Indicates continuation on to the next line
- `$`:Indicates what follows is a variable

CHAINING OF COMMANDS

- `::`: Used to separate commands and execute them sequentially, as if they had been typed on separate lines whether or not the preceding ones succeed.
- `&&`: operator will continue executing until something fails. Processing stops as soon as a condition evaluates to false
 - `make && make install && make clean`
- `||`: operator will only continue executing until something succeeds. Only one command is executed successfully. Processing stops as soon as anything is true
 - `cat file1 || cat file2 || cat file3`

PARAMETERS

- Within a script the parameters are represented with `$`:
 - `$0`: Script name
 - `$1, $2, $3, etc.` : First, Second, Third parameter, etc.
 - `$*`: All parameters
 - `$#` : Number of arguments

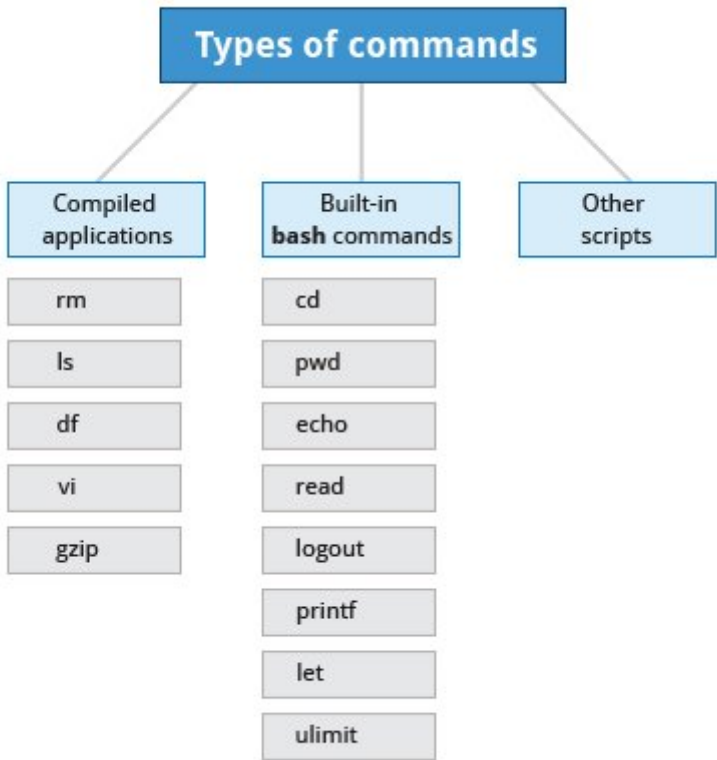
```
#!/bin/bash
echo "This is a sample script named: $0"
echo "Number of paramaters passed: $#"
```

```
echo "Script Parameter 1 is: $1"
echo "Script Parameter 2 is: $2"
```

BUILT-IN SHELL COMMANDS

Shell scripts execute sequences of commands and other types of statements.

- **Compiled applications:** Binary executable files that you can find on the filesystem (like under `/usr/bin`)
- **Built-in bash commands:** Which can only be used to display the output within a terminal shell



- Scripts from other interpreted languages

COMMAND SUBSTITUTION AND NESTING

- The result of one command can be part of another one. To do this for the inner command we can use backticks ``` or more modern (and preferred) `$()`

- The result of the innermost command will be executed in a new shell and the standard output of the shell will be inserted where the command substitution was done.
- Example:
 - `cd /lib/modules/$(uname -r)/`

VARIABLES

- The variables created within a script are available only to the subsequent steps of that script. We must use **export** to make them available
 - `export VAR=value`
 - `VAR=value ; export VAR`
- While child processes are allowed to modify the value of exported variables, the parent will not see any changes; **exported variables are not shared**, they are only copied and inherited

SCRIPT FUNCTIONS

- Also called subroutines is a code block that implements a set of operations
- It requires previous declaration:

```
#!/bin/bash
display () {
    echo "This is a sample function, script name: $0"
    echo "Number of func paramaters passed: $#"
```

CONDITIONALS

THE IF-THEN-ELIF-ELSE-FI STRUCTURE

- `[conditional]` and more modern `[[conditional]]` are used to delineate the test condition, spacing is necessary.
- The General structure of the `if-else` is the following, `elif` can also be used

```
if [ condition ] ; then
    statements
elif [ condition ] ; then
    statements
else
    statements
fi
```

or the equivalent

```
if [ condition ]
then
    statements
elif [ condition ]
then
    statements
else
    statements
fi
```

CONDITIONALS EVALUATION EXPRESSIONS

- **File Testing:** bash provides a set of file conditionals (check the full list with `man 1 test`)
 - `-e file`: Check if the file exists.
 - `-d file`: Check if the file is a directory.
 - `-f file`: Check if the file is a regular file (i.e., not a symbolic link, device node, directory, etc.)
 - `-s file`: Check if the file is of non-zero size.
 - `-g file`: Check if the file has sgid set.
 - `-u file`: Check if the file has suid set.
 - `-r file`: Check if the file is readable.
 - `-w file`: Check if the file is writable.
 - `-x file`: Check if the file is executable.

Examples:

```
#!/bin/bash
file="./test.txt"
if [ -f $file ] ; then
    echo "$file file exists"
fi
```

```
# test for NULL or uninitialized on first parameter passed
if [[ -z $1 ]];
then
    echo "No parameter passed."
else
    echo "Parameter passed = $1"
fi
```

```
# test for NULL or uninitialized on first parameter passed
if [[ $# -lt 1 ]];
then
    echo "# parameters passed ($#) is < 1"
else
    echo "# parameters passed ($#) is > 1"
fi
```

- **Numerical Testing:** bash provides a set of numerical conditionals that use the syntax: `exp1 -op exp2` the operators are:
 - `-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

Example:

```
#!/bin/bash
var=1
if [ $var -eq 1 ] ; then
    echo "var is equal to 1"
fi
```

- **Arithmetic Expressions:**
 - To evaluate an arithmetic expresion the built-in shell format must be used: `$((expr))`
 - The command `let` can also be used

Example:

```
#!/bin/bash
op1=$((1 + 3)); echo "operator 1 is: $op1"
let op2=( 2 + 1 ); echo "operator 2 is: $op2"
echo "Arithmetic example $(( $op1+$op2 ))"
```

BOOLEAN EXPRESSIONS

- `||` OR, `&&` AND, `!` NOT can be used in conditionals

Example

```
#!/bin/bash
AGE=26
if [[ $AGE -ge 20 ]] && [[ $AGE -lt 30 ]] ; then
    echo "you are in your 20s"
fi
```

STRING MANIPULATION & EVALUATION

- `==` and `!=` : Can be used to compare strings.
 - `if [[$color == "red"]]` or `if [["$color" == red]]`
 - `if [[$color != "red"]]` or `if [["$color" != red]]`
- bash allows string manipulation through the following operations
 - `[[! -z string2]]`: Test for NOT null string (zero length string)
 - `[[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.
- **\${string:offset:numchars}**: extract the numbers of characters specified and starting from the offset given
- **\${string##*<specialchar>}**: extract all the character after the given special character

THECASE STATEMENT

The General structure of the **case** statement is the following:

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

Examples:

```
#!/bin/bash
read -p "Respond 'yes' or 'no'? " resp
#resp="yes"

case $resp in
  yes) echo "You responded yes";;
  no)  echo "You responded no";;

  * )  echo "Other response" ;;
esac

#!/bin/bash
month=1
case $month in
  1) echo "January" ;;
  2) echo "February" ;;
  3) echo "March" ;;
  4) echo "April" ;;
  *) echo "Error. No month matches: $month" ;;
esac
```

LOOPING

FOR

The general syntax for the **for** statement is the following, the **for** variables does not need to be declared previously:

```
for variable-name in list
do
  commands
  ...
done
```

Example:

```
#!/bin/bash
i=1
for filename in $(ls)
do
  echo "filename $i is: $filename"
  i=$((i+1))
done
```

WHILE

The general structure of the `while` statement is the following:

```
while [ condition-is-true ]
do
    commands
    ...
done
```

Example:

```
#!/bin/bash
i=0
while [ $i -lt 10 ]
do
    echo "i values is: $i"
    i=$((i+1))
done
```

UNTIL

The general structure of the `until` statement is the following:

```
until [ condition-is-false ]
do
    commands
    ...
done
```

Example:

```
#!/bin/bash
i=10
until [ $i -lt 0 ]
do
    echo "i values is: $i"
    i=$((i-1))
done
```

DEBUGGING

- In bash we can run scripts in debug mode by doing `bash -x ./script-file`
- inthe script we can turn on/off the debugging mode with the following commands
 - `set -x`: Turns **ON** debugging
 - `set +x`: Turns **OFF** debugging

INPUT AND OUTPUT REDIRECTION

- To redirect the output of a program we use the `>` and `>>` operator so instead of directing its output to its default output stream we directed to a file or to another program, same case applies for input redirection.
- When redirecting to a file the `>` create/overwrite a file and the `>>` operator appends to the file
- In Linux all programs that run are given three open file streams
 - **stdin (0)**: By default keyboard for programs that run from terminal
 - **stdout (1)**: By default screen for programs that run from terminal
 - **stderr (2)**: Where output error messages are saved

EXTRA SHELL UTILITIES

- **mktemp**: Used to create temporary files and directories with random names
 - `TEMP=$(mktemp /tmp/tempfile.XXXXXXXXXX)`: Create a temporary file. Use the `-d` option for a temp dir
- **/dev/null**: A black hole use to discard the output of a command but **errors** will still appear
- **\$RANDOM**: Used for generating random numbers
 - `$(($RANDOM%<range>))`

CHAPTER 17: PRINTING



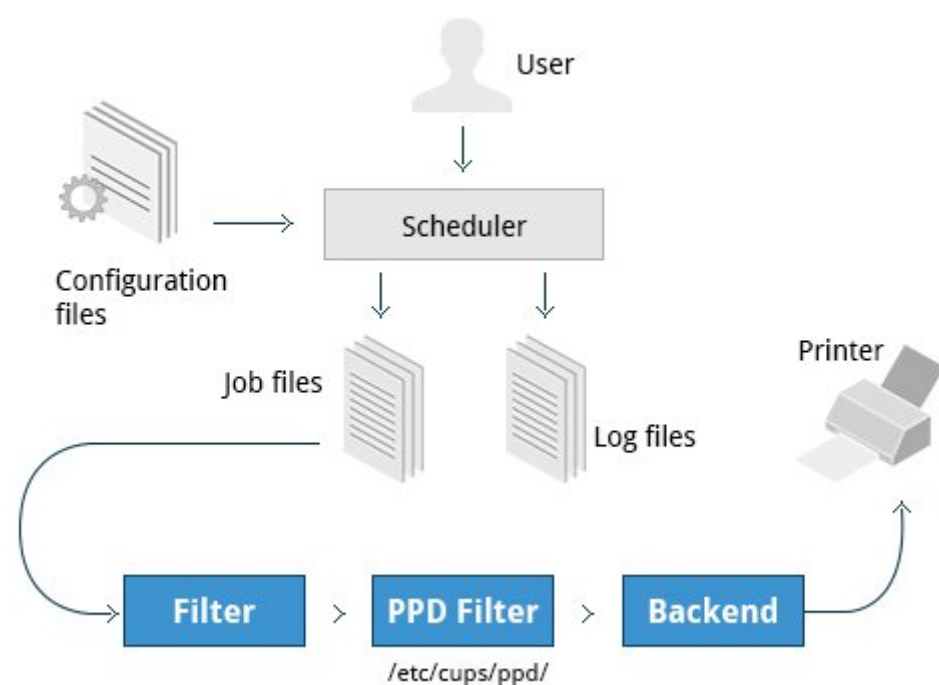
Printing itself requires software that converts information from the application you are using to a language your printer can understand. The Linux standard for printing software is the **Common UNIX Printing System (CUPS)**.

- Printers manufactured by different companies may use their own particular print languages and formats.

CUPS: COMMON UNIX PRINTING SYSTEM

- It is the Linux standard software for printing
- CUPS is the software that is used behind the scenes to print from applications
- It converts page descriptions produced by your application and sends the information to the printer
- Requires:
 - **Configuration Files:** used by the CUPS scheduler and stored at `/etc/cups/`:
 - `printers.conf`: contains printer specific settings for each printer there is a description for the printer's status and capabilities; generated only after adding a printer
 - `cupsd.conf`: system settings not printer specific like which systems can access CUPS network capabilities, how printers are advertised on the local network, etc.
 - **Scheduler:** manages print jobs and manages print jobs and the flow of data through all CUPS components.
 - **Job Files:** Stores print requests as files under the `/var/spool/cups`
 - **Log Files:** Placed in `/var/log/cups` and are used by the scheduler to record activities that have taken place like access, error, and page records.
 - **Filter:** Used to convert job file formats to printable formats.
 - **Printer Drivers:** Contain descriptions for currently connected and configured printers, usually stored under `/etc/cups/ppd/`
 - **Backend:** It is the via used to send data to the printer and also helps to locate devices connected to the system.
- When you execute a print command, the scheduler validates the command and processes the print job creating job files according to the settings specified in configuration files. Simultaneously, the scheduler records activities in the log files. Job files are processed with the help of the filter, printer driver, and backend, and then sent to the printer.
- CUPS web interface is available at: <http://localhost:631>

```
sudo /etc/init.d/cups {start | restart | status stop}
sudo update-rc.d cups restart {enable | disable}
```



- The commands `lp`, `lpr` and `lpstat -a` are used for printing and print status

PDF EDITING

the **pdftk** is a great tool for manipulating PDF files it can merge, split, rotate, encrypt, repair, pull pages, export data, etc. [pdftk](#)

CHAPTER 18: LOCAL SECURITY PRINCIPLES

USER ACCOUNTS

- Linux identifies each user with an user ID (**UID**) and uses a database that associates each **username** with each **UID**
- Creating a user implies adding new user information is added to the user database, creating a home dir and populated with some essential files
- The `/etc/passwd` file stores the following essential fields for each user:
 1. **Username:** User login name (between 1 and 32 characters)
 2. **Password:**
 - User password in *encrypted* format

- The character **x** if the password is stored in the `/etc/shadow` file

3. User ID (UID):

- UID 0: Reserved for root user
- UID [1,99]: Other predefined accounts
- UID [100,999]: System accounts and groups
- UID [1000,+inf]: Normal users

4. Group ID (GID): Primary group ID; the ID stored in the `/etc/group` file

5. User Info: Optional field for extra information such as the complete name of the user

6. Home Dir Path: The absolute path location of user's home directory

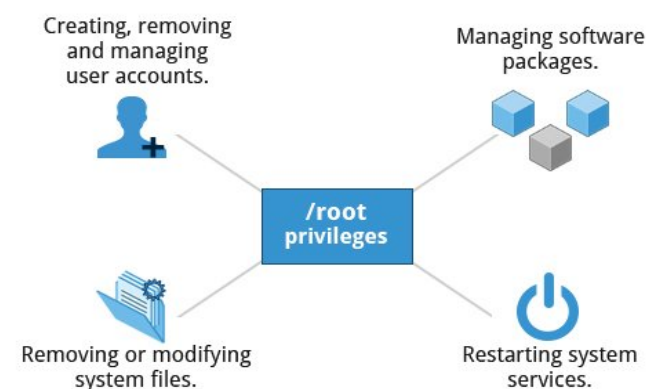
7. Shell Path: The absolute location of a user's default shell

TYPE OF ACCOUNTS

- The `last` utility, which shows the last time each user logged into the system
- Linux distinguishes between four types of accounts to isolate processes and workloads:
 1. root
 2. System
 3. Normal
 4. Network

ROOT

- Most privileged account with authority over the entire system (aka **superuser**), so it has **NO security restrictions** upon it
- When signed as root the shell displays a `#`
- Has the ability to carry out all System Administration. Privileges (root) are required to perform tasks such as:
 - Add/remove accounts
 - Change user passwords
 - Remove/Modify system files
 - Managing Software packages and Installing SW
 - Restarting system services
 - etc.



SU

- Elevates the privileges of a user indefinitely (until the user wants to).
- The **root account password** is needed.

SUDO

- Elevates the user privileges for **ONE** command. Calls to **sudo** trigger a lookup in the `/etc/sudoers` file, or in the `/etc/sudoers.d` directory, which first validates that the calling user is allowed to use sudo and that it is being used within permitted scope
- The **user account password** is needed NOT the root password
- Has extensive logging features, that means it keeps track of when the command is used and if it was or not successful. This info is stored in the `/var/log/secure` and `/var/log/auth.log` files. The typical logging messages contains:
 - Calling date, time and username
 - Terminal info
 - Working directory
 - User account invoked
 - Command with arguments
- Users' authorization for using sudo is based on configuration info stored in the `/etc/sudoers` file and in the `/etc/sudoers.d` dir:
 - basic entry:
 - `who where = (as_whom) what`
 - the `visudo` command can be used to properly edit the **sudoers** file this file **affects ALL users**
 - Adding a file to the `/etc/sudoers.d` with same name of the user that contains the individual user's sudo configuration will **ONLY affects this user**.

CREATING A NEW USER IN LINUX

To create and account using `useradd`, as **root** only:

1. `useradd <username>`
2. `passwd <username>`
3. Enter password when prompted and retype to confirm

SUID (Set owner User ID upon execution) is a special kind of file permission given to a file that 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.

PROCESS ISOLATION

- Processes are naturally isolated, one program cannot access the resources of another process even if both are running with the same user privileges
- It is the reason Linux is considered more secure than other OS

Additional security mechanism:

- Control Groups (cgroups):** Allows to group processes and associate finite resources to each cgroup.
- Virtualization:** HW is emulated so entire systems can be run isolated, basically run isolated Virtual machines on physical host
- Linux Containers (LXC):** Makes it possible to run multiple isolated Linux systems (containers) on a single system using **cgroups**

HARDWARE DEVICES

- Linux limits user access to HW devices in a manner that is extremely similar to a regular file
- HW interaction:
 - Applications interacts by engaging the filesystem layer (independent of the actual HW device),
 - The filesystem layer opens a device special file (**device node**) under **/dev** dir that corresponds to the device being accessed
 - Each device node has standard owner, group and permissions. Security applies as if it were a file

HARD DISK EXAMPLE

A Hard disk for example are represented as `/dev/sd*` which is the device node of the Hard Disk so theoretically if a user has the appropriate permissions it could do something like `echo hello world > /dev/sda1` which shows that this HW device can be interacted as if it were a file but this could actually destroy the FS. So the normal **reading and writing is done at a higher level through the filesystem, and never through direct access to the device node.**

PASWORDS

- Passwords are stored in an encrypted format in a secondary file named `/etc/shadow` and only those with root access can modify/read this file.
- Password Encryption:** Most Linux distributions rely on a modern password encryption algorithm called **SHA-512 (Secure Hashing Algorithm 512 bits)**
- Password Aging:** Method that ensures a password update every certain time. Uses the **chage** which configures the expiration date/time of user info
- Pluggable Authentication Modules (PAM):** Tool that automatically verifies that a password created/modified using **passwd** is sufficiently strong.

BOOT PROCESS PASSWORDS SECURITY (REQUIRING BOOT LOADER PASSWORDS)

To secure the boot process in GRUB version 1:

- Invoke the `grub-md5-crypt` enter and confirm password when prompted.
- Edit the `/boot/grub/grub.conf` file, add `password --md5`

To secure the boot process in GRUB version 2:

- Edit the system configuration files in `/etc/grub.d`.
- Run the `update-grub` command. Check [GRUB2-Pass](#) for example

HW VULNERABILITY

When HW is physically accessible, some security breaching types are:

- Key-logging:** Record real time activity of a HW device (key press)
- Network sniffing:** Capturing network packet level data
- Bootimg with rescue disk or USB key**
- Remounting and modifying disk content**

If you find the information in this page useful and want to show your support, you can make a donation

Use PayPal



or



This will help me to create more stuff and fix the existent content... or probably your money will be used to buy beer 😊

[About](#) · [Terms of Use](#) · [Privacy Policy](#)

Copyright © pepe-docs 2018. All Rights Reserved.

