Massive thanks to the Linux Foundation for offering: LFS101x (Introduction to Linux) // Date: 02 June 2020 // LinuxFoundation (credits to them) in partnership with edX // \_\_author\_\_ = Anvesh G. Jhuboo // Twitter: @anveshjhuboo // Gitlab : @neuralx // Github : @neuralx7

# Chapter 1 Summary

* The Linux Foudation is the umbrella organization for several open-source projects Like Linux, and its work today extends far beyond that to every layer of the software stack
* The 3 major distribution families within Linux are:
  + Red Hat
  + SUSE
  + Debian

# Chapter 2 Summary

* Linux is heavily influenced by UNIX
* Linux accesses many features and services through files, and file-like objects
* Linux is fully multi-tasking, multi-user OS, with built-in networking and service processes known as daemons
* Linux is developed by collaboration of developers around the world, with Linus Torvalds at the head. Technical Skill and a desire to contribute are the only qualification for participating.
* Common terms used in Linux are:
  + Kernel
  + Distribution
  + Boot loader
  + Service
  + Filesystem
  + X Window system
  + desktop
  + environment
  + command line
* A full Linux distro consists of a kernel and several other packages for file-related operations, user management, and software package management.

# Chapter 3 Summary

* A partition is a logical volume of the disk.
* A filesystem is a method of storing/finding files on a hard disk.
* Dividing the hard disk into partitions enables data to be grouped and separated as needed. When a failure or mistake occurs, only the data in the affected partition will be damaged while data elsewhere will likely survive.
* The boot process starts with the: 1) BIOS (initialization of hardware - also called POST) which triggers the : 2) Boot loader (Master Boot Record (MBR) searches for partition table then boot loader, like GRUB, for booting OS) to startup the : 3) Linux Kernel (Kernel is loaded, and takes control in RAM. It is uncompressed, then initializes any built-in device drivers) Which invokes the : 4) initramfs | initrd (Contains binary files and programs that perform all actions needed to mount proper root filesystem.) Which triggers : 5) init program (Handles mounting and pivoting over the final real root filesystem)

# Chapter 4 Summary

* GNOME is a popular desktop environment and GUI that runs on top the Linux OS
* The default display manager is 'gdm': ~> sudo systemctrl startstop gdm
* The Display manger handles:
  + Display management : (Keeps track of the displays)
  + X server : (Provides graphical services to applications, called X Clients) | being replaced by Wayland | use :~> : startx
  + Graphical logins : (Handles graphical logins, and appropriate desktop environments after login)
* Nautilus gives three format to view files:
  + Ctrl-F : (search box)
  + Ctrl-L : (specific directory)
  + Combine
* Deleting a file in Nautilus will automatically move the deleted files to the : ~> : cd '.local/share/Trash/files/'
* Latest modified file can be viewed under : ~> : var/log
* gedit is the default GNOME editor

# Chapter 5 Summary

* Access gnome-tweaks using Alt + F2
* The X server, which provides the GUI, uses : ~> : cd /etc/X11/xorg.conf (if it exists, this is only present in modern Linux distros under unusual circumstances)
* Linux uses Universal Coordinated Time (UTC) for its own internal time keeping, (Also look for CMOS bios)
* Network Time Protocal (NTP) is hte most reliable protocol for setting the local time by consulting internet servers. :~> : cd /etc/ntp.conf
* Network Manager handles network configuration files, and makes it easier and more uniform across distros.
* For Wired connections, the hardware interface and signal are automatically detected and the Network Manager sets the network settings using DHCP For static configurations that do not use DHCP, manual setup can be done easily through Network Manager. The Media Access Control (MAC) is the unique hex number of the network card. For wireless networks, you can view the list of available wireless networks and see which one you are connected to using Network Manager. VPN connections are also supported, including native IPSec, Cisco OpenConnect, Microsoft PPTP, OpenVPN
* For the Debian family system, the package management system is:
  + dpkg : (Debian Package) - low level
  + apt : (Advanced Package Tool) - high level
* For the Red Hat family system, it is:
  + rpm : (Red Hat Package Manager) - low level
  + yum | dnf | zypper | PackageKit - high level
* For OpenSUSE, it is:
  + rpm
  + YaST

# Chapter 6 Summary

* Linux supports several web browsers including Firefox, Chrome, Epiphany, Brave, w3m, lynx, etc
* Linux supports graphical email clients like Thunderbird, Evolution, Claws Mail, and text mode email clients like Mutt, and mail.
* Linux systems also support Filezilla, XChat, Pidgin, etc
* Linux offers LibreOffice
* Linux offers GNU Image Manipulation Program (GIMP) which is like Photoshop
* Linux offers suits of development applications and tools, like compilers and debuggers
* Linux offers sound players like Amarok, Audacity, Rhythmbos, and Spotify among others
* Linux offers movie players including VLC, Plex, MPlayer, Xine, Totem, etc
* Linux offers movie editors including Kino, Cinepaint, Blender, etc
* Other graphical utilities includ eog, Inkscape, convert, Scribus, etc

# Chapter 7 Summary

* Virtual Terminals (VT) are console sessions that use the entire display and keyboard outside of a graphical environment. They are considered "virtual" because, although there can be multiple active terminals, only one terminal remains visible at a time. One VT is reserved for the graphical enviroment. (Ubuntu uses VT 7) The VTs can be accessed using : ~> : Ctrl-Alt-F[2-7] To go back to your home VT: ~> : Ctrl-Alt-F2
* A terminal emulator program on a graphical desktop works by emulating a terminal within a window on the desktop. Access the gnome-terminal by doing : ~> : Alt-F2 and then type 'gnome-terminal' Other terminal programs include: xterm, rxvt, konsole (default on KDE), terminator
* The Graphical desktop can be switched on and off with systemctl utility or telinit:
  + ~> sudo systemctl stop gdm
  + (or sudo telinit 3)
  + ~> sudo systemctl start|restart gdm
  + (or sudo telinit 5)
* Basic Command line utilities:
  + cat : (used to type out a file or combine files)
  + tac : (used to lokk at a file backwards, starting with last line)
  + head : (used to show the first few lines of a file, default -n 10)
  + tail : (used to show the last few lines of a file, changed be flagged with -n 15)
  + less : (same as cat, but with scrolling, and search options with / or ?)
  + man : (used to view documentation)
* Most input lines entered at the shell prompt has 3 basic elements:
  + Command
  + Options
  + Arguments
* Sudo (stands for Superuser do) allows users to run programs using the security priviledges of root, or another user
* Setting up and running sudo:
  + : ~> su (Enter)
  + ~> su Password
  + #
  + ~> echo "myusername ALL=(ALL) ALL" > /etc/sudoers.d/myusername
  + (Create config file in /etc/sudoers.d)
  + ~> chmod 440 /etc/sudoers.d/myusername
  + (Change permissions for smooth operations)
* Loggin In and Out
  + An available text terminal will prompt for a username (with string login:) and password
  + Nothing is displayed on the terminal when typing in the password to prevent others from seeing it
  + You can also connect and log into remote systems by using SSH (Secure Shell)
  + To do so, type ~> ssh [example@remote-server.com](mailto:example@remote-server.com)
  + SSH connects to the remote machine, giving a command line terminal window, verifying identity using either password or crytographic key
* Shutdown
  + : ~> sudo shutdown -h 10:00 "Shutting down for scheduled maintance"
  + ~> tldr shutdown
  + (for more commands)
* Locating Applications
  + which diff ~> /usr/bin/diff : ('which' locates exact source, 'whereis' locates source and man files, broader range of sys dir)
  + whereis diff ~> diff: /usr/bin/diff /usr/share/man/man1/diff.1.gz /usr/share/man/man1p/diff.1p.gz
* Accessing directories
  + ~> pwd
  + (Displays present working directory)
  + ~> cd ~
  + (Changes to home directory, same as only : ~> cd)
  + ~> cd ..
  + (Change to parent directory ..)
  + ~> cd -
  + (Change to previous directory (- (minus))
  + ~> pushd dir1
  + (For remembering more than one directory, pushd pushes the pwd onto a stack)
  + ~> popd dir1
  + ("popd" can then used to retrieve the directory pushd on the stack) The list of dirs : ~> dirs
* Paths
  + Absolute pathname : (begin with root dir and follows the tree, branch by branch until it reaches desired dir or file) Always start with: /
  + Relative pathname : (starts from present working directory) Never starts with: /
    - ie : ~> cd /usr/bin : (Absolute path)
    - ie : ~> cd ../../usr/bin : (Relative path)
* Exploring Filesytem
  + ~> cd /
  + (Changes your current dir to the root (/) dir)
  + ~> ls
  + (list contents of pwd)
  + ~> ls -a
  + (list all files, including hidden files and directories)
  + ~> tree
  + Displays a tree view of the filesystem
* Symbolic Links
  + Symbols links (hard and soft links) are extremely useful in Linux, for creating shortcuts and saving diskspace
    - ln file1 file2 : (Creates a hard link)
    - ln -s file1 file3 : (Creates a soft link)
* Touch
  + Touch is often used to access, change and "modify times" of files. By default, it creates a file's timestamp to match current time.
  + : ~> touch <filename>
  + : ~> touch -t 202012312242 myfile (sets my file datestamp to 12/31/2020, 22:42
* Make and remove directory
  + ~> mkdir exampledir
  + (Creates exampledir with pwd as parent dir) Use -p flag to create recursively
  + ~> rmdir exampledir
  + (Will only work if "exampledir" is empty)
  + ~> rm -rf exampledir
  + (Remove recursively)
* Moving, renaming, removing file
  + ~> mv
  + (rename or move a file to another location, while possible changing its name at the same time)
  + ~> rm
  + (remove a file) -f flag for forcefully removing it, -i flag for interatively removing it
* Command Line Prompt
  + ~> echo $PS1
  + $ (Consider customzing PS1 prompt for adding functionality)
  + : ~> PS1 = "%n $(shrink\_path -f)"
* Standard File Streams
  + When commands are executed, there are 3 standard file streams always open for use:
    - stdin 0 keyboard
    - stdout 1 terminal
    - stderr 2 log file
* IO Redirection
  + Through the shell command, we can redirect the three standard file streams:
    - ~> do\_something < input-file
    - (Send (output of) the input file to (the input of) do\_something)
    - ~> do\_something > output-file
    - (Send (output of) do\_something to (the input of) output-file)
    - ~> do\_something 2> error-file
    - (Send (stderr of) do\_something to (the input of) error-file
    - ~> do\_something > all-output-files 2>&1
    - (Send anything written to a file descriptor 2(stderr) to the same place as file descriptor 1
    - ~> do\_something >& all-outpout-file
    - (Easier syntax of above)
* Pipes
  + The Linux philosophy is to have many simple and short commands cooperate togethe to produce quite complex results.
  + To do so, we use pipes. You can pipe the output of one command or program into another as its input.
  + : ~> command1 | command2 | command3
* Searching for files
  + Efficiently searching for files in Linux greatly enhaces productivity:
    - : ~> locate ...
    - : ~> find ...
  + The "locate" utility program takes advantage of "updatedb" which is a previously constructed database of files and directories.
    - ~> locate zip | grep bin
    - (grep is used to print only the lines with the specified strings, otherwise we would get a long list)
    - ~> updatedb
    - (The database can be updated at any time fro the command line as root user)
  + Wildcard and Matching File names
    - Simple answer
    - use regex
  + The "find" is an extremy useful utility program for a Linux sys admin. It recurses down the filesystem tree for dir(s) and locates files with matched conditions.
    - ~> find /usr -name gcc
    - (searching for files and directories named gcc)
    - ~> find /usr -type d -name gcc
    - (searching only for directories named gcc)
  + - : ~> find /usr -type f -name gcc : (searching only for files named gcc) Adv "find" options
    - ~> find -name "\*.swp" -exec rm {} ';'
    - (find and removes all files that end in ".swp"). Use -ok instead of -exec for interactive removal.
    - ~> find / -ctime 3
    - (find files changed in the last 3 days) , options include -atime, -mtime , number (n) exact | -n | +n
    - ~> find / -cmin -120
    - (find files changed in the last <120 mins) , options include -amin, -mmin , number (n) exact | -n | +n
    - ~> find / -size 0
    - (find files of size 0 bytes), can specify bytes(c), kilobytes(k), megabytes(M), gigabytes(G)
    - ~> find / -size +10M -exec cmd {} ';'
    - (find files greater than 10 MB in size, and run a command on those files)

# Chapter 8 Summary

* The main sources of Linux documentation are the:
  + man pages
    - ~> man -f
    - (same as : ~> whatis)
    - ~> man -k
    - (same as : ~> apropos)
    - ~> man -a
    - (display all pages in all chapters, one after the other), pipe with head of more succint output
  + GNU info
    - ~> info
    - (up arrows, q to quit, h for help, Enter to select menu item, n -> next node, p -> previous node, u -> up node)
  + help
    - ~> man --help
    - (same as : ~> man -h), short description and quick reference, faster than "man" or "info"
    - ~> yelp man:cat : (brings up the Graphical help systems)
  + online documentation sources (like Gentoo handbook, Ubuntu Documentation, etc)

# Chapter 9 Summary

* Processes are used to perform various task on the system
* Processes can be single-threaded or multi-threaded
* Processed can be of different types, such as interative or non-interactive
  + Process type: interactive | batch | daemons | threads | kernel threads
  + The "scheduler" is a critical kernel function that constantly shifts processes on and off the CPU, sharing time to relative priority
    - When a process is in a "running" state, it means it is either currently executing instructions on a CPU, or waiting for a time slice for execution
    - All processes in this state reside on what is called a run queue; on a computer with multiple core, there is a run queue on each
    - When processes go into a "sleep" state, they are said to be sitting on a wait queue (Can also go into zombie state)
* Every process has a unique identifier (PID) to enable the OS to keep track of it:
  + PID : (Unique Process ID number)
  + PPID : (Parent Process ID), Process (Parent) that started this process, if parent dies, the PPID will refer to adoptive parent, which is "kthreadd" with PPID=2
  + TID : (Thread ID number), same as PID for single-threaded process; for multi-threaded process, each thread shares the same PID, but has a unique TID
* Terminating process
  + ~> kill -SIGKILL <pid>
  + Some -SIGKILL and -9 are the same
  + ~> kill -9 <pid>
  + You can only kill your own processes, those belonging to another user are off limits, unless root
* The nice value, or niceness, can be used to set priority (-20 highest priority to +19 lowest priority)
  + ~> ps lf
  + (display process with priority)
  + ~> renice +5 <pid>
  + (increase the niceness lowers the priority)
  + ~> sudo renice -5 <pid>
  + (only root can increase the priority by decreasing niceness)
* "ps" provides information about the current running processes
  + ~> ps -ef
  + (Displays all the processes in the system in full detail)
  + ~> ps -eLf
  + ("" with one line information of every thread - one process can contain multiple threads)
  + ~> ps aux
  + (Display all process for all users), consider piping or grep for more succint outputs
  + ~> ps axo stat,pid,pcpu
  + ("axo" allows you to specify which attributes you want to view)
  + ~> pstree
  + (Displays the processes running in the form of a tree)
* "top" gives constant real-time updates about the overall system performance, as well as info about processes running on the system
* Load average indicates the amount of utilization the system is under at particular times
  + ~> w
  + 0.45 0.17 0.12 : last-minute-utilization 5-min-utilization 15-min-utilization
  + ~> top
  + (Real time utilization)
    - top 1st line: how long system has been up, how man user logged on, load average
    - top 2nd line: total number of processes, number of running, sleeping, stopped, and zombie processes
    - top 3rd line: cpu time divided between users (us) and kernel (sy).
    - Also %user-jobs niceness (ni), idle mode(id), jobs waiting(wa), hardware & software interrupts (hi, si). Steal time for VMs (st)
    - top 4th line: Physical Memory (RAM)
    - top 5th line: Swap Space
  + - top Output : PID, USER, PR, NI, VIRT, RES, SHR, S, %CPU, %MEM, TIME+, COMMAND Commands within top: - : ~> t : (Toggle summary info rows 2 & 3) - : ~> m : (Toggle memory info rows 4 & 5) - : ~> A : (Sort process list by top resource consumers) - : ~> r : (Renice (change priority) of process(es)) - : ~> k : (Kill a specific process) - : ~> f : (Enter top config screen) - : ~> o : (Interactively select new sort order in process list)
  + : ~> uptime
* Linux supports backgrounds and foreground processing for a job
  + ~> sleep 100 &
  + (Append '&' to another command to make it run in the background)
  + ~> fg
  + (Bring job to foreground)
  + ~> bg
  + (Push job to background) Use Ctrl-Z to suspend, and Ctrl-C to terminate foregrond jobs
  + ~> jobs -l
  + (Displays all jobs running in the background, -l flag adds PID)
* "at" executes any non-interactive command at a specified time
  + ~> at now + 2 days
  + (Specifies that the task needs to be completed 2 days from now)
  + at> cat file.txt
  + (Specifies task to be performed, note that PS1 changed)
  + at> <EOT>
  + (Ctrl-D to end task)
  + ~> atq
  + (Back to normal terminal, and use atq to view queued jobs)
* "cron" is used to schedule tasks that need to be performed at regualar intervals
  + Each line of a crontab file contain 5 + 1{command} fields

{command}

* - Min Hr Day Mth WeekDay {Command} : [0-59]Min [0-23]Hr [1-31]Day [1-12]Mth [0-6]WeekDay,(0=Sunday) Example: - create a simple task every day at 11 AM. - : ~> echo "0 11 \* \* \* /tmp/myjob.sh" > mycrontab : (Create a file "mycrontab" with the fields) - : ~> echo "#!/bin/bash" > /tmp/myjob.sh : (Put shebang into /tmp/myjob.sh) \* ~> echo "echo Hello I am running $0 at $(date) >> /tmp/myjob.sh : (Append to /tmp/myjob.sh) - : ~> chmod +x /tmp/myjob.sh : (Make it executable) - : ~> crontab mycrontab : (Put it into crontab) - : ~> crontab -l : (verify it was loaded), or : ~> cat /var/spool/cron/myusername - : ~> crontab -r : (Remove cronjob), if machine is NOT up at 11 AM, "anacron" will run the job another time.
* sleep
  + sleep NUMBER[suffix] : (s for seconds, m for minutes, h for hours, d for days)

# Chapter 10 Summary

* The filesystem starts at the root directory (/)
  + Linux supports several native filesystem types like:
    - ext3, ext4, squashfs, btrfs
  + There are also implementations of filesystems used on other os, such as:
    - Windows (ntfs, vfat)
    - SGI (xfs)
    - IBM (jfs)
    - MacOS (hfs, hfs+)
* The fileystem hierarchy standard (FHS) provides Linux developers and system admins a standard directory structure for the filesystem
  + In Linux (and Unix-like OS), "everything is a file" or treated as such
  + This means whether we are dealing with data files, docs or devices, we can interact with them throught the same kind of I/O operations
* Partitions help to segregate files according to usage, ownership, and type
* Filesystems can be mounted anywhere on the main filesystem tree at a mount point. Automatic filesystem mounting can be set up by editing /etc/fstab
  + ~> sudo mount /dev/sda5 /home
  + (mount a filesystem somewhere within the filesystem tree), basic arguments are "device node" and "mount point"
  + ~> sudo umount /home
  + (unmount the partition - command is umount, not unmount!)
  + ~> df -Th
  + (diplays information about mounted filesystems, including filesystem type, and usage statistics)
* NFS (Network File System) is useful for sharing files and data through the network systems
  + Many system admins mount remote users' home directories on a server in order to give them access to the same files and config files across multiple client systems.
  + On the server machine, NFS uses "daemons" (built-in networking and service processes in Linux) and other system servers are started by typing:
    - ~> sudo systemctl start nfs
    - (The file : ~> cd /etc/exports contains dirs and permission that a host is willing to share with other systems over NFS.)
    - ~> /projects \*.example.com(rw)
    - (This allows the dir /projects to be mounted using NFS with read & write (rw) permissions and shared with other hosts in the example.com domain)
    - ~> exportfs -av
    - (Notify Linux abt the directories that you are allowing to be remotely mounted using NFS) can also do : ~> sudo systemctrl restart nfs
    - ~> sudo systemctl enable nfs
    - (Ensure that the NFS service starts whenever the system is booted)
  + On the client machine, it is desired to have the remote filesystem automatically mounted upon system boot
    - : ~> servername:/projects /mnt/nfs/projects nfs defaults 0 0 : (an entry in the client's : ~> cd /etc/fstab needs to be modified for this default behaviour)
    - : ~> sudo mount servername:/projects /mnt/nfs/projects : (you can also mount the remote filesystem without a reboot or as a one-time mount directly by using "mount")
* Filesystem like /proc are called pseudo filesystems because they exist only in memory
  + ~> mount
  + (show all mounted filesystems)
  + ~> cat /etc/fstab
  + (show filesystems, excluding special filesystems required for normal operation, subset of "mount")
  + ~> cat /proc/mounts
  + (This is how the utility gets its information, same as "mount" more or less)
* /root is the home directory of the root user
  + ~> /bin
  + (contains executable binaries, essentail commands used to boot the system or in single user mode, and essential commands like "cat", "cp", "ls", "mv", "ps", "rm")
  + ~> /sbin
  + (essential binaries related to system admin, such as "fsck" and "ip") (Symbolically linked together:: /sbin -> /usr/sbin and:: /bin -> /usr/bin)
  + ~> /proc
  + (contains virtual files, that exist in memory, that permit viewing constantly changing kernel data) (Has runtime info like cpuinfo, interrupts, partitions, sys dir...)
  + ~> /dev
  + (contains device nodes, used by most hardware and software devices, except for network devices) (contains /sda1(first partition), /lp1(second printer) /random(rand nums))
  + ~> /etc
  + (home for system config files) (ie, /passwd /shadow /group for managing user account are all found there)
  + ~> /lib /lib64
  + (contain libraries for essential programs in /bin && /sbin) (Also are symbolically linked to /usr/lib) (Kernel modules are located in: ~> /lib/modules/<kernel-version-no>)
  + ~> /run /media /mnt
  + (used to temporarily mount filesystems, so called "loopback" filesystem which are files that pretend to be partitions)
  + ~> /opt
  + (optinal application software packages)
  + ~> /sys
  + (virtual pseudo-filesystem giving information about the system and the hardware, can be used to alter sys parameters and for debugging)
  + ~> /srv
  + (site-specifc data served up by the system, seldom used)
  + ~> /tmp
  + (temporary files, on some distros erased across a reboot, or may actually be ramdisk in memory)
  + ~> /usr
  + (multi-usr applications, utilities and data)
* /var may be put in its own filesystem so that growth can be contained and not fatally affect the system
  + ~> /var/log
  + (system log files)
  + ~> /var/lib
  + (packages and database files)
  + ~> /var/spool
  + (print queues)
  + ~> /var/temp
  + (temporary files)
* /boot contains the basic files needed to boot the system
  + ~> /vimlinuz
  + (Compressed linux kernel required for booting)
  + ~> /initrd
  + (initial ram filesystem, required for booting) - sometimes called ~> initramfs
  + ~> /config
  + (kernel configuration file, used for debugging and bookkeeping)
  + ~> /System.map
  + (Kernel symbol table, only used for debugging) [/grub is also found under /boot]
* /usr theoritically contains non-essential programs (not needed to initially boot system) and scripts and has some of these:
  + ~> /usr/include
  + (header file used to compile applications)
  + ~> /usr/lib
  + (libraries for programs in /usr/bin and /usr/sbin)
  + ~> /usr/lib64
  + (64-bit libraries "")
  + ~> /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 linux kernel)
  + ~> /usr/local
  + (data and program specific to the local machine, sub-dir include /bin /sbin /lib /share /include, etc)
  + ~> /usr/bin
  + (primary directory of executable commands on the system)
* "diff" is used to compare files and directories
  + ~> diff file1 file2
  + (flags include -c, -r (recursive for dirs), -i (ignorecase), -w(ignore tab/spces), -q (quiet, only report differences), use "diff3" to compared 3 files
  + ~> cmp file1 file2
  + (use "cmp" to compare binary files, not "diff")
* "patch" is very useful in Linux. Modifications to source code, and config files are distributed with patch files, contain the deltas or changes from old to new file vesions.
  + : ~> patch -p1 < patchfile
  + ~> patch originalfile patchfile
  + (make a patchfile using : ~> diff -Nur file1 file2 > patchfile) Then you can apply it
* File extensions in Linux do not necessariy mean that a file is of a certain type (This is very different from Windows!)
  + ~> file <filename>
  + (use the "file" utility to find information and the real nature of the files)
* "cp" is used to copy files on a local machine, while "rsync" is used to copy files from one machine to another, as well as synchronize contents
  + Rsync is much faster than "cp" because it checks if the file being copied already exists, and avoid unnecessary modifiations, and only copies part of the files that are changed.
  + Rsync is very efficient because only differences are transmitted over the network, one can sync the destination tree with the origin using -r
    - : ~> rsync -r project-X archive-machine:archives/project-X : (Always test with --dry-run option first to ensure you get the results you want)
    - ~> rsync sourcefile destinationfile
    - ~> rsync --progress -arvxH --delete sourcedir destdir
* "gzip", "bzip2", "xz", and "zip" are used to compress files
  + ~> gzip
  + Most frequently used Linux compression utility : ~> gzip \* : Compresseses all files in the pwd, each file is compressed and renamed a .gz extension : ~> gzip -r testX : Compresses all files in the "testX" dir, along recursively with all files in all dirs under "testX" : ~> gunzip foo : De-compresses foo found in the file foo.gz, same as : ~> gzip -d foo
  + ~> bzip2
  + Produces files significantly smaller than those produced by gzip : ~> bzip2 \* : Compresses all files in pwd, and replaces each one with a .bz2 extension : ~> bunzip2 \*.bz2 : Decompress all files with an extension .bz2 in pwd, same as : ~> bzip2 -d
  + ~> xz
  + The most space-efficient compression utility used in Linux : ~> xz \* : Compresses all files in pwd, and replaces each one with a .xz extension : ~> xz foo : Compresses "foo" into "foo.xz" with default compression level (-6) and remoes 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 -d \*.xz : Decompresses the files compressed using "xz"
  + ~> zip
  + Is often used to examine and decompress archives from other OS : ~> zip bkup \* : Compress all the files in pwd, and places them under "backup.zip" : ~> unzip bkup.zip : Extracts all files in "backup.zip" and places them in the current directory
* "tar" allows you to create or extract files from an archive file, often called a tarball.
* ~> tar xvf mydir.tar
* Extract all the files in "mydir.tar" into "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 comprss with xz
* ~> tar xvf mydir.tar.gz
* extract all the files in "mydir.tar.gz" into the "mydir" directory. Note you should NOT tell tar it is in gzip format!
* "dd" is used to make large exact copies, even of entire disk partitions, efficiently
* : ~> tldr dd

# Chapter 11 Summary

* The holy war of test editors
* Use Vim, period.

# Chapter 12 Summary

* Linux is a multi-user system
  + All Linux users are assigned a unique user id (uid), which is just an integer. Normal users start at uid of 1000 or greater.
  + Linux uses groups for organizing users. Control of groups is administered through: ~> /etc/group
  + Users also have one or more group IDS (gid), including a default one which is the same as the uid
  + The numbers are associated with names throught the files:
    - ~> /etc/passwd
    - ( ie, george:x:1002:1002:George Smith:/home/george:/bin/zsh )
    - ~> /etc/group
    - ( ie, george:x:1002 )
  + Adding and Removing Users
    - ~> sudo useradd username1
    - (: ~> sudo useradd -m -c "Elizer Jio" -s /bin/bash ejio ) : (We can now use: ~> adduser , more modern)
    - (
    - ~> sudo passwd ejio )
    - (
    - ~> grep ejio /etc/passwd /etc/group )
    - The home directory for newuser "ejio' inherits from
    - ~>ls -la /etc/skel )
    - ~> sudo userdel username1
    - (: ~> sudo userdel -r ejio )
  + Adding and Removing Groups
    - : ~> sudo /usr/sbin/groupadd anewgroup
  + - : ~> sudo /usr/sbin/groupdel anewgroup Adding a user to an already existing group is done with "usermod" - : ~> groups user1 - : ~> sudo /usr/sbin/usermod -a -G anewgroup user1 - : ~> groups user1
* Use the "who" command to find the currently logged-in users, use -a flag for all info
* To find current user ID, use the "whoami" command
  + ~> id
  + (gives information about the current user)
* The "root" account has full access to the system, it is never sensible to grant full root access to a user
  + External attacks often contain tricks to elevate to the root account
* "sudo" is used to assign root priviledges to a regular user on a temporary basis
  + : ~> sudo <command>
  + ~> /etc/sudoers && etc/sudoers.d
  + (contains "sudo" config files)
* The shell program (bash) uses multiple starup files to create the user environment. Global settings are found at : ~> cd /etc/profile
  + ~> ~/.bash\_profile || ~/.bashrc
  + ("~/.bashrc" is triggered and if not then the others are triggerd in order, stopped when first is found)
  + : ~> ~/.bash\_login
  + : ~> ~/.profile
* Advantages of startup files are:
  + Customizing user prompt (PS1)
  + set user's terminal type (bash or zsh)
  + set command-line shortcuts and aliases
    - : ~> alias +='pushd .'
    - : ~> alias -='popd'
    - : ~> alias ..='cd ..'
    - : ~> alias ...='cd ../..'
    - : ~> alias beep='echo -en "00?"'
    - : ~> alias md='mkdir -p'
    - : ~> alias o='less'
    - : ~> alias rd='rmdir'
    - : ~> alias python='python3'
    - : ~> alias git='git --no-pager'
    - : ~> alias grep='grep --color=auto'
    - : ~> alias egrep='egrep --color=auto'
    - : ~> alias fgrep='fgrep --color=auto'
    - : ~> alias you='if test "$EUID" = 0 ; then /sbin/yast2 online\_update ; else su ~ -c "/sbin/yast2 online\_update" ; fi'
  + set the default text editor, etc
* An environment variable is a character string that contains data used by one or more applications.
  + The built-in shell variables can be customized to suit your requirements.
  + : ~> echo $SHELL
  + ~> export VARIABLE=value
  + (same as: ~> VARIABLE=value; export VARIABLE )
  + : (Edit ~/.bashrc or ~/.zshrc and add export VARIABLE=value, then source ~/.bashrc)
  + : ~> SDIRS=s\_0\* KROOT=/lib/modules/$(uname -r)/build make modules\_install
  + : ( This feeds the values of "SDIRS" and "KROOT" environment variables to the command "make modules\_install" )
* The PATH Variable
  + PATH is an ordered list of difrectories (the path) which is scanned when a command is given to find the appriopriate 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 give time.
    - :path:path2 : (there is a null directory before the first colon (:) )
    - path1::path2 : (there is a null directory between path1 and path2)
  + To prefix a private bin to your path:
    - ~> export PATH=$HOME/bin:$PATH
    - ~> echo $PATH
  + Adding /tmp to Your PATH
    - ~> echo "echo HELLO, a phony ls program." > /tmp/ls
    - ~> chmod +x /tmp/ls
    - ~> OLDPATH=$PATH
    - ~> PATH=$PATH:/tmp : (append to PATH)
    - ~> ls /usr : (execute)
    - ~> PATH=/tmp:$PATH : (prepend to PATH) (Note: very dangerous, and is a trival way to insert a "Trojan Horse" program.)
    - ~> ls /usr : (execute)
    - ~> PATH=$OLDPATH
* The PS1 Variable and the Terminal
  + The PS1 is the primary prompt variable which controls what your command line prompt looks like.
  + The following characters can be included in PS1:
    - u -username
    - h -hostname
    - w -current working directory
    - ! -history number of this command
    - d -Date
  + They must be surrounded by single quotes when used:
    - ~> echo $PS1
    - (mine is customized to be : ~> '%n $(shrink\_path -f)>'
    - : ~> export PS1='u@h:w$ '
* "history" recalls a list of previous commands, which can be edited and recycled
  + ~> cd ~/.bash\_history
  + (or stored in : ~> ~/.zsh\_history )
  + ~> set | grep HIST
  + (view details)
  + ~> !!
  + (execute previous command, pronounced bang-bang)
* - : ~> CTRL-R : (Search previously used commands) or use Up/Down arrow keys Adv
  + ~> !
  + (Start 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")
* In Linux, various keyboard shortcuts are used at the terminal instead of long actual commands
  + ~> Ctrl-L
  + (Clear screen)
  + ~> Ctrl-D
  + (Exit current shell)
  + ~> Ctrl-Z
  + (Puts the current process into suspended background)
  + ~> Ctrl-C
  + (Kills the current process)
  + ~> Ctrl-H
  + (Works same as backspace)
  + ~> Ctrl-A
  + (Go to beginning of a line)
  + ~> Ctrl-E
  + (Go to the end of the line)
  + ~> Ctrl-W
  + (Deletes the word before the cursor)
  + ~> Ctrl-U
  + (Deletes whole line)
  + ~> Tab
  + (Auto-completes files, dirs, binaries)
* File Permissions
  + : ~> chmod permission filename
  + : rwx (u - user, g - group, o - other/world)
  + : 4 - (read permission)
  + : 2 - (write permission)
  + : 1 - (execute permission)
  + ~> Hence 7 - (read/write/execute)
  + 6 - (read/write)~
  + 5 - (read/execute)
* File Ownership
  + : ~> chown owner filename
* File Group Ownership
  + : ~> chgrp group filename

# Chapter 13 Summary

* The command line usually allows users to complete tasks more efficiently than the GUI
* "cat", short for concatenante is used to read, print, and combine files
  + ~> cat file1 file2
  + (Concatenate multiple files and display the output)
  + ~> cat file1 file2 > newfile
  + (Concatenate multiple files and save output in 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, "")
  + ~> cat > filename << EOF
  + (Read from stdout, send to file, when 'EOF' is typed, quit) (same as: ~> cat << EOF > filename)
* "echo" dislays a line of text either on the stdout or to place in a file
* "sed" stands for & is a stream editor used to filter and perform substitions on files and text data streams
  + ~> sed -e command <file>
  + (Specify editing commands at the terminal, operates on file, and outputs on stdout)
* - : ~> sed -f scriptfile <file> : (Specify scriptfile containing "sed" commands, "") Adv
  + ~> sed s/old/new file
  + (Substitute the fist string occurence in every line)
  + : ~> sed s:old:new:g file : (Substitute all string occurences in every line) (You can choose your delimiter, ie ':', '/', etc)
  + ~> sed 1,3s/old/new/g file
  + (Substitute all string occurences in range of lines)
  + ~> sed -i s/old/new/g file
  + (Save changes for string substitution in same file) (NOT reversiblebetter: ~> !! > file2)
* "awk" is an interpreted programming language, used as a data extraction and reporting tool
  + ~> 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) Basics - : ~> awk '{ print $0 }' /etc/passwd : (Print entire file) - : ~> awk -F: '{ print $1 }' /etc/passwd : (Print first field/col of every line, separated by a space) - : ~> awk -F: '{ print $1 $7 }' /etc/passwd : (Print first and seventh field of every line)
* "sort" is used to sort text files, and output streams in either ascending or descending order
  + ~> sort <filename>
  + (Sort the lines in the file, according to the chars at the beginning of the line)
  + ~> cat file1 file2 | sort
  + (Combine the two files, the sort the lines and then display output to terminal)
  + ~> sort -r <filename>
  + (sort the file in reverse order)
  + ~> sort -k 3 <filename>
  + (sort the file by the 3rd field on each line instead of the beginning)
* "uniq" eliminates duplicate entries in a text file
  + ~> sort file1 file2 | uniq > file3
  + ("uniq" requires duplicate entries to be consecutive, so we need to "sort" first and then pipe output into "uniq")
  + ~> sort -u file1 file2 > file3
  + (If "sort" is used with the -u flag (stands for uniq), then we can do the above in one step)
  + ~> uniq -c filename
  + (Count the number of duplicate entries)
* "paste" combines fields from different files, it can also be used to extract and combine lines from multiple sources
  + ~> paste file1 file2
  + (Paste contents from two files)
  + ~> paste -d, file1 file2
  + (Common delimiters are 'space', 'tab', '|', 'comma', etc)
* "join" combines lines from two files based on a common filed; it only works if the files share a common field
  + ~> join file1 file2
  + (Combine two files based on a common field)
* "split" breaks up a large file into equal-sized segments
  + It is generally used on relatively large files.
  + By default, "split" breaks up the file into 1000-line segments. The original file remains unchanged, and a set of new files with the name + prefix are created.
    - ~> split infile
    - (split a file into segments)
    - ~> split infile <prefix>
    - (split a file into segments using a different prefix)
* REGEX are text strings used for pattern matching
* "grep" searches for text files and data streams for patterns can be used with Regex
  + ~> 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 num 0 through 9)
  + ~> grep -C 3 [pattern] <filename>
  + (Print context files (specified num of lines above or below the pattern, here it is 3) for matching the pattern.
* "tr" translates characters, copies stdin to stdout, and handles special characters
  + ~> tr [options] set1 [set2]
  + (The items in the square brackets are optional)
* - : ~> cat filename | tr a-z A-Z : (In filename, tr Translates all lower case characters to upper case) Basics - : ~> tr abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ : (Convert lower case to upper case) : reverse to (Convert lower case to upper case) - : ~> tr '{}' '()' < inputfile > outputfile : (Translates braces into parenthesis) - : ~> echo "This is for testing" | tr '[:space:]' 't' : (Translate white-space into tabs) - : ~> echo "This is for testing" | tr -s '[:space:]' : (Squeeze repetition of charactes using -s) - : ~> echo "the geek stuff" | tr -d 't' : (Delete specified characters using the -d option) - : ~> echo "my username is 432234" | tr -cd '[:digit:]' : (Complement the sets using -c option) - : ~> tr -c [: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" saves a copy of stdout to a file while still displaying at the terminal
  + ~> ls -l | tee newfile
  + (Tees the 'ls -l' to out put stream and save 'ls -l' to newfile)
  + ~> cat newfile
  + (Will display the outpout of 'ls -l')
  + : (Tee can be effective to use with sudo and I/O redirection)
* "wc", short for word count, displays the number of lines, words and characters in a file or group of files
  + ~> wc
  + (lines-l words-w bytes-c)
* "cut" extracts columns from a file
  + ~> ls -l | cut -d" " -f3
  + (Display the third column delimited by a blank space)
  + ~> cut -f2 filename
  + (Cut second column of filename)
* "less" views files a page at a time and allows for scrolling
  + less does not have to load the whole page in memory first before displaying it, so it's way quicker
  + : ~> less somefile
  + : ~> cat somefile | less
* "head" displays the first few lines of a file or data stream
  + : ~> head -5 somefile
* "tail" displays the last few lines of a file or data stream
  + ~> tail -15 somefile
  + (show the last 15 lines)
  + ~> tail -f somefile.log
  + (monitor new output in a growing log file)
* "strings" extract printable character strings from binary files
  + It is useful in locating human-readable content embedded in binary files; for text files, one can just use "grep".
    - : ~> strings book1.xls | grep my\_string
* "z" command family is used to read and work with compressed files
  + ~> zcat compressed-file.txt.gz
  + (To view compressed gzip file) (use : ~> bzcat (for bzip2) : ~> xzcat (for xz))
  + ~> zless somefile.gz
  + (To page through compressed gzip file) (:~> bzless (for bzip2) :~> xzless (for xz))
  + ~> zgrep -i less somefile.gz
  + (To search into gzip compressed file)
  + ~> zdiff file1.txt.gz file2.txt.gz
  + (To compare two compressed files)

# Chapter 14 Summary

* Explain basic networking concepts, including types of networks and addressing issues
  + The IP (Internet Protocol) address is a unique logical network address that is assigned to a device on a network.
    - The "internet" is the largest network in the world, and can be called "the network of networks"
  + IPv4 uses 32-bits for addresses and IPv6 uses 128-bits for addresses
    - One reason IPv4 has not disappeared completely is that it uses 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.
      * A 32-bit IPv4 address is divided into four 8-bit sections called octets(another word for "byte").
      * -ie, IPv4 address 172. 16. 31. 46 -ie, Bit format 10101100.00010000.00011111.00101110
  + Every IP address contains both a network and a host address field
    - The Net ID is used to identify the network
    - The Host ID is used to identify a host in the network
  + There are 5 classes of networks addresses available: A, B, C, D & E
    - Decoding IPv4 addresses \*
    - Octet1 Octet2 Octet3 Octet4
    - NetID HostID HostID HostID : A : 126 Networks | 1.0.0.0 to 127.255.255.255 hosts (16.7 million unique hosts)
    - NetID NetID HostID HostID : B : 16,384 N | 128.0.0.0 to 191.255.255.255 hosts (65,536 unique hosts)
    - NetID NetID NetID HostID : C : 2.1 million N| 192.0.0.0 to 223.255.255.255 hosts (256 unique hosts)
    - multicast address : D :
    - reserved for future use : E :
  + IP Address allocation
    - Can be allocated 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
    - DHCP (Dynamic Host Config Protocol) is used to assign IP addresses
  + DNS (Domain Name System) is used for converting Internet domain and host names to IP addresses
    - ~> hostname
    - (view hostname - the special hostname with IP 127.0.0.1 describes the machine you are on)
    - ~> host example.com
    - (lookup hostname using DNS)
    - ~> nslookup example.com
    - (lookup nameservers interactively)
    - ~> dig example.com
    - (lookup domain name info from nameserver)
* Configure network interfaces and use basic networking utilites, such as "ifconfig", "ip", "ping", "route", and "traceroute"
  + ~> nmtui
  + (graphical version of Network Manager)
  + ~> nmcli
  + (command line interface for Network Manager)
  + "ip" utility is used to do several network tasks (it replaced "ifconfig" and "route")
    - ~> ip addr show
    - (to view IP address)
    - ~> ip route show
    - (to view routing information)
  + "ping" is used to check if the remote host is alive and responding)
    - "ping" is used to check whether or not a machine attached to the network can receive and send data
    - "ping" is frequently used for network testing and management, however its usage can increase network load unacceptably
    - ~> ping -c 10 linuxfoundation.org
    - (Use -c flag to limit number of packets ping sends before it quits, or use Ctrl-C)
  + "route" is used to manage IP routing
    - A network is a collection of nodes
    - Data moves from source to destination by passing through a series of routers and potentially multiple networks
    - Servers maintain routing tables containing the addresses of each node in the network
    - The IP routing protocols enables routers to build up a forwarding table that correlates final destination tables with next hop addresses.
      * ~> route -n
      * ~> ip route : (show current routing table)
      * ~> route add -net address
      * ~> ip route add : (add a static route)
      * ~> route del -net address
      * ~> ip route del : (delete a static route)
    - "traceroute" is used to inspect the route the data packet takdes to reach the destination host
      * : ~> traceroute <address>
  + You can monitor and debug network problems using networking tools
    - "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" test DNS workings. A good replacement for host and nslookup
* Use graphical and non-graphical browsers, such as Lynx, w3m, Firefox, Chrome and Epiphany
  + Firefox, Chrome, Chromium, Brave, and Epiphay are the main graphical browsers used in Linux
  + Lynx, Links, w3m are non-graphical / text-browsers used in Linux
    - : ~> w3m <url>
    - : ~> tldr w3m
* Transfer files to and from clients and servers using both graphical and text mode applications such as Filezilla, ftp, sftp, curl, and wget
  + "wget" is used to download webpages
    - it can handle large file downloads, passwd-required downloads, multiple-file and recursive downloads
      * : ~> wget <url>
  + "curl" is used to obtain information about URLs
    - "curl" allows you to save the contents of a web page to a file, as does "wget"
      * : ~> curl <url>
      * : ~> curl -o saved.html <http://www.mysite.com> : (Contents of main index file at the website will be saved in saved.html)
  + FTP (File Transfer Protocol) is used to transfer files over a network
    - ftp, sftp, ncftp, and yafc are command line FTP clients used in Linux
    - FTP clients enable you to transfer files with remote computer using the FTP protocol
    - All web browsers support FTP, all you have to do is to give a URL like
      * : ~> <ftp://ftp.example.com> : (the usual <http://> becomes <ftp://>)
    - FTP has fallen into disfavor on modern systems, as it is insecure
      * It was removed in favor of "rsync" and web browser "https" access
      * sftp is very secure, it uses the ssh protocol (It encrypts its data. However it does not work with "anonymous FTP")
  + SSH is used to run commands on remote systems
    - SSH is a cryptographic network protocol used for secure data communication
    - ~> ssh some\_system
    - (login to a remote system)
    - ~> ssh [someone@some\_system](mailto:someone@some_system)
    - (run as another user) same as: ~> ssh -l someone some\_system
    - ~> ssh some\_system my\_command
    - (run a command on a remote system via SSH)
    - SCP (Secure Copy) can be used to move files securely between two networked hosts using SSH protocol
    - : ~> scp <localfile> <[usr@remotesystem](mailto:usr@remotesystem)>:/home/user

# Chapter 15 Summary

* Scripts are a sequence of statements and commands stored in a file that can be executed by a shell. (most common is "bash")
  + As a script executes, one can check for a specific value or conditon or failure as the result
    - success: 0
    - failure: non-zero
  + Basic Syntax and Special Chars
    - #
    - (used to add comments, except when used as # or as #! when starting a script)
    - (used at the end of a line to indicate continuation on the next line)
    - ;
    - (used to interpret what follows as a new command to be executed next)
    - $
    - (indicates what follows is an environment variable)
    - >
    - (redirect output)
    - <
    - (redirect input)
    - >>
    - (Append output)
    - |
    - (used to pipe the result to the next command)
  + Multiple Commands on a Single Line
    - ~> make ; make install ; make clean
    - (the three commands will all execute, even if the ones preceding them fail)
    - ~> make && make install && make clean
    - (subsequent commands are aborted when an earlier one fails - when using && operator)
    - ~> make || make install || make clean
    - (here, you proceed until something succeeds, and then further steps are aborted)
  + Chaining commands is not the same as piping them. In chaining, each step exits before the next one starts. In piping, commands can operate simultaneously.
* Scripts behave differently based on the parameters (values) passed to them
  + Shell scripts execute sequences of commands and other types of statements. These commands can be:
    - Complied applications : (ie, rm, ls, df, vi, gzip, ... complied from lower level languages such as C)
    - Built-in "bash" commands : (ie, cd, pwd, echo, read, logout, printf, let, ulimit, ...) (: ~> help to get full list)
    - Other scripts : (shell scripts, or scripts from other interpreted languages, such as perl and Python)
  + Script Parameters
    - $0
    - (Script name)
    - $1
    - (First parameter)
    - $2, $3, etc
    - (Second, third parameter, etc)
    - $\*
    - (All parameters)
    - $#
    - (Number of arguments)
* Command substitution allows you to substitute the result of a command as a portion of another command
  + This can be done in two ways: by enclosing the inner command in: ~> $( ) or with backticks: ~> (`)
    - : ~> ls /lib/modules/$(uname -r)/ :the $( ) method allows command nesting, the other one is deprecated
* Environment variables are quantities either preassigned by the shell or defined and modified by the user
  + As covered earlier, some standard enviroment variables are HOME, PATH, and HOST.
  + : ~> echo $PATH
  + No prefix is required when setting or modifying the variable value
  + : ~> MYCOLOR=blue
  + A list of environment variables can be obtained using : ~> env :: ~> set :: ~> printenv
* To make environment variables visible to child processes, they need to be exported
  + by default, the varialbes created withing a cript are available only to the subsequent steps of that script.
  + To make them available to child processes (sub-shells), they need to be exported:
    - : ~> 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
  + Trying to export with no arguments will give a list of all currently exported environemnt varialbes
* Functions or routines are groups of commands that are used for execution
  + The proper syntax is:
  + : ~> function\_name () {
    - command...
    - }
* Output redirection is the process of writing the output to a file
* Input redirection is the process of reading the input from a file
* "if" statement is used to select an action based on a condition
  + A general definition is:
  + : ~> if condition ; then
    - statements
    - elif othercondition ; then
    - statements
    - else
    - statements
    - fi
  + A more compact form of the "if" statement is:
  + : ~> if TEST-COMMANDS; then CONSEQUENT-COMMANDS; fi
  + Testing for files
    - Bash provides a set of file conditionals, that can be used with the "if" statement to test
      * File or directory existence
      * Read or write permissions
      * Executable permission
    - Docs
      * ~> man 1 test
      * (View all of them here) [IMPORTANT for fundamentals of bash scripting!]
      * Basics
        + -e file
        + (Checks if file exists)
        + -d file
        + (checks if file is a directory)
        + -f file
        + (checks if file is a regular file ; ie, NOT a symbolic link, node, dir, etc)
        + -s file
        + (checks if file is of non-zero size)
        + -g file
        + (checks if file has "sgid" set)
        + -u file
        + (checks if file has "suid" set)
        + -r file
        + (checks if file is readable)
        + -w file
        + (checks if file is writable)
        + -x file
        + (checks if file is executable)
* Arithmetic expressions consists of numbers and arithmetic operators, such as +,-,\*,/
  + Numerical Test
    - Basics
      * -eq
      * (Equal to)
      * -ne
      * (Not equal to)
      * -gt
      * (greater than)
      * -lt
      * (less than)
      * -ge
      * (greater or equal to)
      * -le
      * (less or equal to)
  + "expr" utility
    - "expr" is a standard but deprecated utility used to evaluate numerical expressions
    - : ~> expr 8 + 8
    - : ~> echo $(expr 8+8)

# Chapter 16 Summary

* This is an extension of Chapter 15
* Strings can be manipulated to perform actions such as comparison, sorting, and finding length
  + Basic String Operators:
    - [[ string1 > string2 ]]
    - (compares the sorting order of string1 and string2)
    - [[ string1 == string2 ]]
    - (compares the characters of string1 and string2)
    - myLen1=${#string1}
    - (saves the length of "string1" in the variable "myLen1")
  + Parts of a String
    - : ~> $(string:0:n} : ('0' is the offset of the string (where to begin), 'n' is the num of chars needed)
    - ~> $(string#\*.}
    - (Extract all the characters in the string after a dot '.')
* Boolean expressions can be used when working with multiple data types, like strings, num or even files
* Output of a Boolean is either true or false
* Boolean Opearotors include && (AND), || (OR), and !(NOT) operators
* "case" statements can be used in senarios where the value of a variable can lead to different execution paths
  + Basic structure of "case":
  + : ~> case expr in
    - pattern1) execute commands;;
    - pattern2) execute commands;;
    - pattern3) execute commands;;
      * ) execute default\_command or nothing ;;
    - esac
* Looping Constructs
  + 3 most often types of loops
    - for loop
      * Basic Syntax
      * : ~> for var in list
      * do
      * execute one iteration for each item in list until end of list
      * finished
      * done
    - while loop
      * Basic Syntax
      * : ~> while condition is true
      * do
      * commands
      * done
    - until
      * Basic Syntax
      * : ~> until condition is false
      * do
      * commands
      * done
* Script debugging methods help troubleshoot and resolve erros
  + Script Debug Mode
    - ~> bash -x ./script\_file
    - (Run a bash script in debug mode) (IMPORTANT)
    - Another way is to bracket parts of the script with "set -x" and "set +x"
* The standard and error output from a script or shell commands can easily be redirected into the same file or separate files to aid in debugging, and saving results
  + : stdin 0
  + : stdout 1
  + : stderr 2
  + Using Redirection, we can redirect the 'error output' to an : ~> error.log file and then use "cat" to display contents to aid in debugging
    - : ~> ./some\_script 2> error.log
* Creating Temporary Files and Directories
  + Linux allows you to create temporary files and directories, which store data for a short duration, both saving space and increasing security
  + The best practice is to create random and unpredictable filenames for temporary storage
  + "mktemp" utility allows us to do the above
    - ~> TEMP=$(mktemp /tmp/tempfile.XXXXXXXX)
    - (To create a temporary file)
  + - : ~> TEMP=$(mktemp -d /tmp/tempfile.XXXXXXXX) : (To create a temporary directory) Important
    - Sloppiness in creation of temporary files can lead to real damage, either by accident or if there is a malicious actor
      * For example, if someone were to create a symbolic link from a known tmp file used by root to "/etc/passwd" like this:
        + : ~> ln -s /etc/passwd /tmp/tempfile
      * There could be a big problem if the script run by root has a line like this:
        + ~> echo $VAR > /tmp/tempfile
        + (The passwd file wlll be overwritten by temporary file contents)
      * To prevent such a situation, make sure you randomize your temporary files by replacing the two above lines with:
        + : ~> TEMP=$(mktemp /tmp/tempfile.XXXXXXXX)
        + : ~> echo $VAR > $TEMP
* /dev/null
  + Certain commands (like "find") produce large amounts of outpout, which can overwhelm the console
  + To avoid this, we can redirect the large outpout to a special file (a device node) called "/dev/null"
  + This pseudofile is also called the bit bucket or black hole
  + All data written in it is discarded and write operations never return a failure condiiton
    - ~> ls -lR /tmp > /dev/null
    - (Ignore entire output stream, errors will stil display)
    - ~> ls -lR /tmp >& /dev/null
    - (Both "stdout" and "stderr" will be dumpted into the black hole)
* Random Numbers and Data
  + Linus provides several different ways of generating random numbers, which are widely used
    - ~> $RANDOM
    - (Generates a random number between 0 - 99999)
  + The Kernel generates random numbers by using an entropy pool
    - Two device nodes namely "/dev/random" && "/dev/urandom" draw on the entropy pool to provide random numbers which are drawn from the estimated bits of noise in the entropy pool
      * ~> /dev/random
      * (used where very high quality randomness is required, such as one-time pad or key generation, but is relatively slow)
      * ~> /dev/urandom
      * (faster and good enough for most cryptographic purposes)
    - When the entropy pool is empty, "/dev/random" is blocked and does not generate any number until additonal env noise is gathered
    - Whereas "/dev/urandom" reuses the internal pool to produce more pseudo-random bits

# Chapter 17 Summary

* CUPS provides two command-line interfaces: the System V and BSD
  + ~> Ctrl-P
  + (Access printer options)
* The CUPS interface is available at <http://localhost:631>
  + Managing the CUPS daemon:
    - : ~> systemctl status cups
    - : ~> systemctl [enablestartrestart] cups
* "lp" (System V) and "lpr" (BSD) are used to submit a document to CUPS directly from the command line
  + ~> lp <filename>
  + (print to default printer)
  + ~> lp -d printer <filename>
  + (print to specific printer)
  + ~> program | lp
  + (pipe output of program to lp)
  + ~> lp -n number <filename>
  + (print multiple copies)
  + "lpoptions" can be used to set printer options and defaults
    - ~> lpoptions -d printer
    - (set the default printer) (use : ~> lpoptions help)
  + ~> lpq -a
  + (show the queue status)
  + ~> lpadmin
  + (configure printer queues)
* Managing Print Jobs
  + 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
    - ~> lpstat -p -d
    - (Get a list of available printers, along with their status)
    - ~> lpstat -a
    - (check the status of all connected printers, including job numbers)
    - ~> lprm job-id
    - (Cancel a print job, same as: ~> cancel job-id)
    - ~> lpmove job-id newprinter
    - (Move a print job to a new printer)
* PostScript effectively manages scaling of fonts and vector graphics to provide quality prints
* "enscript" is used to convert a text file to PostScript and other formats, like rich Text Format (RTF), and HTML
  + ~> 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 a default printer)
  + Converting between PDF and PostScript
  + ~> pdf2ps file.pdf
  + (convert file.pdf to file.ps, can also use: ~> pdftops file.pdf)
  + ~> ps2pdf file.ps
  + (convert file.ps to file.pdf, can also use: ~> pstopdf file.ps)
  + : ~> convert input.ps output.pdf : ~> convert input.pdf output.ps
* Portable Document Viewer (PDF) is the standard format to exchange documents while ensuring a certain level of consistency in the way the documents are viewed
  + Viewing PDF content can be done with "evince", "okular", "ghostView", and "xpdf"
  + Manipulating PDF
    - At times, you may want to merge, split or rotate PDF files.
    - Programs that enables these tasks are:
      * "qpdf" is widely avaialble on Linux distros and very full-featured
        + ~> qpdf --empty --pages 1.pdf 2.pdf -- 12.pd
        + (Merge 1.pdf and 2.pdf, save output in 12.pdf)
        + ~> qpdf --empty --pages 1.pdf 1-2 -- new.pdf
        + (Write only pages 1, and 2 of "1.pdf", save output to new.pdf)
        + : ~> qpdf --rotate=+90:1 1.pdf 1r.pdf : (Rotate 1 of "1.pdf" 90 degress clockwise and save to "1r.pdf")
        + : ~> qpdf --rotate=+90:1-z 1.pdf 1r-all.pdf : (Rotate all pages of "1.pdf" 90 degrees clockwise and save it "1r-all.pdf")
        + ~> qpdf --encrypt mypw mypw 128 -- public.pdf private.pdf
        + (Encrypt with 128 bits "public.pdf" using the passwd "mypw" with output as "private.pdf")
        + ~> qpdf --decrypt --password=mypw private.pdf decrypted.pdf
        + (Decrypt "private.pdf" as "decrypted.pdf". You will be prompted for password)
      * "pdftk" was once very popular, but is now starting to be obsolete as it depends on unmaintained package (libgcj)
        + ~> pdftk public.pdf output private.pdf user\_pw PROMPT
        + (Apply a paswd PROMPT to the PDF file)
      * "Ghostscript", invoked with "gs" is widely available and well maintained but can be a little complex
        + ~> gs -dBATCH -dNOPAUSE -q -sDEVICE=pdfwrite -sOutputFile=all.pdf file1.pdf file2.pdf file3.pdf
        + (combine 3 pdfs into one)
        + ~> gs -sDEVICE=pdfwrite -dNOPAUSE -dBATCH -dDOPDFMARKS=false -dFirstPage=10 -dLastPage=20 -sOutputFile=split.pdf file.pdf
        + (split pg 10-20 out of pdf)
* "pdfinfo" can extract information about PDF documents
  + ~> pdfinfo /usr/share/doc/readme.pdf
  + (collect the details of a document)
* "flpsed" can add data to a PostScript document
* "pdfmod" is a simple application with a Graphical Interface that you can use to modify PDF documents

# Chapter 18 Summary

* The root account has authority over the entire system
  + By default, Linux distinguishes between several account types in order to isolate processes and workloads
  + The four types of accounts are:
    - root (#)
    - system
    - normal : (can use SUID -Set User ID upon execution, same as Window's "run as" can grant temporary elevated permissions to the account)
    - network
  + "last" utility shows the last time each user logged into the system
    - : ~> last
    - : ~> last -F -a
* "sudo" & "su"
  + In order to perform any privileged operations such as system-wide changes, you need to use either "su" or "sudo"
  + "su" requires root passwd whereas "sudo" requires user's passwd(which may or may not also be root passwd in case of single user systems)
  + "su" does not require to retype the passwd again whereas "sudo" requires the passwd to be retyped (this can be avoided for a configured amount of time)
  + "su" has limited logging features whereas "sudo" has detailed logging features
* Calls to "sudo" trigger a lookup in "/etc/sudoers" or "/etc/sudoers.d", which first validates that the calling user is allowed to use sudo and that it is being used within the permitted scope
  + "sudo" keeps track of unsuccessful attempts at gaining root access
    - ~> /etc/sudoers
    - (config file)
    - ~> /etc/sudoers.d
    - (config file)
    - The basic structure of entries in these files is:
    - : ~> who where = (as\_whom) what
    - "visudo" is used to safely edit "sudoers" file
    - ~> visudo /etc/sudoers
    - (Edit)
    - ~> visudo -f /etc/sudoers.d/username
    - (Check for errors)
  + A message such as the following would appear in the system log file (/var/log/secure) when trying to execute "sudo bash" without successful authentication
    - : ~> authentication failure; logname ...... auth could not identify password for [op]....command=/bin/bash
* One of the most powerful features of "sudo" is its ability to log unsuccessful attempts at gaining root access.
  + By default, "sudo" commands and failures are logged in "/var/log/auth.log" under the Debian family and "/var/log/messages" in other distribution families
    - ~> /var/log/auth.log
    - (Debian family)
    - ~> /var/log/messages
    - (Other)
    - ~> /var/log/secure
    - (Other)
  + This is an important safeguard to allow for tracking and accountability of "sudo" usage
    - Using : ~> sudo whoami would generate in /var/log/auth.log:
    - : ~> Jun 6 00:54:21 nx-computer0 sudo: nx : TTY=pts/0 ; PWD=/etc/sudoers.d ; USER=root ; COMMAND=/usr/bin/whoami
* Process Isolation
  + Linux is considered more secure than other many other OS because processes are naturally isolated from each other
  + One process cannot access another person's resources, even when that person is running with the same user priviledges
  + Recent security mechanisms that limit risks further include:
    - cgroups : (control groups - allows system admins to group processes and associate finite resources to each group)
    - 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 processes can be isolated, but entire systems can run simultaneously at isolated and insulated guests (VMs) on one physical host
* Hardware Device Access
  + Linux limits user access to non-networking hardware devices in a way that is very similar to regular file access
  + Applications interact with the filesystem layer (which is independent of the actual device or hardware the file resides on)
  + This layer then opens a device special file (also known as device node) under the "/dev" dir that corresponds to the file being accessed
  + Each device special file has a standard owner, group and world permission fields - enforcing security naturally
* Using the user credentials, the system verifies the authenticity and identity
  + Originally encrypted passwds were stored in : ~> /etc/passwd which was readable by everyone. This made it rather easy to crack passwds
  + On modern systems, passwds are stored in an encrypted format under : ~> /etc/shadow Only those with root access can modify this file
* The SHA-512 algorithm is typically used to encode passwords. Deterministic one-way hash function.
  + : ~> sha512sum <filename>
* Pluggable Authentication Modules (PAM) can be configured to automatically verify that passwords created or modified using "passwd" are strong enough (that notion of strongness is configurable)
  + "passwd" utility can be combined with PAM to check if a pasword is sufficiently strong
    - : ~> passwd
  + "chage" utility can be used to remind users to change or update their passwds after a set period of time
    - ~> chage -l <username>
    - (list passwd information, not passwd itself, for user)
    - ~> sudo chage -N 10 <username>
    - (set passwd to expire in 10 days)
  + Password cracking programs, such as "John the Ripper" can be used to secure the passwd file and detect weak passwd entries
    - It is recommended to obtain written authorizations before installing such tools on any system that you do not own
  + Requiring Boot loader passwords
    - Securing the boot process with a secure password to prevent someone from bypassing the user authentication step
      * Note: while using a bootloader passwd alone will stop a user from editing the bootloader config file, it will NOT prevent a user from booting from an altenative boot media
      * Thus, using a bootloader passwd together with a BIOS password should be used for "full" protection.
        + ~> grub.cfg
        + (can never be edited directly)
        + ~> /etc/grub.d
        + (have to modify this config file) then run : ~> update-grub or: ~> grub2-mkconfig to save new config file
* Basic IT security policy should start with requirements on how to properly secure physical access to servers and workstations
  + Hardware Vulnerability
    - When a harware is physically acessible, security can be compromised by :
      * key logging : (record real time activity of a computer, including keys they press - recorded data is then transmitted to remote machines)
      * network sniffing : (cpaturing and viewing the network packet level data on your network)
      * booting with a live or rescue disk
      * remounting and modifying disk content
  + Security Guidelines
    - Phystical access to a system makes it possible for attackers to easily leverage several attack vectors, in a way that makes all OS level recommendations irrelevant
      * Lock down workstations and servers
      * Protect your network links such that it cannot be accessed by people you do not trust
      * Protect your keyboard where passwords are entered to ensure the keyboards cannot be tampered with
      * Ensure a passowrd protects the BIOS in such a way that the system cannot be booted with a live or rescue DVD or USB key
* Keeping systems updated is an important step in avoiding security attacks