**Linux**

**History and distributions:**1969 where Ken Thompson and Dennis Ritchie of Bell Laboratories developed the UNIX operating system. It was later rewritten in C to make it more portable and eventually became a widely used operating system.

Richard Stallman: GNU project and GPL

Linus Torval: in 1991, a young fellow named Linus Torvalds started developing what we now know today as the Linux kernel.

**The kernel is the most important piece in the operating system. It allows the hardware to talk to the software. It actually acts as a interface between user-level applitcaions and computer Hardware and pretty much is the OS in itself. It is the core of computer OS.**

A Linux system is divided into three main parts:

* Hardware - This includes all the hardware that your system runs on as well as memory, CPU, disks, etc.
* Linux Kernel - As we discussed above, the kernel is the core of the operating system. It manages the hardware and tells it how to interact with the system.
* User Space - This is where users like yourself will be directly interacting with the system.

Linux distributions:

**Debian - > Ubuntu - > Linux Mint**

Debian is an operating system composed entirely of free and open-source software. It’s widely known and has been in development for over 20 years. There are three branches that you can use, Stable, Testing and Unstable.

Stable is an overall good branch to be on. Testing and Unstable are rolling releases. This means that any incremental changes in those branches will eventually become Stable.

**Arch linux (pacman)**

**Gentoo (portage)**

**RHEL - > Fedora**

he Fedora Project is community driven containing open-source and free software. Red Hat Enterprise Linux branches off Fedora, so think of Fedora as an upstream RHEL operating system. Eventually RHEL will get updates from Fedora after thorough testing and quality assurance. Think of Fedora as an Ubuntu equivalent that uses a Red Hat backend instead of Debian.

OpenSUSE  
**Package managers for different Linux distributions:**

* Debian: Advance package tools (**apt**) <- DPM
* RHLE: Yellowdog Updater Modified (**yum**) for versions above 6, Dandified YUM (**dnf**) for versions <=6. <-RPM
* openSUSE: Zypp (**zypper**)

**SHELL:**

The shell is basically a program that takes your commands from the keyboard and sends them to the operating system to perform. If you’ve ever used a GUI, you’ve probably seen programs such as “Terminal” or “Console” these are just programs that launch a shell for you.

**SHELL formate:**username@hostname:current\_directory

pete@icebox:/home/pete $

**Linux terminal commands**

Commands:

1. **Echo command:**  
   The echo command just prints out the text arguments to the display.

$ echo Hello World

1. **whoami, date**
2. **Print working Directory (pwd) command:**

Everything in Linux is a file, as you journey deeper into Linux you’ll understand this, but for now just keep that in mind. Every file is organized in a hierarchical directory tree. The first directory in the filesystem is aptly named the root directory. The root directory has many folders and files which you can store more folders and files, etc. Here is an example of what the directory tree looks like:

/

├── bin/ # Essential command binaries

├── boot/ # Boot files (kernel, initrd)

├── dev/ # Device files

├── etc/ # System-wide configuration files

├── home/ # User home directories

├ └── user1/

├── lib/ # Shared libraries

├── mnt/ # Mounted file systems

├── opt/ # Optional packages

├── proc/ # Process and system information

├── root/ # Home directory for root user

├── sbin/ # System binaries (admin commands)

├── tmp/ # Temporary files

├── usr/ # User binaries and programs

├ ├── bin/

├ └── local/

└── var/ # Variable data (logs, spool, etc.)

1. **Change directory**

There are two different ways to specify a path, with absolute and relative paths.

* Absolute path: This is the path from the root directory. Every time your path starts with / it means you are starting from the root directory. For example, /home/pete/Desktop.
* Relative path: This is the path from where you are currently in filesystem.

It can get pretty tiring navigating with absolute and relative paths all the time, luckily there are some shortcuts to help you out.

* . (current directory). This is the directory you are currently in.
* .. (parent directory). Takes you to the directory above your current.
* ~ (home directory). This directory defaults to your “home directory”. Such as /home/pete.
* - (previous directory). This will take you to the previous directory you were just at.

$ cd /home/pete/Pictures

1. **List directories**

The ls command will list directories and files in the current directory by default, however you can specify which path you want to list the directories of. Also note that not all files in a directory will be visible. Filenames that start with . are hidden, you can view them however with the ls command and pass the -a flag to it (a for all). There is also one more useful ls flag, -l for long, this shows a detailed list of files in a long format.

**$ ls  
$ ls /home/pete**

**$ ls -a**

**$ ls -l**

1. **touch**

Touch allows you to the create new empty files.

**$ touch mysuperduperfile**

1. **file**

To find out what kind of file a file is, you can use the file command. It will show you a description of the file’s contents.

**$ file banana.jpg**

1. **cat**

A simple command to use is the cat command, short for concatenate, it not only displays file contents but it can combine multiple files and show you the output of them.

**$ cat dogfile birdfile**

1. **less**

If you are viewing text files larger than a simple output, less is more. (There is actually a command called more that does something similar, so this is ironic.) The text is displayed in a paged manner, so you can navigate through a text file page by page.

Go ahead and look at the contents of a file with less. Once you’re in the less command, you can actually use other keyboard commands to navigate in the file.

**$ less /home/pete/Documents/text1**

Use the following command to navigate through less:

* q - Used to quit out of less and go back to your shell.
* Page up, Page down, Up and Down - Navigate using the arrow keys and page keys.
* g - Moves to beginning of the text file.
* G - Moves to the end of the text file.
* /search - You can search for specific text inside the text document. Prefacing the words you want to search with /
* h - If you need a little help about how to use less while you’re in less, use help.

1. **History:History of commands**
2. **cp**

Much like copy and pasting files in other operating systems, the shell gives us an even simpler way of doing that. A useful command is to use the -r flag, this will recursively copy the files and directories within a directory. One thing to note, if you copy a file over to a directory that has the same filename, the file will be overwritten with whatever you are copying over. This is no bueno if you have a file that you don’t want to get accidentally overwritten. You can use the -i flag (interactive) to prompt you before overwriting a file.

**$ cp mycoolfile /home/pete/Documents/cooldocs**

**$ cp \*.jpg /home/pete/Pictures**

**$ cp -r Pumpkin/ /home/pete/Documents**

**$ cp -i mycoolfile /home/pete/Pictures**

1. **mv**

Used for moving files and also renaming them. Quite similar to the cp command in terms of flags and functionality. Like cp, if you mv a file or directory it will overwrite anything in the same directory. So you can use the -i flag to prompt you before overwriting anything. Let’s say you did want to mv a file to overwrite the previous one. You can also make a backup of that file and it will just rename the old version with a ~.

**$ mv oldfile newfile**

**$ mv oldfile newfile**

**$ mv file\_1 file\_2 /somedirectory**

**$ mv directory1 directory2**

**$ mv -i directory1 directory2**

**$ mv -b directory1 directory2**

1. **mkdir**

The mkdir command (Make Directory) is useful for that, it will create a directory if it doesn’t already exist. You can even make multiple directories at the same time. You can also create subdirectories at the same time with the -p (parent flag).

**$ mkdir books paintings**

**$ mkdir -p books/hemmingway/favorites**

1. **rm**

To remove files you can use the rm command. The rm (remove) command is used to delete files and directories. You can’t just rm a directory by default, you’ll need to add the -r flag (recursive) to remove all the files and any subdirectories it may have. -f or force option tells rm to remove all files, whether they are write protected or not, without prompting the user (as long as you have the appropriate permissions).

You can remove a directory with the rmdir command.

**$ rm file1**

**$ rm -f file1**

**$ rm -r directory**

**$ rmdir directory**

1. **find**

With find you’ll have to specify the directory you’ll be searching it, what you’re searching for, in this case we are trying to find a file by the name of puppies.jpg. You can specify what type of file you are trying to find. One cool thing to note is that find doesn’t stop at the directory you are searching, it will look inside any subdirectories that directory may have as well.

**$ find /home -name puppies.jpg**

**$ find /home -type d -name MyFolder**

1. **help**

Linux has some great built-in tools to help you how to use a command or check what flags are available for a command. One tool, help, is a built-in bash command that provides help for other bash commands (echo, logout, pwd, etc).

**$ help echo**

1. **man**

You can see the manuals for a command with the man command.

**$ man ls**

1. **whatis**

If you are ever feeling doubtful about what a command does, you can use the whatis command. The whatis command provides a brief description of command line programs.

**$ whatis cat**

1. **alias**

Sometimes typing commands can get really repetitive, or if you need to type a long command many times, it’s best to have an alias you can use for that.

**$ alias foobar='ls -la'**

**Keep in mind that this command won't save your alias after reboot, so you'll need to add a permanent alias in:**

**~/.bashrc**

**$ unalias foobar**

1. **exit**

**To quit the shell interface.**

**Text Commands:**

1. **Streams**

In Linux, **streams** refer to the abstraction of input and output (I/O) data flow in programs, especially in the context of Unix-based systems like Linux. Streams facilitate communication between programs and the external environment (such as files, user input, or another program). They are part of the Unix philosophy of "everything is a file."

Types of Streams in Linux

**1. Standard Input (stdin):**

**-** File descriptor: `0`

- Purpose: This is the default input stream for reading data into a program. By default, it usually reads from the keyboard unless redirected.

- Usage: In a shell, you can provide input to a program through stdin using redirection:

**$ ./program < input\_file.txt**

**2. Standard Output (stdout):**

- File descriptor: `1`

- Purpose: This is the default output stream where programs send output data. Typically, it outputs to the terminal unless redirected.

- Usage: To redirect the standard output to a file:

**$ ./program > output\_file.txt**

**$ pwd < path/to/some/file > sample.txt**

**$ ls < path/to/some/file > sample.txt**

**$ echo ‘Some text’ > sample.txt**

**$ cat sample.txt >> sample2.txt**

**>> is an upgrade to simple > operator as it allows append operation in case the file is present and filled.**

**3. Standard Error (stderr):**

- File descriptor: `2`

- Purpose: This stream is used to output error messages. It’s separate from stdout to allow error messages to be handled differently from regular output.

- Usage: To redirect stderr to a file:

**$ ./program 2> error\_log.txt**

**Additional Types of Streams (Pipes and Redirection)**

**1. Pipes:**

- Pipes (`|`) are used to connect the output of one command directly into the input of another. This allows chaining commands and processing data sequentially.

**- Example:**

**$ cat file.txt | grep "search\_term"**

**$ ls -la | tee sample.txt**

**$ echo $PATH | tr ‘:’ ‘\n’**

- Here, the output of `cat` is passed directly as input to `grep`.

- **The second one is the | tee operator as it will allow default operation as well as input for the next operation.**

- The third one is to get the path environment variables in a properly formatted view.

**2. File Descriptors:**

- Linux treats files, devices, and sockets as file descriptors. These descriptors are integers that the operating system uses to reference streams.

- Example of redirecting both stdout and stderr to a file:

**$ ./program > output\_file.txt 2>&1**

**3. Stream Redirection:**

- Redirection allows the standard streams to be diverted to/from files, devices, or other programs.

- Input redirection (`<`) reads from a file instead of stdin.

- Output redirection (`>`, `>>`) writes to a file instead of stdout, or appends to the file if `>>` is used.

Streams are a key concept in Linux, making I/O operations efficient, flexible, and uniform across different resources.

**The .bashrc File**

The .bashrc file is a shell script that is executed whenever a new terminal session is started in **interactive** mode (e.g., opening a new terminal window, or logging into a system). It is specific to the **Bash shell**, which is one of the most popular Unix/Linux command-line interpreters.

**Key Purposes of .bashrc:**

* **Customization**: It is used to customize the shell environment for the user. Users can add functions, aliases, and variables to suit their needs.
* **Environment Variables**: The file is often used to define or modify environment variables like PATH, HOME, or PS1 (which controls the command prompt appearance).
* **Aliases**: You can define shortcuts for commonly used commands. For example, aliasing ll to ls -alF for listing files with detailed information.
* **Shell Functions**: Custom functions that execute multiple commands can be defined for reuse.

**Location:**

* Typically, it is found in the user's home directory as a hidden file: ~/.bashrc.

**Example of .bashrc Content:**

# Aliases

alias ll='ls -alF'

alias grep='grep --color=auto'

# Custom PS1 (Prompt string)

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

# Export environment variables

export PATH=$PATH:/usr/local/myprogram/bin

# Custom functions

mkcd() { mkdir -p "$1"; cd "$1"; }

**How to Apply Changes:**

After editing .bashrc, you need to reload it for changes to take effect:

$ source ~/.bashrc

**When .bashrc is Executed:**

* It runs for **interactive non-login shells**, meaning when you open a new terminal window or tab, .bashrc is sourced.

**Relationship between .bashrc and .bash\_profile**

**The env Command**

The env command in Linux is used to either print the environment variables or execute a command in a modified environment.

Key Functions of env:

* Display Environment Variables: Running env without arguments displays all environment variables and their values.
* Modify the Environment for Commands: You can use env to run a command in a modified environment by temporarily setting or unsetting environment variables.
* Clear the Environment: It can also run a command with a completely clean environment by using the -i option.

**Syntax:**

**$ env [OPTION]... [VARIABLE=VALUE]... [COMMAND [ARG]...]**

**Common Use Cases:**

1. **Displaying all environment variables:**

**$ env**

1. **Running a command with a modified environment:**

**$ env PATH=/custom/path myprogram**

In this case, myprogram will be executed with the PATH variable temporarily set to /custom/path.

1. **Running a command with an empty environment:**

**$ env -i myprogram**

The -i option clears all existing environment variables, and myprogram runs with a clean environment.

1. **Set environment variables and run a command:** You can temporarily set environment variables for the duration of a command:

**$ env MY\_VAR=example command**

**Environment Variables:**

Environment variables are system-wide variables that define the behavior of the shell and other programs. Examples include:

* **$PATH: A list of directories the shell searches for executable files.**
* **$HOME: The user's home directory.**
* **$USER: The current username.**
* **$SHELL: The user's default shell.**

**Log in: ssh <user/acount\_name>@<ip\_addr>**

Log out**: exit**

Root directory (root of FS): **cd /**

Home directory (root/home/user\_name**): cd ~**

**For WSL on Windows: You can download wsl (make sure to add it to environment variables) and try installing available distributions by first knowing what distributions can be installed on your machine. This can be done by command in cmd:  
> wsl –list –online**

**> wsl install <distro name>**

Change account to root for WSL: You’d need to know your distributions command to access it via windows cmd. You can view it on Windows store for the installed distribution. For Debian: debian amnd for linux unbuntu: ubuntu.

Now just type this in windows command prompt: <distribution\_access\_command> config –default-user root

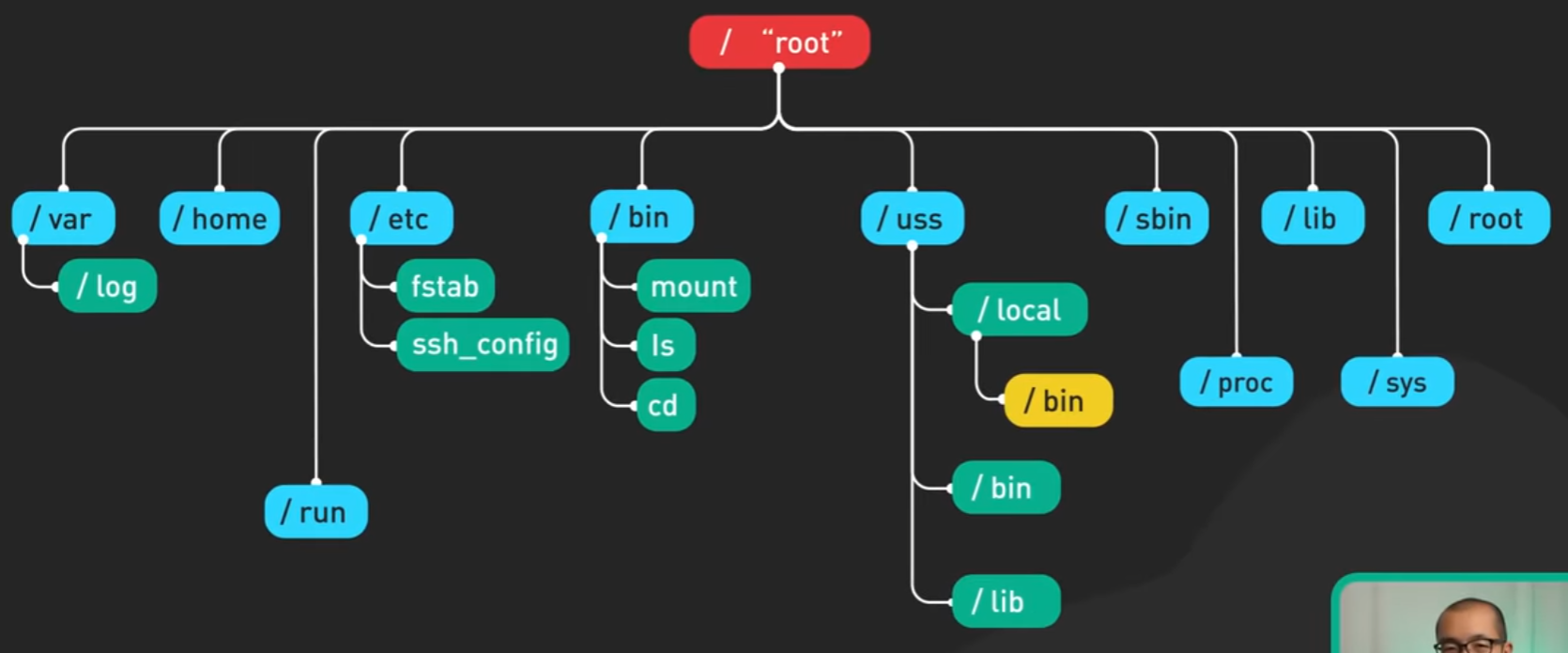
Or to change it back to normal user/account: <ditribution\_access\_command> config –default-user <account>

Confirm OS/Distribution on Linux: see contents of etc/os-release

All user/account details: see contents of etc/passwd

Command binary location:> **which <command>**

**Linux File system:**



**Major directories in root directory:**

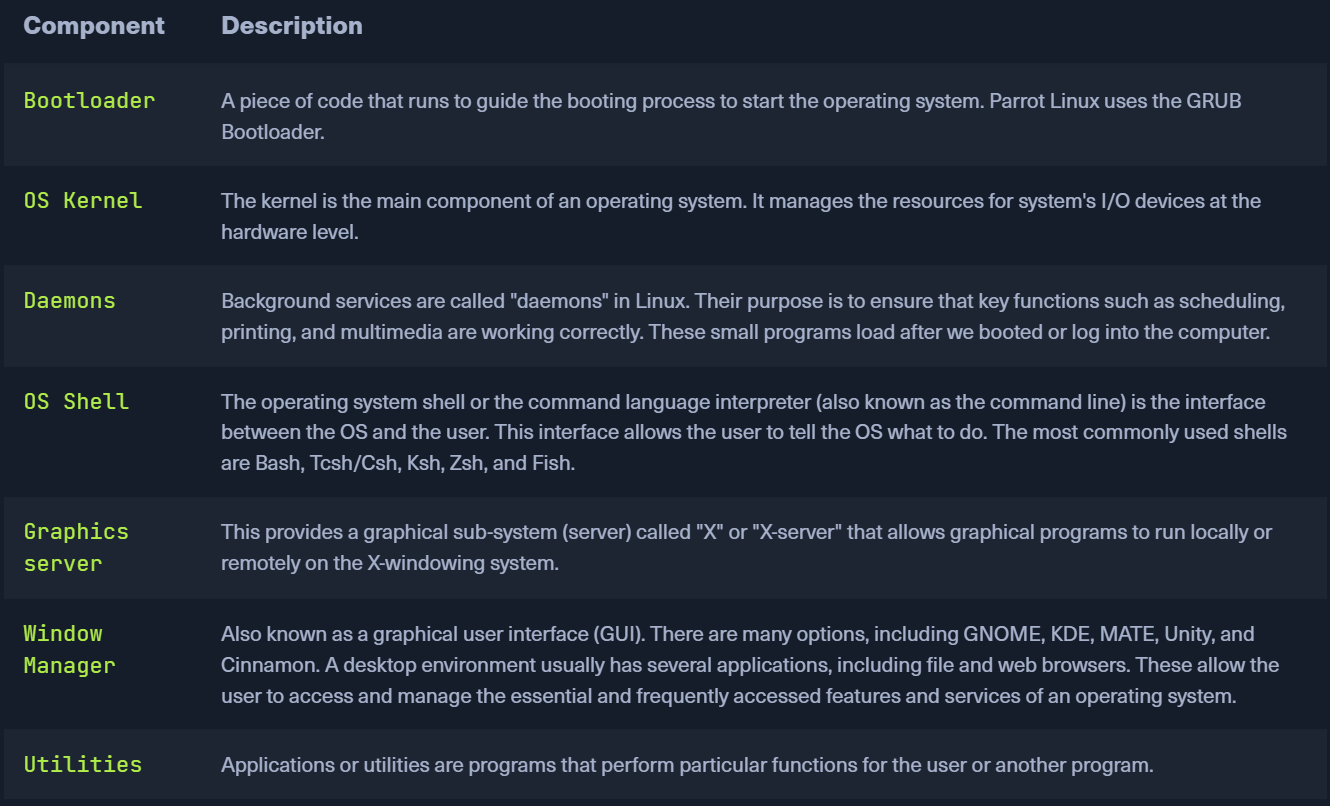
|  |  |
| --- | --- |
| / | The top-level directory is the root filesystem and contains all of the files required to boot the operating system before other filesystems are mounted as well as the files required to boot the other filesystems. After boot, all of the other filesystems are mounted at standard mount points as subdirectories of the root. |
| /bin | Contains essential command binaries. Holds system level binary files (system binaries). |
| /sbin | Contains executable binaries used by the super user (root if sudo group has no other member). |
| /boot | Consists of the static bootloader, kernel executable, and files required to boot the Linux OS. |
| /dev | Contains device files to facilitate access to every hardware device attached to the system. NOTE: /dev/null is directory which deletes automatically whatever is pushed into it. |
| /etc | Home to Local system-wide configuration files. Configuration files for installed applications may be saved here as well. (.conf files) |
| /home | Each user on the system has a subdirectory here for storage. |
| /lib | Shared library files that are required for system boot. |
| /media | Holds files system for External removable media devices such as USB drives are mounted here. |
| /mnt | Temporary mount point for regular filesystems. |
| /opt | Optional files such as third-party tools can be saved here. |
| /root | Like Special /home dir for super user |
| /temp | Directory holding temporary files for the running services |
| /var | Directory holding log files for applications and system srevices and tasks. |

**Note:** /bin contains essential system binaries that are needed for the system to boot and perform basic operations even before user gets mounted at boot time. Where as /usr/bin holds non-essential system binaries for user programs. And usr/local/bin holds installed program’s executable binaries. In summary, /bin contains essential binaries for system operation, /usr/bin holds non-essential user binaries, and /usr/local/bin is for locally installed software that is not part of the system's package management.

**Note: /media holds FS of the devices in /dev, attached to the file tree.**

**NOTE: Everytihng in Linux is a File**

**Components of Linux OS:**



Few in depth commands:

1. Usermod: modify users/groups:

**:> sudo usermod -aG <group(s) seperated by commas> <username>**

This adds user with <username> to a group/groups, Provided that current user is in sudoers file. (is admin or in sudo group)

**:>sudo groupadd <groupName>**

Create a group and add it in the list of groups

**:> groups <username>**

List all the groups where this user with <username> belongs

**:> usermod -l <new\_username> <old\_username>**

Chnages the username of the user.

**:>usermod -L <username>**

Lockt a user with <username> from root/other user(with root priviledges) account. This will not allow the user with <username> to login to their account as the admin/root has locked him/her. Inorder to unlock the user, root/user (with admin priviledges must) use -U flag.

**:> usermod -U <username>**

In the same way we can set an expiration date for a user account, from root/user with root priviledges. This can be done from root by:

**:> usermod <username> -e YYYY-MM-DD**

And we can check that using “chage” comand:

**:> chage -l <username**>

This will deliver us the account expiry details

1. Sudo: Grant Root privileges to current acount.

**Installation of this utility: sudo apt update**

**sudo apt install sudo**

**Switching to root: su - (switch user/login to another user, root here)**

Logout: exit

Check if <user> is from ‘sudo’ group**:> groups <user>** { list all groups:> groups }

Add the user, if he is not from ‘sudo’ group**:> usermod -aG <group> <user\_name>**

**:> usermod -aG sudo <user>**

NOTE: In order to get the new group membership working, reboot/logout-login to the user acccount.

How to check what sudo-command permissions does current user have:> sudo -l

Add sudo to previously (top on history stack) written non-priviledged command:> sudo !!

How to see list of admin-priviledged users/groups: View the content of the sudoers file (/etc/sudoer)

This file can be editted as wellm under root privileges. Root permissions can look like: root ALL=(ALL:ALL) ALL

Which translates to ‘For All the servers, All the users under All the groups have access to All the commands’

To edit this file (given that we are in priviledged user account):> sudo visudo

New\_user ALL=(ALL:ALL) /usr/bin/apt (**New user can only user sudo utility with apt command)**

How to add a user\_account:> sudo adduser <new\_user\_username>

Switch to (another user), say new\_account:> sudo su - <new\_usename>

Everytime you learn a priviledged command, you would be asked to type passord. To avoid this we can add something like this {say we want new\_user to use sudo only with apt command without ever asking for password}, the modify the sudoer file like this:

New\_user ALL=(ALL:ALL) NOPASSWD: /usr/bin/apt

1. Htop: monitor system/machine resources.

**Installation of this utility: sudo apt update**

**sudo apt install htop**

**:> htop**

In htop interface, you can sort the processes by the CPU usage/memory usage using**: Shift+P or Shift+M shortcuts. You can narow down the list of processes by user by presing ‘u’ and selecting username for which you want to monitor system resources. All the setup shortcuts are present in the htop interface.**

1. **ps** command: analyse about the processes running on the machine/server.

**:> ps**: Displays a snapshot of processes associated with the current terminal.

**:>** **ps aux**: Displays a detailed list of all processes running on the system, including those of other users. This is often used to get a comprehensive view of all processes.

**:>** ps -e: Shows information about every process on the system.

**:>** ps -ef: Similar to ps -e, providing a full listing of processes along with additional information.

The ps command, when used without specific options, typically shows a snapshot of processes associated with the terminal in which it is run.

Different terminal sessions may have different sets of running processes, and the ps command without specific options will display processes associated with the terminal from which it is executed. If you want to see a comprehensive list of all processes on the system, regardless of the terminal session, you can use the ps aux or ps -ef command as mentioned earlier.

1. **Data Streams in Linux**: In linux data streams have 3 by default channels. These 3 channels are fixed to 0, 1 and 2 for standard input stdin, standard output/stdout and standard error/stderr channel respectively. You can use the output/err output from one command to be utilised for the next command using ‘**>’ operator** between to commands. This is called comand chaining. Format is

<command1> <channel\_number> **>** <command2>………..

1. **Nano editor:** :>nano <file\_name>

It is a simple text editor to change and read file contents. Write out option is to save. ‘Where is ’ is search option. ‘Cut text’ is to cut entire line if not seleceted a specific part. ‘paste text’ is to paste/uncut option.

:> nano +<line\_number> <file\_name> //open at exactly given line number