Agenda

1. Dealing with Users & Groups
2. File Permissions and Ownership
3. Package Managers

Dealing with Users & Groups

User Account Types

1. Superuser
2. System User
3. Regular User

1. **Superuser/root (UID=0)**:
   This is the administrator of the system and has unlimited access to the resources of the system and unlimited ability to alter it
2. **System user (UID=1-999)**:
   These are accounts created by programs supporting the operation of the system. They exit so that each process can secure and isolate its resources and files from other processes
3. **Regular User (UID>=1000)**:
   These are the accounts used by the users of the system to finish their day to day work

User groups

1. Primary Groups
2. Supplementary Groups

1. **Primary group**:
   This is the group that a new user creates on the system belongs to.
   It is named the same name of the user and is the default group that a files created by the user belongs to.
   Only one primary group can exist for a given user account
2. **Supplementary groups**:
   Any other group a user belongs to other than the primary group

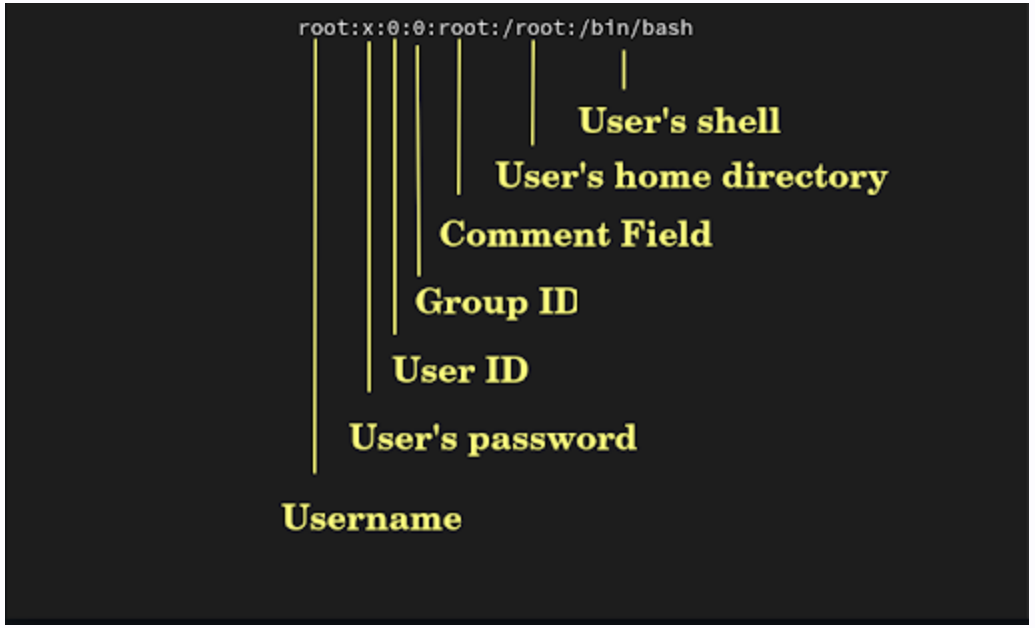
The `id` command

- to show information about the currently logged-in user use `id`
- to show information about the currently logged-in user use `id username`

```
[me@linuxbox ~]$ id
uid=500(me) gid=500(me) groups=500(me)`
```

The `/etc/passwd` file

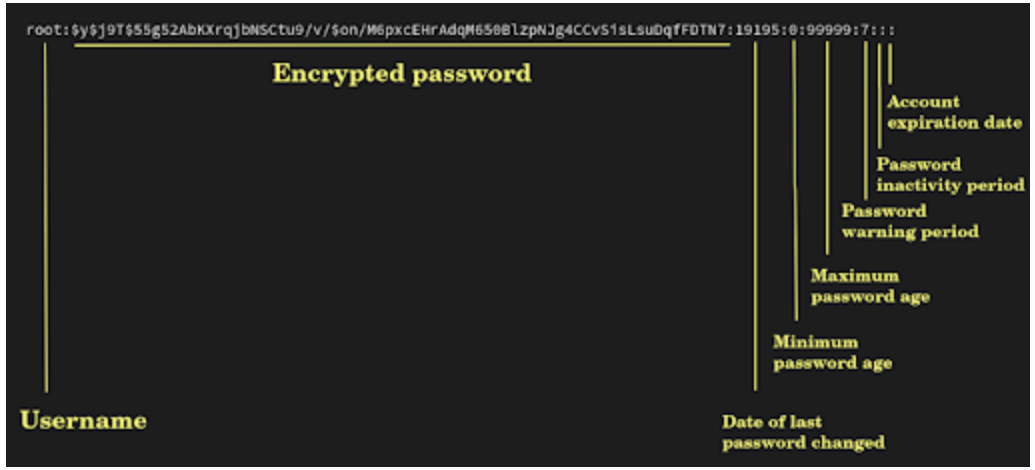
- This file stores information about the users on the system.
- Each line is giving information about a different user and is formatted as follows (the GID in this files is that of the primary group of the user)



> **Note**:
> if the user shell is set to "/sbin/nologin" that mean interactive logins are not allowed for that user

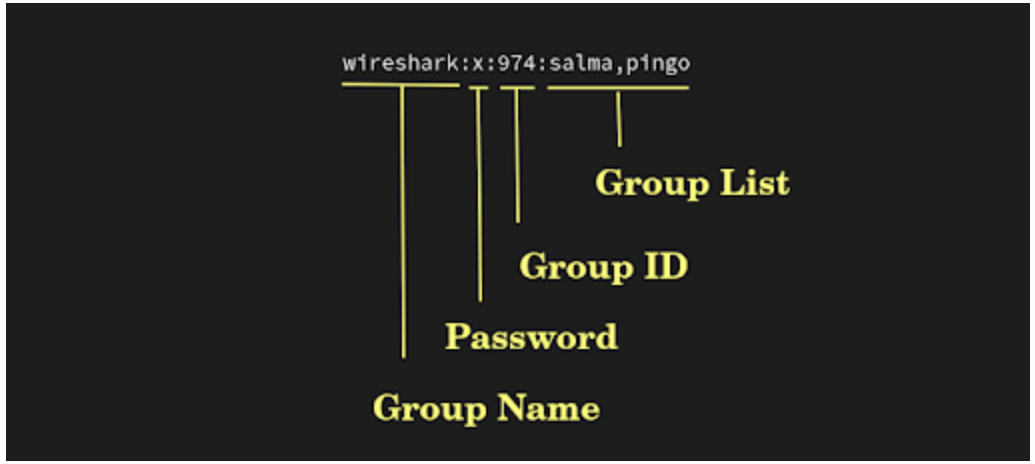
The `/etc/shadow` file

- This file stores information about user authentication. Each line is information about a different user and is formatted as follows



The `/etc/group` file

- This file stores information about user groups on the system. Each line is information about a different group and is formatted as follows



Gaining Superuser Access

The Sudo mechanism

- When we use the sudo command before a command that is reserved for the root user only, sudo searches the “sudoers” file (a file only accessed by the root user) and then checks if the root user gave you permission to run this command

- If it finds that you have the permission, you are then allowed to use this command with root privileges

- Else you are denied use of the command

Managing Users

Creating users

- Users are added using the `useradd` command
- syntax : `useradd [options] <username>`
- For unspecified options the command reads them from the `/etc/default/useradd` and the `/etc/login.defs` files

Editing and setting passwords

- Passwords are edited and set using the `passwd` command
- syntax : `passwd <username>`
- The command will then prompt you to change or set the password for the account specified
- Of course only the root user or a user with sudo privileges can change and set password for other accounts

Switching users

- We can switch between users using `su <username>`
- The password for the user we are switching to is needed unless you are the root user.

Deleting users

- Users are deleted using the `userdel` command
- **syntax** : `userdel [options] <username>`
- If the “-r” option is specified the command deletes the home directory of the user; else, it only removes the data of the user from the “/etc/passwd” files while keeping the files in the home directory

> Note: leaving the home directory can lead to security issues regarding file access

Managing groups

creating groups

- groups are created using the `groupadd` command
- syntax : `groupadd [options] <group name>`
- For unspecified option the command reads them from the “/etc/login.defs” file

Adding users to groups

- Existing users accounts are added to groups using the `usermod` command
- syntax : `usermod [options] <group name> <username>`
- So to add the user OSC to the group testGroup we will write `usermod -a -G testGroup OSC`
- The `-G` option tells the command that we will add the user to a supplementary group. The `-a` option puts the command in append mode; otherwise, the command will remove the user from all groups unspecified in the command (will overwrite instead of add)

Deleting groups

- Groups are deleted using the `groupdel` command
- syntax : `groupdel <group name>`
- You cannot delete the primary group of a user account

Hands on

1. Create a user called `member`
2. Set the password for testUser (make it something you can remember)
3. Create a group called `OSC`
   Get the UID, primary group GID, and supplementary groups of the new user
4. Add `member` to `OSC`
5. Finally delete `OSC`

File Permissions and Ownership

- Every file and directory on your UNIX/Linux system has the following 3 permissions defined for all the 3 owners discussed above.

    1. **Read** `r` :

        This permission gives you the authority to open and read a file.

        Read permission on a directory gives you the ability to list its content.

    2. **Write** `w` :

        The write permission gives you the authority to modify the contents of a file.

        The write permission on a directory gives you the authority to add, remove and rename files stored in the directory.

        Consider a scenario where you have to write permission on file but do not have write permission on the directory where the file is stored. You will be able to modify the file contents. But you will not be able to rename, move or remove the file from the directory.

3. **Execute `x`**:

   In Windows, an executable program usually has an extension ".exe" and which you can easily run. In Unix/Linux, you cannot run a program unless the execute permission is set. If the execute permission is not set, you might still be able to see/modify the program code(provided read & write permissions are set), but not run it.

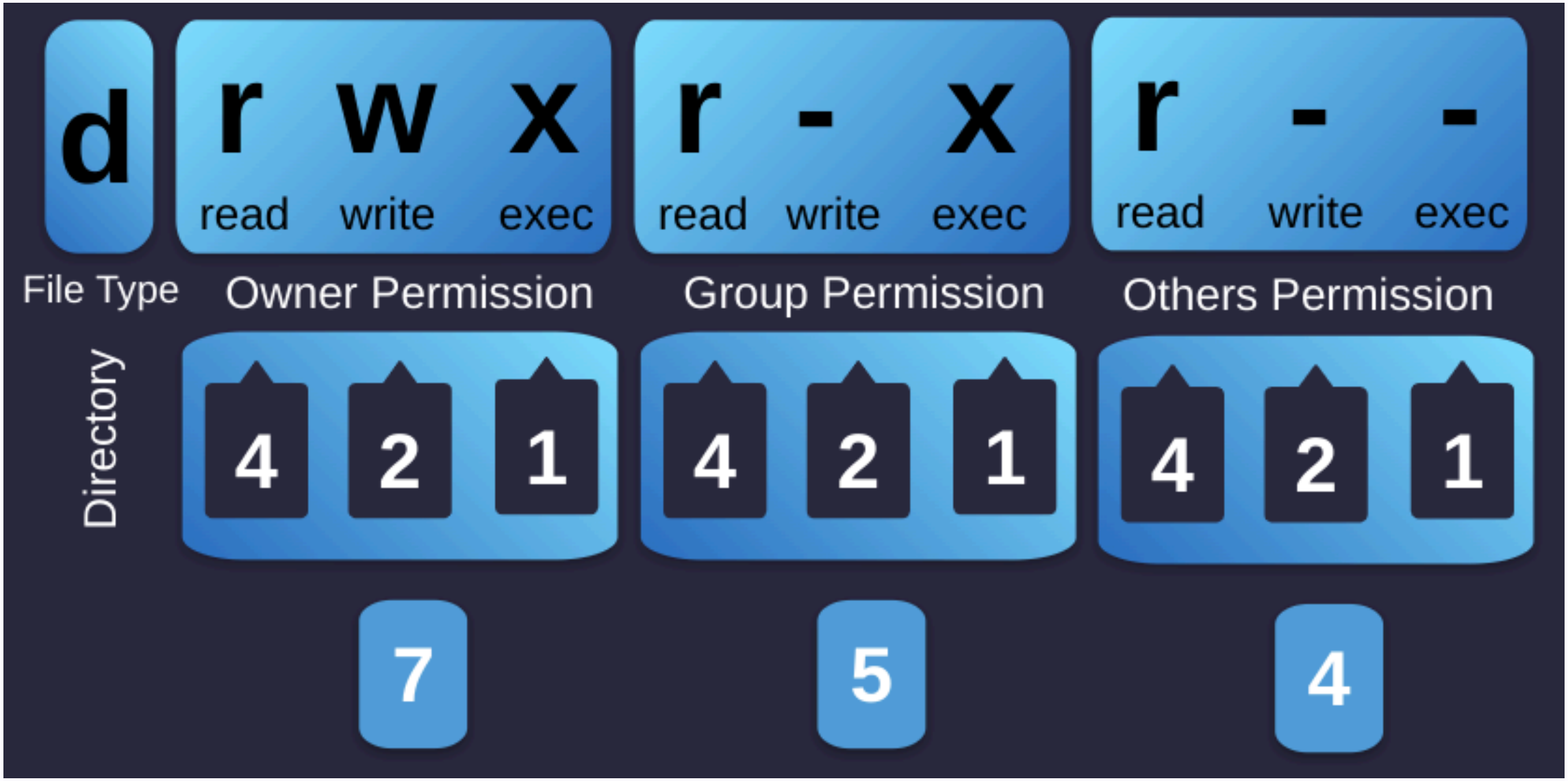
   The execute permission on a directory means that the contents of the directory can be accessed. (You can change into the directory, read information about its files, and access its files if the files' permissions allow it.)

   By default, any newly created files are not executable regardless of its file extension suffix.

> **Ownership : Every file and directory is assigned 3 types of owner**

1. **User**:
   A user is the owner of the file. By default, the person who created a file becomes its owner.
2. **Group**:
   A group can contain multiple users. All users belonging to a group will have the same Linux group permissions access to the file.
3. **Others**:
   Any other user who has access to a file. This person has neither created the file, nor he belongs to a group who could own the file. Practically, it means everybody else. Hence, when you set the permission for others, it is also referred as set permissions for the world.

Linux File Permissions


d
r w x
read write exec
r - x
read write exec
r - -
read write exec
File Type
Owner Permission
Group Permission
Others Permission
Directory
4 2 1
4 2 1
4 2 1
7
5
4

Changing Permissions

The chmod command

There are two way to use the chmod:

1. Absolute mode
2. Symbolic mode

Symbolic Mode

- In the symbolic mode, you can modify permissions of a specific owner.
- Syntax:

```
Chmod [ownerType] [operator] [new permission] [file name]
```

User	Denotations
u	user/owner
g	group
o	other
a	all

Operator	Description
+	Adds a permission to a file or directory.
–	Removes the permission.
=	Sets the permission and **overrides** the permissions set earlier.

Absolute Mode

In this mode, you can specify the permissions in the following way:

Number	Permission Type	Symbol
0	No Permission	`---`
1	Execute	`--x`
2	Write	`-w-`
3	Execute + Write	`-wx`
4	Read	`r--`
5	Read + Execute	`r-x`
6	Read +Write	`rw-`
7	Read + Write +Execute	`rwx`

- Example : `chmod 764 sample`
  - '764' absolute code says the following:
    - Owner can read, write and execute
    - Usergroup can read and write
    - World can only read
  - This is shown as `-rwxrw-r--`

Changing Ownership and Group

1. **chown**
2. **chgrp**

chown

- This command could be used to change both the owner and the group to which a file or directory belongs
  command : `chown [owner][:[group]] file...`
  - To change file or directory owner:
    `chown new user file/directory name`
  - To change file or directory owner and group:
    `chown new user:new group file name`

chgrp

- This command can only be used to change the group to which a file or directory belongs.
- To change group-owner only : `chgrp group_name filename`

groups

- You can use the command `groups` to find all the groups you are a member of.
  - To list all groups you are a member of : `groups`
  - To list all groups of a user : `groups username`

Hands On

try the following:

- Create a text file called `test.txt` in a folder on the desktop called `testFolder`
- Give the user who owns the file `execute` permission
- Stop users who are in the group owning the file from being able to `edit` it
- Stop users who are neither the owner nor in the owning group from being able to `list` the content of the `testFolder` directory

Package Managers

Packages

- Linux OS is made up of 2 parts:
  - The linux kernel
  - Software packages that work with the kernel to give us a complete OS
- A package is usually referred to as an application, it can be a GUI, command line tool, etc...
- Everything other than the kernel is a package in Linux.

Packages (cont..)

- Contents of packages include:
  - Binaries or the executable programs
  - metadata files containing the version, dependencies, signatures and other relevant information
  - documentation and manuals
  - configuration files

Packages dependencies

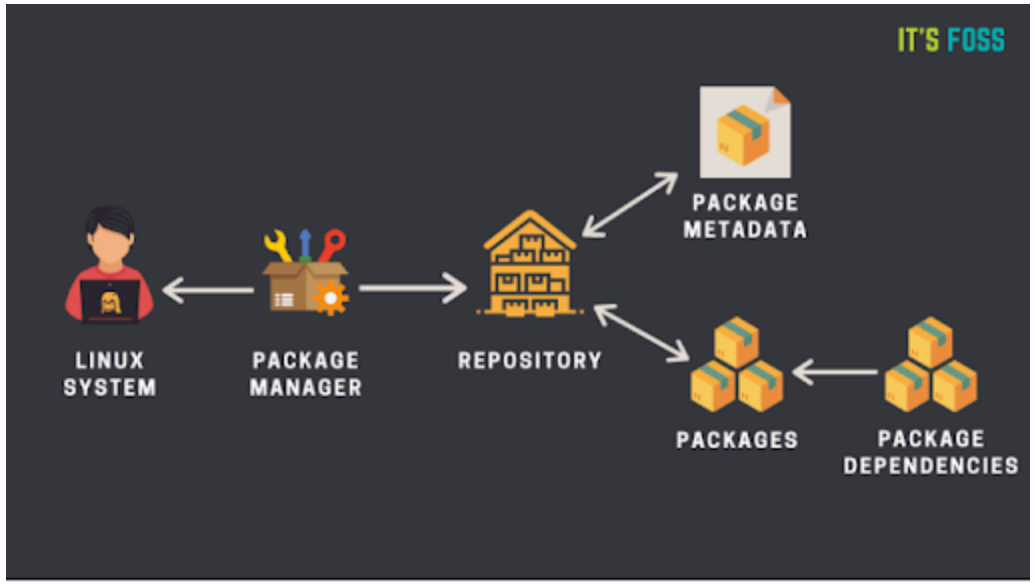
- Packages very rarely work by themselves, they are most often accompanied by dependencies to help them run. In Linux, these dependencies are often other packages or shared libraries. Shared libraries are libraries of code that other programs want to use and don't want to have to rewrite for themselves. If the dependencies aren't there the package will end up in a broken state and most of the time not even install.
- For example, if you need to run python-based apps, then you need to have the python present.
- Practically any software will not just have one dependency, it can have tens and sometimes even hundreds of dependency packages.

Packages repositories

Repositories (or repos for short) are basically a place where verified packages are stored for easy retrieval by the package manager. They can be online like the YUM repository or they can be on a local folder or a DVD where you have a special collection of software that you need

Package managers

A package manager is a tool that allows users to install, remove, upgrade, configure and manage software packages on an operating system. The package manager can be a graphical application like a software center or a command line tool like apt-get or pacman.



Package managers

- Almost all linux dstros have software repositories, which is a collection of packages.
- Repos also have metadata files which contain information about packages like the name, version, description, etc...
- Your system's package manager first interacts with the metadata. The package manager creates a local cache of metadata on your system, when you run the `update` command through any package manager, this local chache is updated.
- When you run the installation command of your package manager (for example apt install [Package Name]), the package manager refers to this cache. If it finds the package information in the cache, it uses the internet connection to connect to the appropriate repository and downloads the package first before installing on your system
- A package may have dependencies, package managers install these dependencies with the package and remove them if the package is removed.

Package managers

- Functions of package managers:
  - Automatically resolves dependencies.
  - Install, uninstall and update with ease.
  - Verifies the integrity of the package before installing it.
  - Verifies the architecture compatibility.
  - Keeps track of all the packages installed in the system so that the system administrator can easily get information about what packages are present, when was it installed, what version are they running in, etc.

Commands

- Most commands used with package managers are the same across different managers with the difference in the manager name, here are some examples for commands using `apt`:
  - Search for a package: `sudo apt search package_name`
  - Install a package: `sudo apt install package_name`
  - Remove a package: `sudo apt remove package_name`
  - Update package cache: `sudo apt update`
  - Upgrade all packages: `sudo apt upgrade`
  - Upgrade a specific package: `sudo apt upgrade package_name`
- It’s always recommended to run `update` before `upgrade`.
- Also it’s nest practice to upgrade your system before installing a new package.

Thanks!