1. **Executable files** :- Script executable files and Binary executable files . In windows OS , the newly created file has executable permission by default but not in unix based OS but we can change the permission of the files using chmod command .
   1. Script Executable files:- These are simple text files or any file with or without the extension which is treated as executable depending on the terminal.

* And the terminal considers the code written inside the file as commands and executes those commands.
* Bash terminal considers all text files as executable files but other terminals considers specific file extensions files as executable files like the powershell consider “.bat” file as script executable file.
* We have to pass the path of that file in our terminal in order to execute that file.
  1. Binary Executable Files:- These files have .exe extension in the window OS but they may or may not have extension in unix based OS.
* There are 2 ways to run these type of file . One is by double clicking that file and other by using terminal.
* We have to pass the name of that file in our terminal in order to execute that file and no need to provide the path of that file.
* We may or may not require to write .exe at the end of our command in order to execute these files.
* These files will be execute in all terminals.
* In order to execute binary executable files from our terminals ; we need to add the path in our system environment variable. The path should be the directory of that exe file where it is located.

In order to run to run the executable file from the terminal then instead of giving the relative or absolute path , we can add the path variable of these files in our environment variable.And then we can easily run our file.

**Question** :- if i want to execute the executable files. so both files script executable file and binary executable file in my project and i decided to run these file from the terminal then instead of giving the relative or absolute path , we can add the path variable of these files in our environment variable. And it is the usecase which i understand. My question is will the script executable files also work if the path is added . Or the only binary executable files work?

**Answer** :- Both **script executable files** and **binary executable files** will work if the directory containing them is added to the **PATH** environment variable. Here's how it works for each type:

**1) Binary Executable Files**

* These are compiled programs (e.g., .exe, .bin, .sh files) that can be directly executed by the operating system.

**How they work**:

* When you add the directory containing a binary executable to the PATH, the OS directly invokes the binary when you type its name in the terminal.
* For example, if you have myprogram.exe in a directory added to the PATH, typing myprogram will execute it.

**2) Script Executable Files**

* These are scripts written in languages like Python, Node.js, Bash, etc., and are not directly executable by the operating system unless a **runtime** or **interpreter** is specified.

**Examples:**

* A Python script: myscript.py
* A Node.js script: myscript.js
* A Bash script: myscript.sh

**How Commands Get Executed in our Terminal:-**The command written in the terminal will be executed on the basis of :-

* Alias
* Function
* Built-in
* Hashed Array or Table
* Executable

**Packages**:- The modules present inside the node\_modules folder are called as package or npm module.

* There are 2 types of package which can come inside our node\_modules folder which are Library package and CLI package . And these 2 package will go in the category of either local or global . And the category tells us that whether the package is present globally in the system or in the locally created project .
* The Library packages should be treated as dependencies for our project and the CLI package should be treated as dev dependencies for our project .
* When we install the CLI package , then in the node\_modules folder .bin folder is created . And sometimes for the Library package also .bin folder is created because Library package can also be dependent on some other CLI package.

**NPX**:-npx stands for **Node Package Executor**. It is a command-line tool included with Node.js (starting from version 5.2.0 of npm, which comes bundled with Node.js).npx searches for a file and executes it. Its primary purpose is to execute Node.js packages directly without having to install them globally.

**How npx Works**

* When you run an npx command followed by the file name then:-
  1. If :- It Searches for package.json file with the following mandatory fields “name” and “bin”.The name filed should be same as the name written after the npx command and the bin points to the file which will run if name field matches.
  2. Else If :- It checks if the package exists in your local node\_modules/.bin directory.
  3. Else If :- Searches for a package in the global space where all our global packages are installed. (We can check the path where our global packages are installed using this command :- npm root –g ) . So from this step we find that we need to add that global path in our environment variables. Also if the package is present in the global space then we can execute that package directly by its name in the terminal without using npx prefix . It is achievable because the path is added in the environment variable.
  4. Else If:- It checks the package in the npm-cache folder which is present in the local folder of the the parent directory of global folder .
  5. Else If:- It will go to the npm website and check for the package name and check whether the package exists there or not and also checks wheter that package is cli package or not . If it is cli package then it prompts us to download the package and if we allow then it downloads the package from the npm registry and stores the package in the npm-cache folder . Executes the command or script. Now one thing to note here is that the npm-cache folder path is not present in our environment variable . And that’s why we have to run the package present inside our npm-chache using the npx command followed by the package name. We can’t directly run the package by using it’s name even if the package is present inside our npm-cache folder.

So from my point of view the benefit of installing the package using npx is that if the package doesn't existed in the local or global then it downloads the package in the npm-cache folder and we don't need to add the path of that folder in our environment varialbe. But i still think that there is no such big contribution from the npx in terms of downloading a package as it is only able to differentiate the cached package from the non-cached package.

However npx gives us opportunity to run the package . But there are several alternatives for running a package

There is one more usecase of npx is that You can specify the version of a package to use with npx, which is helpful for running a specific version of a CLI tool. And it is helpful because the downloaded versioned package is installed in the cached folder if not already exist. And thus it doesn’t modify the dependencies of our ongoing project and also don’t pollute our global packages folder space.

npx create-react-app@4.0.0 my-app (Example)

Brief description of using specific version of package using npx command is :-

### ****Process of**** npx ****with a Specified Version****

#### 1. ****Checking Local**** node\_modules ****Directory****

* **What Happens**:
  + First, npx checks if the specified version of the package exists in your **local project's node\_modules directory**.
* **Outcome**:
  + If the specified version is already installed locally, npx uses it directly.
  + If a different version is present locally, npx **ignores** it and proceeds to fetch the specified version.

#### 2. ****Checking Global Directory****

* **What Happens**:
  + If the specified version isn't found locally, npx checks the globally installed packages.
* **Outcome**:
  + If the **exact version** specified is installed globally, npx uses it.
  + If another version is installed globally, npx skips it and moves to the next step.

#### 3. ****Checking the**** npm-cache ****Folder****

* **What Happens**:
  + If the specified version isn’t found in local or global directories, npx looks in the **npm-cache** folder.
* **Outcome**:
  + If the exact version is cached, npx uses the cached version.

#### 4. ****Fetching from npm Registry****

* **What Happens**:
  + If the specified version is not found in any of the above locations, npx downloads the required version from the **npm registry**.
* **Outcome**:
  + The version is cached in the **npm-cache folder** and used to run the specified command.

**Comparison Between npm install and npx**

* npm install <package>:
  + Always installs the specified version to your project or globally.
  + You manage the installation paths.
  + You need to manage different versions manually.
* npx <package>@<version>:
  + Ensures the exact specified version is used for the command.
  + Avoids cluttering local or global installations.
  + Automatically downloads and caches the package if required.

**Comparison Between npm install and npx**

* npm install <package>:
  + Always installs the specified version to your project or globally.
  + You manage the installation paths.
  + You need to manage different versions manually.
* npx <package>@<version>:
  + Ensures the exact specified version is used for the command.
  + Avoids cluttering local or global installations.
  + Automatically downloads and caches the package if required.