**NODE PACKAGE MANAGER (NPM)**

npm – Package manager for JavaScript

Package manager – To solve a problem statement –> write a code, this can be reusable code.

Individual files containing reusable code are called modules.

A package is a directory with one or more modules and a special file named package.json. Package.json has metadata about the package.

Package == module

Q. How does NPM help to manage these packages?

NPM allows you to publish reusable code/modules into the NPM registry so that other developers can just reuse your code. When you make updates to your code, NPM makes it easy for other developers who are dependent on your code to check for your updates and download them.

NPM is a way to share your code with other developers, reuse the code for other developers, and easily manage the different versions of your code.

**GETTING STARTED**

npm is shipped with node js.

To check the version of node js type **node -v** and hit enter.

To check the version of npm type **npm -v** and hit enter.

**Npm, help –** to get help on npm. This will show the typical usage of npm.

**npm <command> -h** – This provides help for the mentioned command.

**npm help-search** command is used to search for particular information in the npm document. Ex. **npm help-search update**

Try **npm help update.**

**PACKAGE.JSON**

In any JS project, we include a file called package.json in our project folder.

Package.json files have two benefits:

1. Manage dependencies of the project – mention versions of dependencies.

2. Add scripts – Scripts help with the initial build of the project.

Use the **npm init** command to create a package.json file in any project, and click enter.

**PACKAGE.JSON defaults**

To override npm defaults, ex. the author name type **npm config set init-author-name “Nikita” or npm set init-author-name “Nikita”**

Then run **npm init –yes**

To get the default set values, use the command – **npm config get init-author-name**

To delete the default set values, use the command – **npm config delete init-author-name**

**INSTALLING LOCAL PACKAGES –** packages only for a specific project

Ex. **npm install moment,** this will install a package under the node\_modules folder.

To add this dependency to package.json, use the –save flag

Ex. **npm install moment –save**

This will create a new node ‘dependencies’ inside package.json

To add a particular package only for the sake of development and not use it as a production dependency, use the –save-dev flag

Ex. **npm install moment –save-dev**

This will create a new node ‘devDependencies’ inside package.json

**UNINSTALLING LOCAL PACKAGES –** Uninstall the package from the project

Ex. **npm uninstall moment,** this will install a package under the node\_modules folder. This will not remove the entry from the package.json file. To do that use, **npm uninstall moment –save-dev**

**INSTALLING PACKAGES GLOBALLY –**

Use the command **npm install moment -g.** This will install a package under the node\_modules folder. This command does not add any dependency to package.json

**UNINSTALLING PACKAGES GLOBALLY –**

Use the command **npm uninstall moment -g**

**ALIASES for UNINSTALL**

* npm remove <package-name>
* npm rm <package-name>
* npm un <package-name>

**TO CHECK THE LIST OF GLOBALLY AND LOCALLY INSTALLED PACKAGES**

Use **npm list,** or **npm list –depth 1**

Depth 1 displays 1 dependency of installed packages

Depth 0 displays no dependencies on installed packages

To check global packages, use

**npm list –global true –depth 0**

**NPM VERSIONING**

**Semantic versioning –** It is a representation where the version is represented by three numbers that mean the same thing for every developer.

The first number represents the major version number, the second number represents the minor version number, and the third number is the patch number.

Anytime there is a bug fix or performance improvement, the patch number gets updated.

The minor version gets updated whenever there’s a new feature, but that feature is not going to break any existing functionality.

The Major version number is updated only when there’s a breaking change or a break in the existing functionality.

To install a specific version of the package, use the @ symbol with npm install @<package-name>.

**Meaning of ^ -** Stick to the major version but retrieve the latest minor and patch version

**Meaning of ~ -** Stick to the major and minor versions as is and retrieve the latest patch version

**NPM UPDATE**

To update any package use, **npm update <package-name>**

To update all packages just use, the **npm update**

To update npm itself, open the command prompt with administrative privileges and run **npm install npm@latest**

**NPM PRUNE**

To remove extraneous packages from the project folder, ones which are not added to the package.json file, use the command – **npm prune**

This will remove all extra packages from the project folder which are not mentioned in the package.json file.

**NPM SCRIPTS -** To run multiple commands in go, use npm scripts