**NPM COMMANDS:**

* npm -v (for checking version)
* npm install npm@latest -g (this will update to latest version)
* npm init (this will create packge.json)
* npm init --yes (this command will not ask for question)
* npm update (to update local package but updates only minor and patch)
* npm outdated (list all the outdated packages)
* npm-check-updates or ncu (for checking latest updates)
* ncu -u (to update package.json but it will not update package. We have to update it manually)
* npm i <package\_name>@<version>
* npm uninstall( or un) <package\_name>

[g ---> GLOBAL]

* npm install -g <package\_name> (to install a package globally)
* npm update -g <package\_name>
* npm update -g (update all global packages)
* npm uninstall -g <package\_name>
* npm install express — save
* npm list (to list all the dependencies and their exact version)
* npm list --depth=0 (to list only those dependencies on which our app depends not

dependencies of dependencies )

* npm view <package\_name> (gives metadata about that package)
* npm view <package\_name> <property name>(for e.g. dependencies ) (gives only

dependencies of a particular package)

**To add an entry to your package.json's dependencies:**

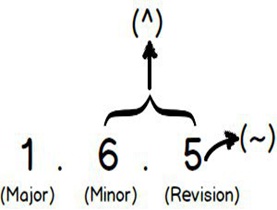
* npm install <package\_name> [--save-prod] (by default)

**To add an entry to your package.json's devDependencies:**

* npm install <package\_name> --save-dev (save as developer dependency)

**===> PACKAGE.JSON**

* Most software versions follow semantic versioning. In semantic versioning, versions are divided into three distinct numbers as shown in the image below.
* A version is made up of three parts: X,Y,Z where those are major, minor and patch versions respectively.
  + - A change in patch represents a bug fix that doesn’t break anything.
    - A change in minor version represents a new functionality that doesn’t break anything.
    - A change in major version represents a large change that breaks compatibility.



* By default, npm installs the latest version, and prepends a caret e.g. “^1.2.12”. This signifies that at a minimum, version 1.2.12 should be used, but any version higher than that is OK, as long as it has the same major version, 1. Since minor and patch numbers only represent bugfixes and non-breaking additions, you should be safe to use any higher same-major version.
* “^1.8.3” means that update is acceptable until the major version is 1.
* “~1.8.3” means that update is acceptable until the major version is 1 and minor version is 8.
* “^1.8.3” is equivalent to “1.x” and “~1.8.3” is equivalent to “1.8.x”.
* “1.8.3” in this case this exact version is used (no update)

**------------------PACKAGE-LOCK.JSON------------------**

**The Format**

Package-lock is a large list of each dependency listed in your package.json:

* The specific version that should be installed
* The location of the module (URI)
* A hash that verifies the integrity of the module,
* The list of packages it requires, and a list of dependencies.
* The idea then becomes that instead of using package.json to resolve and install modules, npm will use the package-lock.json. Because the package-lock specifies a version, location and integrity hash for every module and each of its dependencies, the install it creates will be the same, every single time. It won’t matter what device you are on, or when in the future you install, it should give you the same result every time