to create a package.json file

**npm init**

before adding any packages in project in node

always create the package.json file using npm init

faster way to create package.json without any questions

**npm init --yes**

installing npm package – underscore

npm i underscore

steps to ignore node\_modules folder :



* create a .gitignore file



* in the file type node\_modules/

In terminal run –

**git status**

**git add .**

**git commit –m “firstcommit”**

**SEMANTIC VERSIONING**

**Major.Minor.Patch**

example :

“mongoose” : “^4.13.6”

The first number is what we call the major version.

The second one is what we call the minor version.

And the third one is what we call the patch version.

Or patch release, which is used for bug fixes. So let's say

tomorrow, the developers of Mongoose find a bug

in the current version. 4.13.6. It

will fix the bug and release a new version and that version would be

4.13.7.

So when they fix a bug, they will increase the patch version.

The minor version is used for adding minor features that don't

break the existing API, so if the Mongoose team

add a new feature without breaking the existing API, they would

increase the minor version. So that version would be 4.

.14.0. 0 because in that version

they haven't found a bug yet, so that version could be

unstable. And as they find them and fix bugs, then they

will increase the patch version. Okay? And finally

if they add a new feature that could potentially break the existing applications

that depend upon this version of Mongoose, then they will

increase the major version. So the next major version

would be 5.0.0.

**^ ( called caret) : Means major version is fixed for this project , and minor and patch may increase to the latest versions when u do npm install again**

**^4.13.6 is similar to 4.x**

**~1.8.3 is similar to 1.8.x , here the major and minor are fixed ( ~ is called TILDE)**

* **LISTING THE INSTALLED PACKAGES**

npm list

* **LISTING DEPENDENCIES OF YOUR APPLICATION ONLY**

npm list - - depth = 0

* **VIEWING REGISTRY INFO FOR A PACKAGE**

npm view mongoose dependencies

* **VIEWING ALL THE AVAILABLE VERSIONS FOR A PACKAGE**

npm view package\_name versions

* **DOWNGRADING VERSION OF A PACKAGE ALREADY INSTALLED OR U WANT TO INSTALL**

npm i mongoose@2.4.2

* **FINDING OUTDATED PACKAGES IN PROJECT MODULES**

npm outdated

npm update

NOTE : npm updated only works with patch or minor changes and not major changes

\*NOTE THE BELOW IS NOT RECOMMENDED AS IT MAY BREAK THE PROJECT

* **TO UPDATE PACKAGE TO MAJOR LATEST VERSION**

**npm i -g npm-check-updates**

this will install the package npm-check-updates

To run this package type :

**npx npm-check-updates : this will show all the outdated packages**

To update to major changes run :

**npx ncu -u**  or  **npx npm-check-updates -u**

**note :** RUNNING the above command only updates the package.json and not install the package so we need to write some more commands to do major changes

To do final install run : npm install.

Some packages need not to be installed in the production dependencies rather than in the testing dependencies :

JS hint is an example of production dependency

**JS HINT :** Which is a static analysis tool for JavaScript code.

**To install non production dependencies we need to run another piece of code :**

npm i jhint -- save-dev

* **REMOVING A DEPENDENCY :**

npm uninstall jshint --

* **REMOVING A DEV DEPENDENCY :**

npm uninstall – save-dev jshint

* **WORKING WITH GLOBAL PACKAGES**

npm i -g package\_name :install package globally

npm view -g dependencies : shows global installed dependencies

* **PUBLISIHING A PACKAGE ( CREATING OUR OWN PACKAGES)**

mkdir lion-lib : create a library called lion

cd lion-lib

npm init –yes

create index .js

in index .js

module.exports.add = function(a,b) {return a+b;}

* **TO LOGIN TO NPM JS**

npm login

**EXPRESS :**

* **NODEMON**
* **AUTHENTICATION**

**\*bcrypt :**  used to create hash value for any string value , we use it to generate hash for passwords

bcrypt.compare(req.body.pass, user.pass) returns true if both has are same .

* JSON WEB TOKEN (JWT)

npm I jsonwebtoken

when client does a login we authenticate and send back json web token to the client

* CONFIG MODULE:

In the config module we store important keys like private keys, passwords etc

npm I config