3. Node Package Manager (npm)

Node Package Manager or npm is a command line tool as well as a registry of third party libraries that you can add to Node apps.

https://www.npmjs.com/

npm comes with Node, so when you install Node you also install npm. To check the version of npm that is running on the machine you can run `npm -v`. These two programs are developed independently. In order to install or update npm to a specific version you can run

```
sudo npm -i -g npm@7.0.0
```

package.json

Before you add any packages to your Node application, you need to create a file that is called `package.json`. This is a JSON file that includes some basic information about your app or project, such as its name, version, authors, address of git repository and so on.

To create the `package.json` file you need to run `npm init`. This will walk you through creating the file step by step. This represents metadata of the project. As a best practice, whenever you start a Node app, you need to create this file.

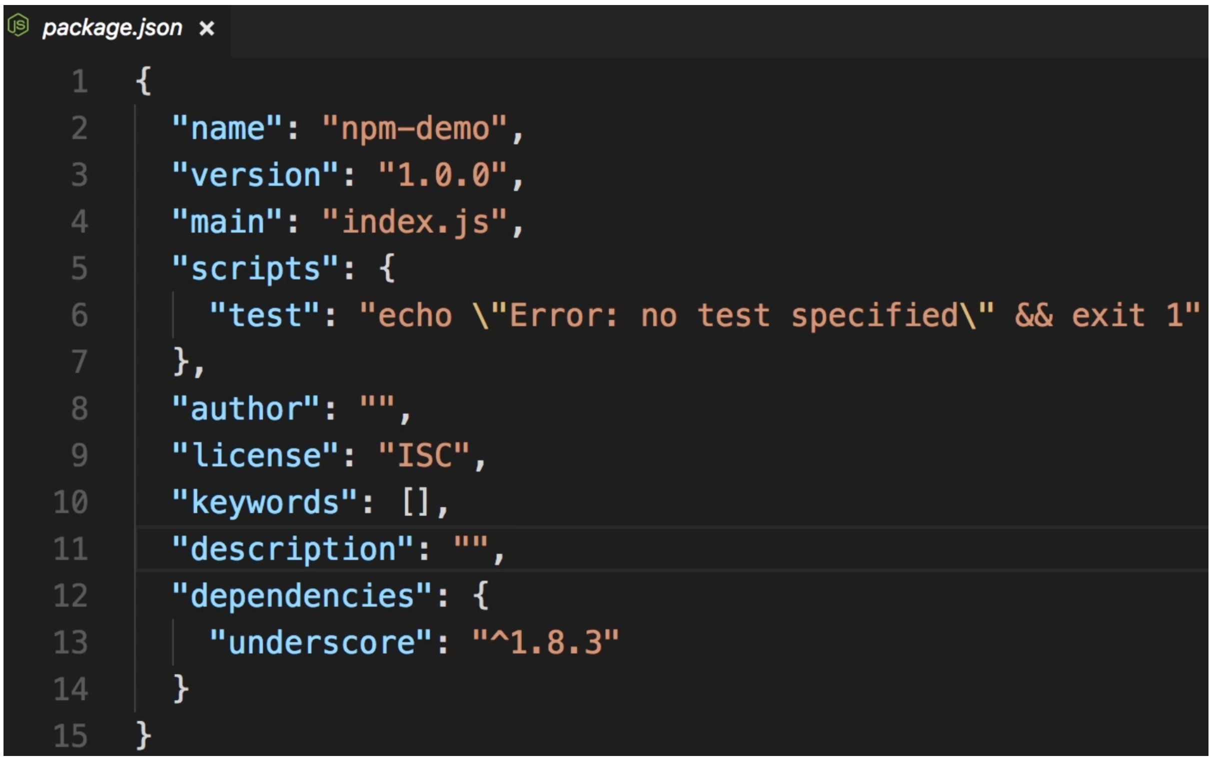
Installing a package

Firstly, you can search for any package that you want to install on the npm website. There you can see basic information about the package, such as the latest version, the publisher, the address of the github repository, some statistics and documentation among others.

To install a package you need to run the command `npm i` followed by the name of the package, for example:

```
npm i underscore
```

When you run this in `package.json` you can see a new property called dependencies and under that you can see the package that you installed with its version. npm will always install the latest version of the package, unless you specify the version that you want to install.



```json
{
  "name": "npm-demo",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "keywords": [],
  "description": "",
  "dependencies": {
    "underscore": "^1.8.3"
  }
}
```

Using a package

To use a package that you installed, you need to import it to the Module by using the require function. For example:

```
var _ = require('underscore');
```

When you supply the Module name, the require function will assume the following:

1. That it is a core Module
2. That is a file or folder
3. That is a file that exists inside the `node_modules` folder

Package dependencies

In previous versions of npm, when a package had dependencies, those dependencies would be installed inside that package folder, inside another node_modules folder. This created a mess, because the same package was installed multiple times and also in some situations it would end up with a very deeply nested structure. In recent versions of npm this behavior is changed and now all dependencies of our app as well as their dependencies are stored under a node_modules folder. There is an exception, if one of these packages uses a different version of one of these dependencies, then that version will be installed locally with that package.

NPM Packages and source control

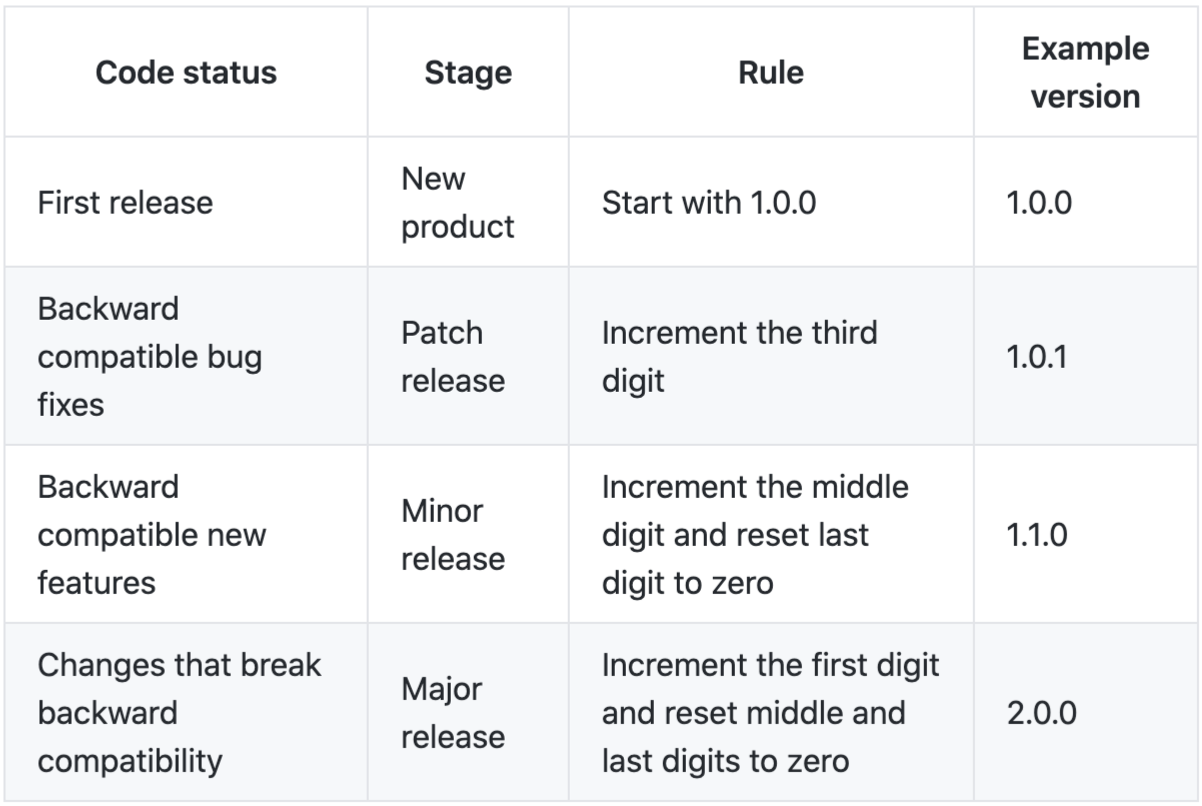
Since all the dependencies are stored in the package.json file, it is easy to restore them on any machine, so there is no need to save the node_modules folder on a repository or when sending the code to someone else. In order to restore the dependencies you simply run the command `npm i`, so npm looks at the package.json and then downloads the dependencies from npm registry. This is why you should exclude the node_modules folder from the source control repository. Assuming that you are using git, you can exclude it by adding a new file on the root of the repository with the extension `.gitignore`, then inside that file you can add node_modules/ indicating that it is a folder.

Semantic Versioning

In Semantic Versioning or SemVer, the version of a Node package has three components, the first number is what we call the Major version, the second one is called Minor version and the third one is called Patch version.

```
Major.Minor.Patch
```

- Major is used for adding new features that could potentially break the existing apps that depend upon this version of the package.
- Minor is used for adding new versions that do not break the existing API.
- Patch is used for fixing bugs.



Code status	Stage	Rule	Example version
First release	New product	Start with 1.0.0	1.0.0
Backward compatible bug fixes	Patch release	Increment the third digit	1.0.1
Backward compatible new features	Minor release	Increment the middle digit and reset last digit to zero	1.1.0
Changes that break backward compatibility	Major release	Increment the first digit and reset middle and last digits to zero	2.0.0

You can specify which update types your package can accept from dependencies in your package's package.json file. For example, to specify acceptable version ranges up to 1.0.4, use the following syntax:

- Patch releases: 1.0 or 1.0.x or ~1.0.4
- Minor releases: 1 or 1.x or ^1.0.4
- Major releases: * or x

Listing the installed packages

If you want to see the list of all installed dependencies with their exact version, you can run the command `npm list`. To check only the dependencies that you installed without their own dependencies, you simply run `npm list -depth=0`.

Another way to check the version of a specific package is to go to that package folder and open the package.json file inside that folder, then go to the bottom of it you will find the property *version*.

Viewing registry info for a package

To find all the metadata about a specific library you can run the command `npm view` followed by the package name, for example:

```
npm view mongoose
```

What you will see is the package.json file for that library. But if you are interested only in the dependencies you can run `npm view <package name> dependencies`. Another useful command is `npm view <package name> versions`, with it you can check all of the available versions of that package.

Installing a specific version of a package

To install a specific version of a package instead of the latest version, you need to run the following command:

```
npm i <package name>@<version>
```

For example:

```
npm i mongoose@2.4.2
```