**NPM**

Node Package Manager is a package manager for Javascript, it is pre-installed with Node.JS. It is used to easily install modules/packages on your system. Modules (aka dependencies) are basically JS libraries. NPM makes it easy for developers to share and re-use code.

\*DL NodeJS to get npm\*

How to use

Create a folder to begin a working project and open up the command line at the working directory location (right click folder > gitbash here).

\*basic command – *npm -v* find out current version of npm (also *npm –version*). Just typing *npm* will show help page with all commands possible\*

Run *npm init* in terminal to create *package.json* file, which can be seen as the most important file! This file has all the application information, hold all app dependencies (modules that the app depends on). After running *npm init*, the terminal will prompt you for some information, e.g. author name, application name/version etc… this can be skipped.

To install modules

*Npm install modulename*

e.g. npm install lodash

//this will now be saved under node\_modules directory in the web directory of the project

Create a javascript file with the same name as the “main” property in the package.json file. This can be changed manually.

e.g. (using lodash – JS library)

const \_ = require(‘lodash’); //required to use module

const numbers = [15,68,9,64];

\_.each(numbers, function(n, i) {

Console.log(n);

});

//you can run this script through the terminal using *node app.js* (or name of the main js file)

\*If using GitHub, you can put the node\_modules directory in the .gitignore file. If an external person wishes to contribute, they can see all modules needed by looking at the package.json file and installing modules on their own system. This can be made quick via typing *npm install* in the terminal (without specifying module name)\*

When installing modules, you can install to devDependencies (as opposed to just dependencies), in the package.json file, by using: *npm install modulename -D*

This is useful for modules which are only used in development and not production.

\*the node\_modules folder can get huge because your dependencies are stored there as well as your dependencies dependencies’ etc…\*

\*do not be concerned if the terminal replies with ‘warn deprecated’ when installing modules – this may appear if some dependencies (and their dependencies) are out of date\*

\*You can also install specific versions of modules e.g. *npm install lodash@4.17.3.* Use *npm update lodash* to update\*

\**npm list* shows all modules installed in app\*

Version numbers (semantic versioning)

4.17.3

4 = major version

17 = minor version (e.g. new features)

3 = patch version (bug fixes)

Look in the package.json file to see version number of dependencies. See the character prior to the version number:

|  |  |
| --- | --- |
| ^4.17.4 (default) | Installs latest minor version (will not install latest major version) |
| ~4.17.4 | Keeps minor version and only installs latest patch version |
| 4.17.4 | Keeps version the exact same |
| \* | Keeps up to date completely (this is generally not a good idea) |

To uninstall module

*Npm uninstall modulename*

e.g. npm uninstall lodash

Global modules

Global modules, as opposed to local modules which are just saved in the node\_modules folder of our project, are installed on your actual machine. To install a module globally, use:

*Npm install -g modulename*

To find the location of globally saved modules: *npm root -g*

These modules can now be run anywhere! Preferably within an application/project

e.g. npm install nodemon

//in terminal: *nodemon*

//this updates the script every time you save (like livereload), so no need to restart browser.

//*ctrl+c* to stop.

e.g.2 npm install -g live-server

//in terminal: *live-server*

//*ctrl+c* to stop

//This provides a live server instead of running locally

\*to uninstall a global module: *npm uninstall -g modulename*\*

Scripts

Look in package.json file and look at the scripts object – by default, “test” will be a property in this object, but this can be removed/replaced. Generally “start” is a good property to use with the main js file as the value => “start”:”node index.js”. In the terminal you can now use *npm start* to run the application.

\*For most scripts, you would have to use the keyword *run* e.g. *npm run server* – if *server* was the property in the scripts object\*