**Working with NPM**

[NPM](https://www.npmjs.com/) is a JavaScript package manager used for managing software dependencies, publishing packages so others can use them, and running applications (for instance, using npm start to start an app).

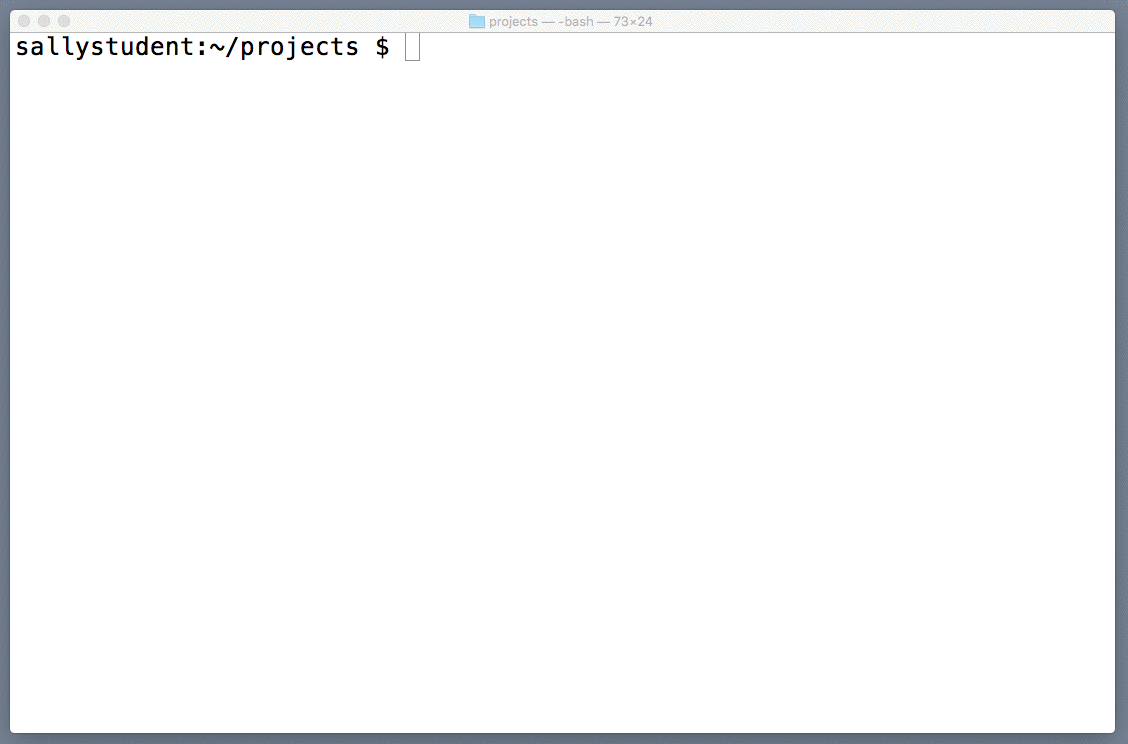
In this assignment, we'll learn how to bootstrap new projects with NPM, install dependencies, and write scripts for starting our apps.

**package.json and dependency management**

We use NPM to manage the software dependencies for a project. These dependencies are listed in a [package.json file](https://docs.npmjs.com/files/package.json" \t "_blank). To learn how this works, let's bootstrap a new Express project.

From the command line, run mkdir learn-npm && cd learn-npm to create a new folder and move into it.

Next, run the command npm init. This is the command we use to initialize new NPM-based projects. We'll just add a value for the name and description properties, and go with the defaults for the others.



If you run ls in your learn-npm folder, you'll see that NPM created a new package.json file, with the values we specified. Your package.json file should look something like this:

{

"name": "learn-npm",

"version": "1.0.0",

"description": "Learn how npm works",

"main": "index.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"author": "",

"license": "ISC"

}

You should always use npm init to create a package.json file, but know that if you were to create a package.json from scratch, it **must** contain the name and version properties. The scripts property is important, and we'll cover that in a moment.

With our new project initialized, let's try adding a dependency. Run the command npm install --save express. The --save flag tells NPM that in addition to installing Express into the current project, it should also add it to the dependencies array in package.json.

Once Express installs, run ls again from your learn-npm folder. You'll see that NPM created a new folder: node\_modules. Running ls node\_modules, you'll see that in addition to installing Express, NPM has installed a couple dozen other dependencies.

We only asked for Express, so why did it install so many other packages? This is because Express itself has numerous dependencies, which in turn themselves have additional dependencies. After NPM installed Express, it logged the dependencies it installed. You should have seen something like this:



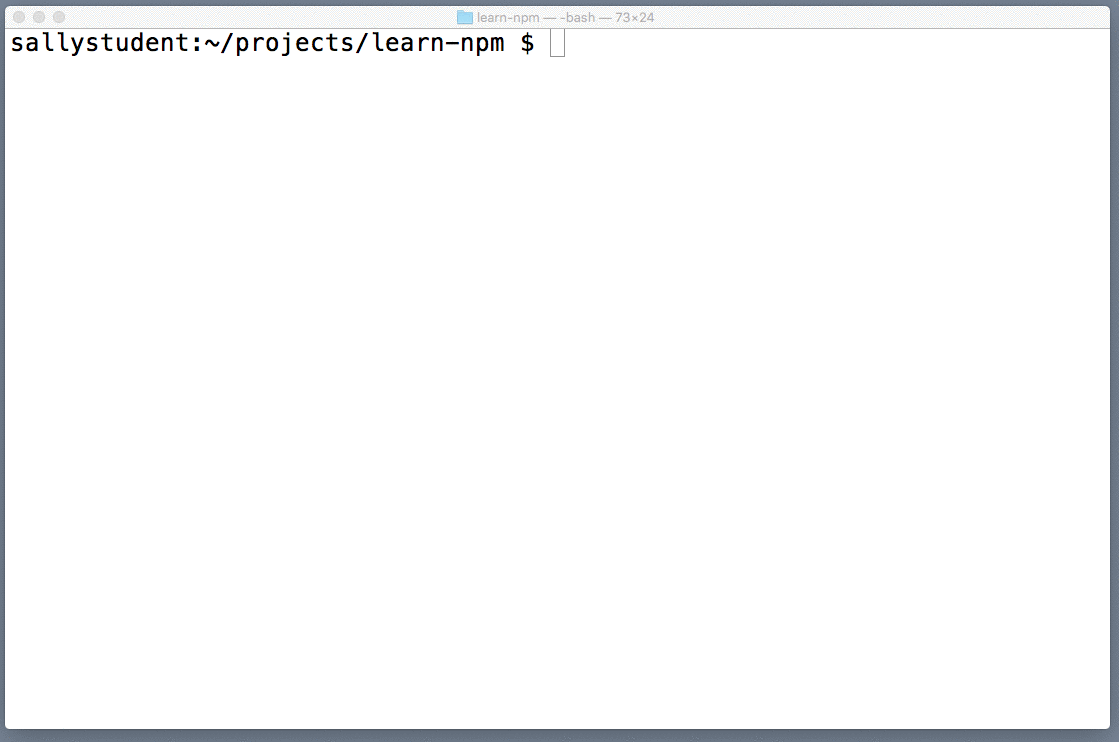
The dependencies are represented as a tree. We installed Express, which depends on a package called accepts, which itself depends on a package called mime-types, etc.

Compare this dependency tree with what you see when you run ls node\_modules, and you'll see that NPM flattens the tree — all of the dependencies get installed at the same level in the node\_modules folder. This is a good thing, because it means if two packages — call them A and B — both depend on a third package, C, that third package only needs to be installed once.

**.gitignore node\_modules**

One of the advantages of using a package manager like NPM is that we can specify a list of dependencies and commit that list to our repo, *without* committing the dependencies themselves. When a collaborator needs to install the project, they download the repo and then run npm install to install the dependencies locally.

This means that you should **not** add your node\_modules folder to Git. To ensure this, you should create a .gitignore file in the root of your project, and add node\_modules as an entry. This will cause Git to ignore the node\_modules folder and its content.



**Working with existing projects**

When you're working with an existing project that uses NPM, the typical flow is to clone the project repo to the computer you're working on, cd into it, and then run npm install. This will cause NPM to install all of the dependencies listed in package.json.

**Scripting**

When we looked at the package.json above, we saw that it contained a scripts property. By default, NPM added a stub entry for test. From the learn-npm folder, you can run the command npm test which will cause the value at scripts.test to be executed (which in this case will simply log that no tests have been specified).

NPM offers several ["hooks" for scripting](https://docs.npmjs.com/misc/scripts#description) that allow you to run commands like npm start and npm test (the two we'll be most interested in for this course).

Let's have a look at an example of a start script. From your projects folder, [clone this repo](https://github.com/Thinkful-Ed/npm-script-example) and then cd npm-script-example. Run npm install to install the dependencies, and then npm start. This will start the app, and you'll see a message logged that the server has started. Hit Ctrl-c to stop the server.

So, we ran the command npm start and Node figured out that it should run the code in server.js. Looking at the scripts.start property, we can see that we've configured our app so that when the npm start command is run, the command node server.js gets executed.

Note that we could also start our server by running node server.js directly from the command line. The beauty of using NPM scripts is that you can use a single command (npm start) on (in principle) any Node application, and the application will start. The application file could be called server.js or app.js or even supercalifragilisticexpialidocious.js, but we're able to abstract away from the particulars of the application to get a common command for starting an app.

Later in this unit, we'll see how to use the npm test command to achieve similar functionality for running our automated software tests.

**More complicated NPM scripts**

Our start command in the small app we just looked at was simple: node server.js. Sometimes, though, you may have more complicated logic around starting an app. When that's the case, the best route is to create a [shell script](https://en.wikipedia.org/wiki/Shell_script) that contains that logic, and have your NPM command point to that shell script.

**In closing**

We've covered a lot of ground in this reading. Moving forward, the key takeaways are:

* Use npm init to bootstrap a new Node project.
* Use npm install to install dependencies listed in a package.json file.
* Use npm install --save myDependency to install a dependency and add it to package.json.
* Always .gitignore your node\_modules folder.
* Use npm start to run your apps.
* When you have more complex scripting needs, point NPM to a separate Bash script file.