



Intro to npm

 <https://github.com/learn-co-curriculum/phase-2-hooks-intro-to-npm-readme> 
<https://github.com/learn-co-curriculum/phase-2-hooks-intro-to-npm-readme/issues/new>

Learning Goals

- Understand how to use npm to add functionality to JavaScript projects
- Use [npm's online platform](https://www.npmjs.com/) [\(https://www.npmjs.com/\)](https://www.npmjs.com/)
- Configure your environment to use npm

Introduction

When using React in our projects, we need to download the actual React code for our projects so that we can use it alongside our own JavaScript code. React itself is written in JavaScript, and it's an [open source library](https://github.com/facebook/react) [_\(https://github.com/facebook/react\)_](https://github.com/facebook/react). React itself *a*lso depends on some other open source JavaScript libraries. So how can we use all this excellent code ourselves?

JavaScript has been around for many years now, and continues to serve as a critical part of the modern, interactive web. There are web developers all over the world writing JavaScript code, each contributing their own bits of work. That's a *lot* of code! In fact, there is a lot of **duplicate** code. Multiple web developers, over the years, have solved the same problems over and over.

For these situations, we have JavaScript **packages**. A package is a file or set of files full of existing, *reusable* code. They are designed to be shared, allowing many web developers to use the same code in their own projects.

To help organize these packages in relation to our own work, we use **npm**, which [may or may not be](https://www.npmjs.com/package/npm#is-npm-an-acronym-for-node-package-manager) [_\(https://www.npmjs.com/package/npm#is-npm-an-acronym-for-node-package-manager\)_](https://www.npmjs.com/package/npm#is-npm-an-acronym-for-node-package-manager) short for **Node Package Manager**. In this lesson, we will be discussing how npm works and why it is useful.

The Value of Existing Code

While it is important to learn the critical skills to solve problems with code, it is equally important that we learn how to identify existing code that suits our needs and incorporate it into our projects. We don't need to always be *reinventing the wheel* and writing code that may already exist.

In fact, with the amount of developers out in the world, it is likely someone else has not only already invented the same wheel, but tested, upgraded and innovated on it so that it is way better than anything we could write ourselves in a short period of time.

Remember, programming is all about providing a solution to a problem. When 'on the job', so to speak, no one gets bonus points for concocting a novel/clever solution to a problem ^{for which} good open source code already exists.



Setting Up Node Package Manager

Before we continue, let's make sure your environment is all set to work with npm.

npm is automatically installed along with **Node.js**, which should already be installed on your system if you've worked through the JavaScript coursework. To confirm you have node installed, enter the following into your command line:

```
$ node -v
```

If a version appears, you have Node.js. If, by chance, you do not have Node.js installed, you can use the [Node Version Manager](https://github.com/creationix/nvm) [_\(https://github.com/creationix/nvm\)_](https://github.com/creationix/nvm) to install Node.js and keep it up to date.

You can also double check npm by running the following:

```
$ npm -v
```

A version number should appear in your terminal. If you'd like, you can update npm by entering the following:

```
$ npm install --global npm  
# or, for short: npm install -g npm
```

Okay, we've got it installed. But what is npm exactly?

npm Introduction

As mentioned, npm is a package manager for JavaScript. This means that npm works with your JavaScript project directories via the command line, allowing you to install packages of preexisting code.

What sort of code? All kinds! Some packages are quite small, like [is-number](https://www.npmjs.com/package/is-number) [_\(https://www.npmjs.com/package/is-number\)_](https://www.npmjs.com/package/is-number), a package that has one function: to check if a value is a number. Some packages are much more complicated. Huge libraries and frameworks, including [React](https://www.npmjs.com/package/react) [_\(https://www.npmjs.com/package/react\)_](https://www.npmjs.com/package/react) and [Express](https://expressjs.com/) [_\(https://expressjs.com/\)_](https://expressjs.com/), are available as npm packages. These larger packages are often *themselves* built using a combination of other packages.

This modular design — the ability to build a package using other packages — allows for developers to continuously expand the JavaScript universe, creating new, more powerful tools and applications on top of existing, tried and tested code.

npm install and package.json

All JavaScript labs at Flatiron rely on npm packages for their tests. Many use the [mocha](https://mochajs.org/) (<https://mochajs.org/>) npm package, which is a popular JavaScript testing framework.

The lessons themselves don't actually include all the code from these different packages directly in the source code. Instead, they contain a list of *dependencies* in a file called `package.json`.

The `package.json` file tells you (and npm) what packages are required for a specific JavaScript application, listing out each package name.

When we run the command `npm install` in a directory where a `package.json` file is present, npm reads the names of each dependency from the `package.json` file and downloads the packages from [npmjs.com](https://www.npmjs.com/) (<https://www.npmjs.com/>), where they are hosted. It then begins installing those packages — *BUT!* those packages also have *their own* `package.json` with their own dependencies! `npm` must also get those packages, and if *those packages* have any dependencies, get them as well. And so on. This is what we refer to as a *dependency tree*.

If you are working in a local environment, running `npm install` creates a folder called `node_modules`, which contains all the downloaded packages. *Note:* the `learn` gem may automatically run `npm install` when you first run `learn test`.

When building a project, you may realize you *need* some specific package. We can install packages by running `npm install <package_name>` while inside a project directory. Running this command will add the package as a dependency in the `package.json` file. This means that if you do not have a correctly structured `package.json` file, the install *will not work!* (Fortunately, `npm` has a built-in command, `npm init`, that we can use to create the `package.json` file. We'll learn more about it a bit later in this lesson.)

A Little More on package.json

The `package.json` file is a key part of sharing JS code repositories on sites like GitHub. Instead of having to include all the dependencies' code with every project, we just include a small file, listing out what npm needs to get for the project.

The file also typically includes information about the project, such as the name, version, author and license.

The `package.json` file is written in JSON, so like an object in JavaScript, it is always wrapped in curly braces, and includes keys and values. A basic example:

```
{
  "name": "react-hooks-intro-to-npm-readme",
  "version": "1.0.0",
  "description": "An introduction to npm and package.json",
  "main": "index.js",
  "scripts": {
    "test": "echo 'hot dog'"
  },
  "dependencies": {
```



```

    "is-number": "^7.0.0"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/learn-co-curriculum/react-hooks-intro-to-npm-rea
  },
  "author": "flatironschool",
  "license": "ISC",
  "bugs": {
    "url": "https://github.com/learn-co-curriculum/react-hooks-intro-to-npm-rea
  },
  "homepage": "https://github.com/learn-co-curriculum/react-hooks-intro-to-npm-rea
}

```

In your terminal, if you are in a directory with the above `package.json` file present, running `npm test` will return "hot dog." This lesson actually does include this `package.json` file, so try it for yourself!

This works because the command `npm test` is saying: "Hey npm, look in `package.json` and find the script with the name of 'test', then execute its value in the terminal."

Having this file present also means it is possible to install additional packages. There is one dependency already included:

```

"dependencies": {
  "is-number": "^7.0.0"
}

```

Running something like `npm install react` will add a second dependency:

```

"dependencies": {
  "react": "^17.0.2"
}

```

Try it now! Then take a look to see just how many dependencies (which React relies on) have been added to your `node_modules` directory.

npm init

Since npm relies on a `package.json` file, it has a built in command to *build* `package.json` files. When you're creating a new project from scratch, running `npm init` on the command line will begin a series of prompts, asking about specific content to include in the file. At the end, it will create a file or edit an existing `package.json` file.

Conclusion



For all React lessons, we rely on npm to set up a lot of things 'under the hood'. The applications we build are made possible by the contributions of thousands of other coders before us!

Remember! While endlessly fun, programming is a means to an end: we (or our employer) has a problem, and we give the computer instructions to crush the problem. If available, open, and secure code already exists do not hesitate to use it! Compared to physical goods, code snippets have less value attributed to novelty. (There is a reason you won't see "artisanal code" being sold on [Etsy](https://etsy.com) [_ \(https://etsy.com\)_](https://etsy.com).)

Resources

- [npm](https://www.npmjs.com/) [_ \(https://www.npmjs.com/\)](https://www.npmjs.com/)
- [About npm](https://docs.npmjs.com/about-npm) [_ \(https://docs.npmjs.com/about-npm\)](https://docs.npmjs.com/about-npm)