<https://medium.com/coinmonks/everything-you-wanted-to-know-about-package-lock-json-b81911aa8ab8>

Let’s say we create a new project that is going to use [express](https://expressjs.com/). After running `npm init`, we install express: `npm install express — save`. At the time of writing, the latest express version is 4.15.4. So “express”: “^4.15.4” is added as a dependency within my package.json and that exact version is installed on my machine. Now maybe tomorrow, the maintainers of express release a bug fix, and so the latest version becomes 4.15.5. Then if someone were to want to contribute to my project, they would clone it, and run `npm install.’ Since 4.15.5 is a higher version with the same major version, that is installed for them. We both have express, but we have two different versions. Theoretically, they should still be compatible, but maybe that bugfix affected functionality that we are using, and our application would produce different results when run with express version 4.15.4 compared to 4.15.5.

**Package-lock**

**The Goal**

The purpose of the package-lock is to avoid the situation described above, where installing modules from the same package.json results in two different installs. Package-lock.json was added in npm version 5.x.x, so if you are using major version 5 or higher, you will see it generated unless you disabled it.

**The Format**

Package-lock is a large list of each dependency listed in your package.json, the specific version that should be installed, the location of the module (URI), a hash that verifies the integrity of the module, the list of packages it requires, and a list of dependencies. Let’s take a look at what the entry for express is:

"express": {  
 "version": "4.15.4",  
 "resolved": "https://registry.npmjs.org/express/-/express-4.15.4.tgz",  
 "integrity": "sha1-Ay4iU0ic+PzgJma+yj0R7XotrtE=",  
 "requires": {  
 "accepts": "1.3.3",  
 "array-flatten": "1.1.1",  
 "content-disposition": "0.5.2",  
 "content-type": "1.0.2",  
 "cookie": "0.3.1",  
 "cookie-signature": "1.0.6",  
 "debug": "2.6.8",  
 "depd": "1.1.1",  
 "encodeurl": "1.0.1",  
 "escape-html": "1.0.3",  
 "etag": "1.8.0",  
 "finalhandler": "1.0.4",  
 "fresh": "0.5.0",  
 "merge-descriptors": "1.0.1",  
 "methods": "1.1.2",  
 "on-finished": "2.3.0",  
 "parseurl": "1.3.1",  
 "path-to-regexp": "0.1.7",  
 "proxy-addr": "1.1.5",  
 "qs": "6.5.0",  
 "range-parser": "1.2.0",  
 "send": "0.15.4",  
 "serve-static": "1.12.4",  
 "setprototypeof": "1.0.3",  
 "statuses": "1.3.1",  
 "type-is": "1.6.15",  
 "utils-merge": "1.0.0",  
 "vary": "1.1.1"  
 }  
 },

Equivalent entries can be found for every package listed in the “requires” section.

The idea then becomes that instead of using package.json to resolve and install modules, npm will use the package-lock.json. Because the package-lock specifies a version, location and integrity hash for every module and each of its dependencies, the install it creates will be the same, every single time. It won’t matter what device you are on, or when in the future you install, it *should* give you the same result every time, which is very useful.

**The Controversy**

So if package-lock is supposed to solve a common problem, why are the top search results for it (other than npm documentation) all about disabling it or questioning the role that it plays?

Before npm 5.x.x, package.json was the source of truth for a project. What lived in package.json was law. npm users liked this model and grew very accustomed to maintaining their package file. However, when package-lock was first introduced, it acted contrary to how many people expected it to. Given a pre-existing package and package-lock, a change to the package.json (what many users considered the source of truth) was not reflected in the package-lock.

**Example:** Package A, version 1.0.0 is in the package and package-lock. In package.json, A is manually edited to version 1.1.0. If a user who considers package.json to be the source of truth runs `npm install`, they would expect version 1.1.0 to be installed. However, version 1.0.0 is installed, despite the fact that v1.1.0 is listed is the package.json.

**Example:** A module does not exist in the package-lock, but it does exist in the package.json. As a user who looks to package.json as the source of truth, I would expect for my module to be installed. However since the module is not present in package-lock, it isn’t installed, and my code fails because it cannot find the module.

Much of the time, because they couldn’t figure out why their dependencies weren’t being installed correctly, users either deleted package-lock and reinstalled, or would disable the package-lock altogether.

This conflict between expect and real behavior sparked a very [interesting issue thread](https://github.com/npm/npm/issues/16866) in the npm repo. Some people thought that the package.json should be the source of truth, some people thought that since package-lock is what npm uses to create the install, that should be considered the source of truth. The resolution to this controversy lies in npm [PR #17508](https://github.com/npm/npm/pull/17508). Npm maintainers added a change that causes package.json to overrule the package-lock if package.json has been updated. Now in both above scenarios, the packages that a user would expect to be installed are installed correctly. This change was released as a part of npm v5.1.0, which went live on July 5th, 2017.