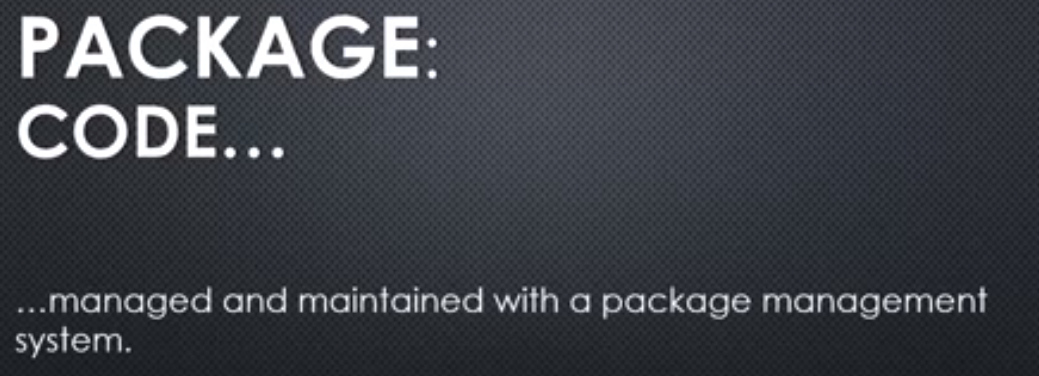
# **NPM : The node package manager**

## **Conceptual Aside : Packages and Package Managers**

In this section, we'll talk about something very exciting.

**The largest ecosystem of open source code in history and what gives you access to it. NPM, the Node Package Manager.**

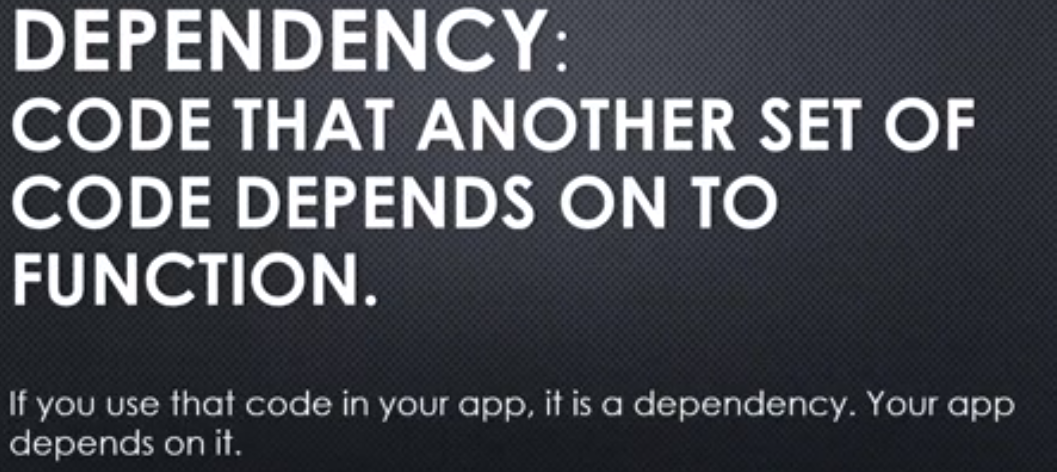


A package, when it comes to software and package managers a package is code. That's really it.

**It's just a collection of code, that works and you can use it in your code. And it's managed and maintained with a package management system. So that makes it a package. You manage and maintain it inside the system and when it's inside that system it's called a package.**



**So, your package then gets installed via this system. The code gets installed via this system. And it gets updated to new versions via this package management system, this software that does that for you, rather than you having to manually download it and figure which version you have and which the new version is, etc. And it deals then not just with what you have or need as far as the version of that code, but it manages dependencies also.**



A dependency, code that another set of code depends on to function. So your code, if you use a package in your code, you depend on that package. You depend on that other code. And the package might itself have dependencies where it depends on other code to work, etc., etc. So this can get really crazy if you don't have a good package management system to know what code depends on what code

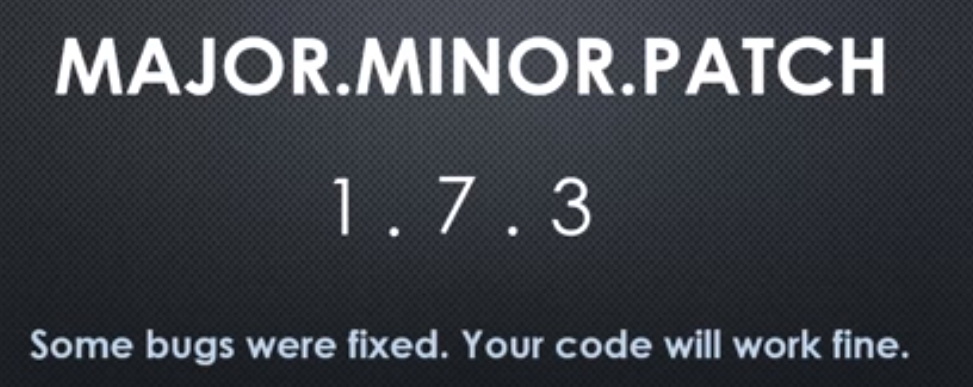
**Packages are just code that are ready to be used.And a package management system then enables us to download that package, install it into our software to use, and update it when new versions come out.**

## **Semantic Versioning (SEMVER)**

Versioning: specifying what version of a set of code this is, so I have a set of code, I have a piece of software and I want to say what version it is, so others can track if the new version has come out, watch for new features, or watch for what's called breaking changes, where if they update to the newest version, some of the code they've written against the old version might break, because of things that have changed.

**and the word semantic in this case, this idea of semantic versioning, just implies that something conveys meaning. So when we say that there's a version, a version number, that that number actually means something, that's the idea. so semantic versioning was an endeavor to make sure that just looking at the version gives us some kind of information**

**the version number is based on rules. So if you agree to use semantic versioning, you agree to follow a certain set of rules, when determining the current version number of the code you're writing. I make a change and so I will change the version number this way.**



The basic core of semantic versioning is this, the version number is major changes dot minor changes dot patches.

What does that mean?

Well, if my current version of the code I've written is let's say, 1.7.2**,** **I would increment the patch number if some bugs were fixed.** So I make some bugs fixes and anyone out there using my code, their code to use my code will still work fine. I didn't make any changes, that would break someone else's code, that was using my module, or my package. So some bug fixes, your code will work fine.



**If I increment the second number, a minor update. That means that I've actually added some new features, not just bug fixes, but your code will still work just fine. It's still backward-compatible. Anything you've written and are using my package, my code, your code will still work just fine.**



**if I make a major change and update the version, the major portion of the version number, that means big changes and your code will maybe break.** So if you update to the latest version, I made from a one point zero dot zero to a two point zero dot zero, or whatever the case is, upping that major number, **that means that you need to take a good look at your code, if you do the update, because something major might have changed and I should probably document the kind of breaking changes, that it would cause.**

**So the numbers mean something. That makes it semantic, so semantic versioning and if I agree to follow this, then I help people using my code and other people who follow this are helping me when I'm using their code, so I know whether or not to update to the latest version. If it's just a bug fix, there's no problem, I'll update. If it's a minor fix, I'll update and check out the new features to see if I want to use them and if it's a major fix, I'll be careful and see what things will break in my code, if I update this package, that I'm using.**

**For more information –**



## **NPM and NPM registry : Other’s people code**

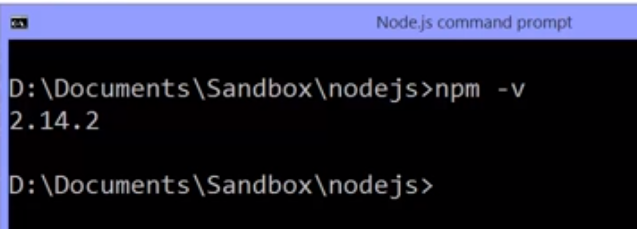
it's time to take a look at nodes package manager and **where all these packages live, the NPM registry.**

**To be clear,** **when we talk about NPM, we're talking about two different things sometimes when developers talk about it.**

**1)NPM the registry, the packages themselves, the collection of packages and NPM the programme, the actual programme that's installed on your computer 2) the package management system itself.**

**So NPM comes along for the ride when installed along with Node.js. So NPM essentially comes with Node.js.**

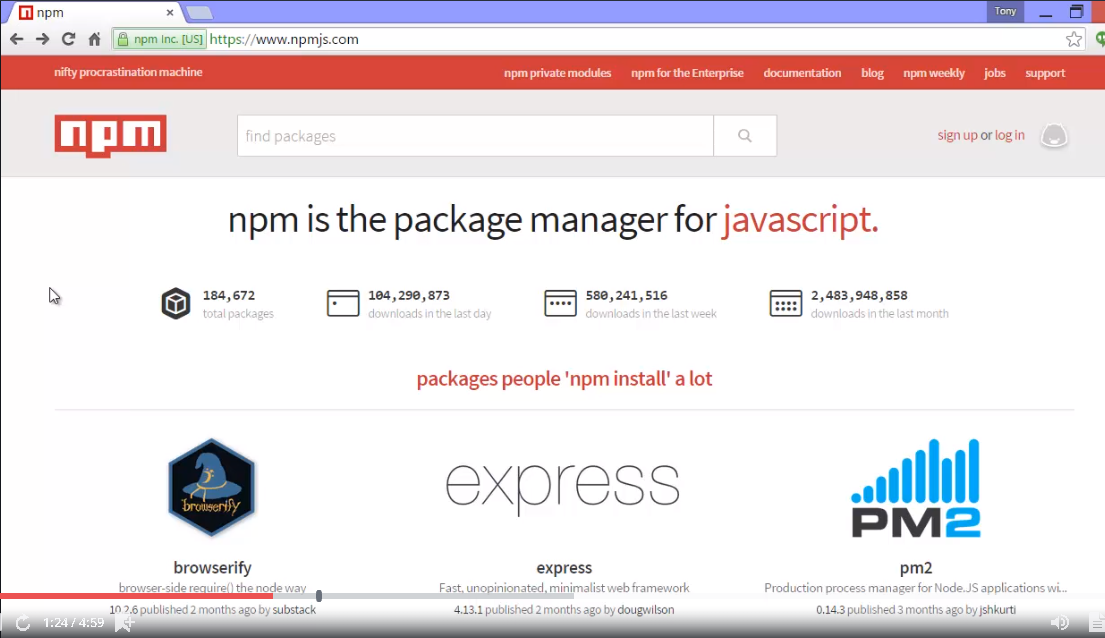
To make sure you have NPM installed, you can do the same thing that we did with node. In your command line type **npm –v** and you should get a version number. If you have a version number, then NPM is installed properly.



**Now the node package manager,** **when you run it to install a package,**

**that is to download some code for use in your application, where does it come from?**

**The code comes from the NPM registry and you can search the registry and look at individual packages and details on them at npmjs.com.**



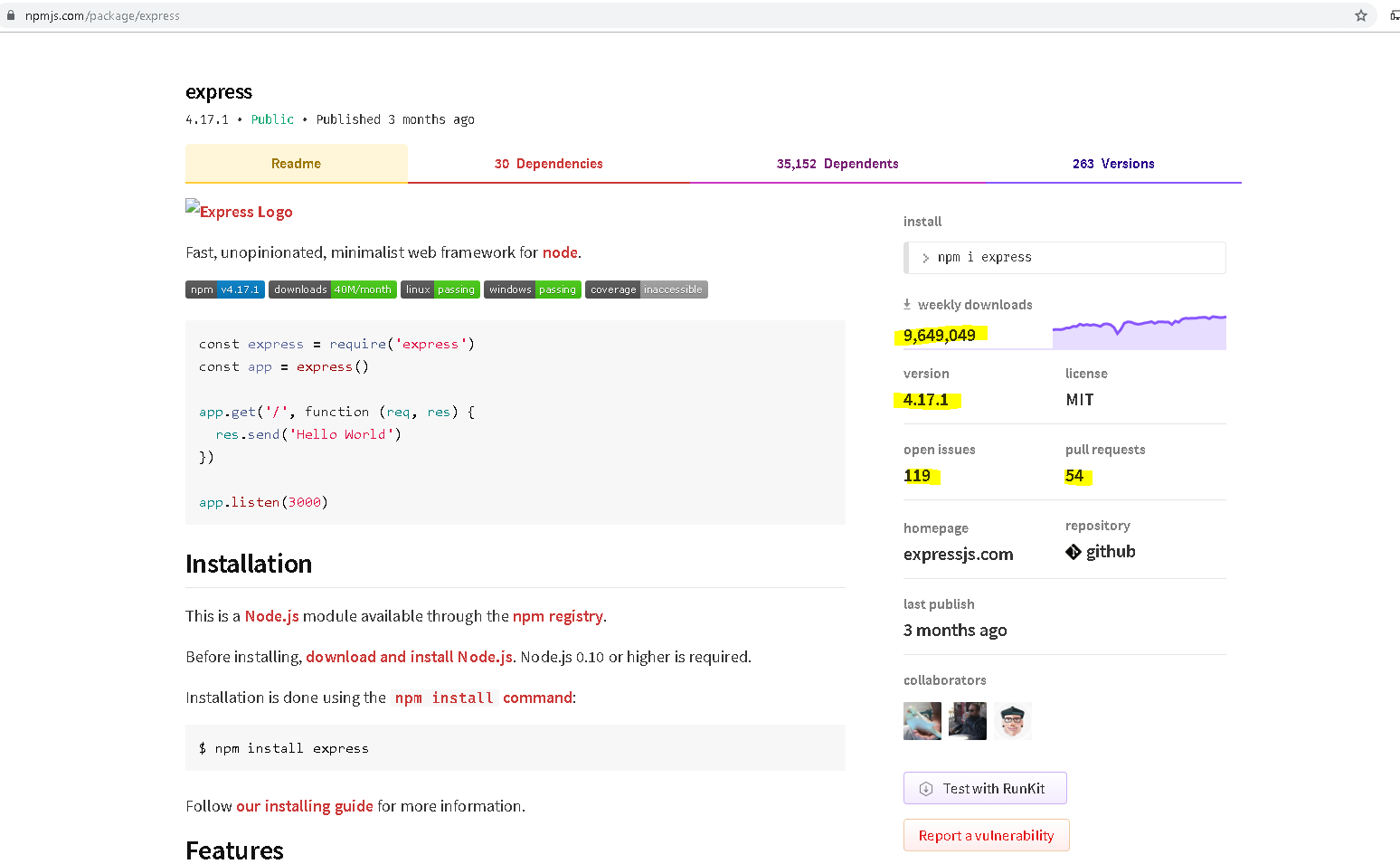
NPM, the registry is available to everyone. Anyone can put packages into NPM. And NPM literally has tens of thousands of packages available. And literally billions of downloads. So that doesn't mean that everything in NPM is good. You have to be judicious and take care when choosing the packages that you're going to install and use in your application.

For example, I could search here and I can go down to popular packages. Express is the one that we're going to installl soon. So if I click on it, I can see some important information that helps me when choosing a package. I can see some documentation on how to use it. I can see the semantic version, the current version that it's on. And **usually there's a link to actually look at the source code of it on github.** So I can see how popular it is on github

and what the code looks like. I can see the licence, so how am I allowed to use it, and the stats, that's very important. How many people are downloading it. How many people are using it?

**The open issues on github means how many issues have been submitted to github, people having some kind of problem or feature request. And open pull requests means people who have made changes to it and requesting their changes be admitted to the main package. And that has to do with open source code, where anyone can really try working on it. And there always should be some way of telling you how to install it.**

We always run the command npm then install and then each package has a unique name. So if I ran **npm install express**, while I'm sitting on the folder that I want to install into, that is the application that I'm using or creating and want to use this in, then it will download the package



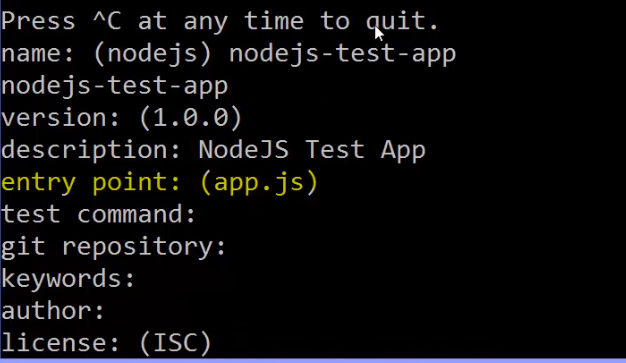
the idea being that you should look around; you should Google it; you should look at it on github; you should look at how many downloads and how people are using it; and see what kind of documentation it has. And you shouldn't be afraid of going to github and looking at the code.

Looking at the JavaScript code consists all JavaScript code and all open source that you can go and look at to see if it looks like something that's going to make sense for your application. **So we have this massive ecosystem of open source software of people just making code they've written available for free.**

## **npm init , nodemon and package.json**

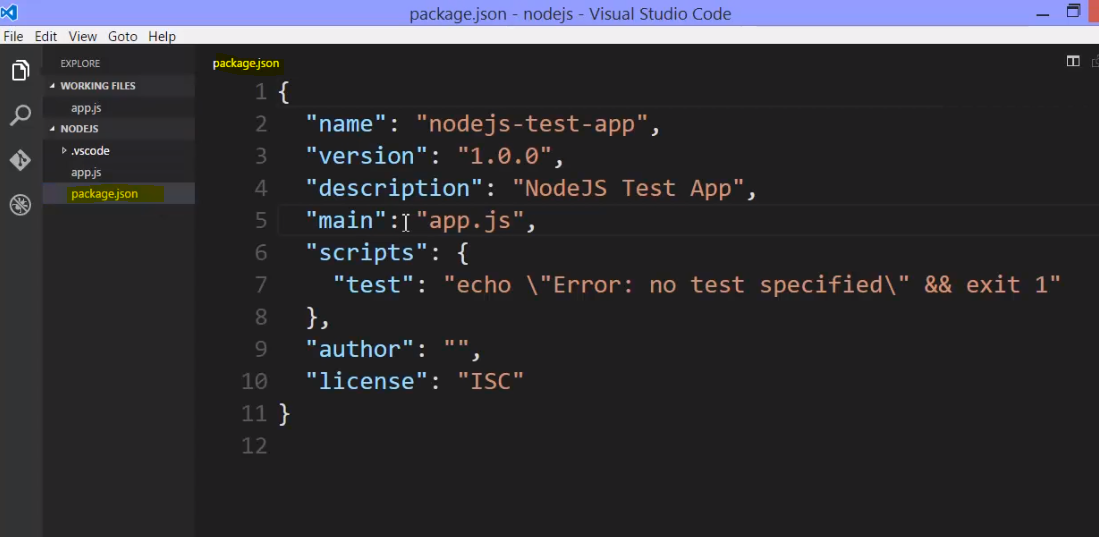
**we have to set-up, my app, so that it can keep track of the packages that I'm installing, the versions that I'm using, and that I could give it to another developer and they could easily also know which packages this app depends on.**

So to use that we're going to use **NPM init.**



**Entry point is what is a Javascript file that Node will run, we may have multiple Javascript files in different modules but there's always one Javascript file that's the entry point that we actually give to Node.**

**All right so what did NPM init do. Well you see it created some JSON. All my settings, and it stored it in a file that it just created for me called package.json.**

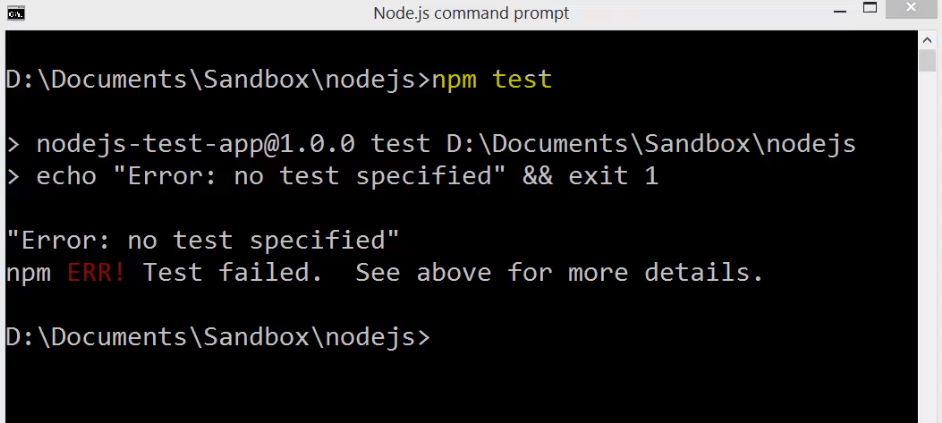


**Now I could have done this manually, but you can use NPM init when you initially set up your new app, to help you do it quickly.**

Now what's in here?

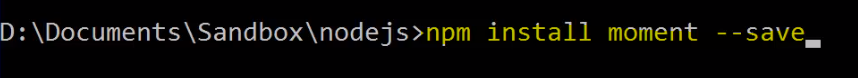
Well just a few things, the things that we entered.

**One thing of note is this script section. This is basically command line, commands. Given a name. And I can say NPM and then this name, and run some scripts. So this is a way for me to quickly run some command line, so I could say NPM test**, and it says "no test specified."



So, it echoed which just means print out to the console. So I can make different scripts, and different names, and then NPM uses package.json to say "oh I know the name of that script and I'll run these commands." So that's one useful thing.

But we're primarily interested in installing, and using packages from the NPM registry. So let's go grab moment.



npm instal moment, and then I put --save. I could also put the --save after the word install.

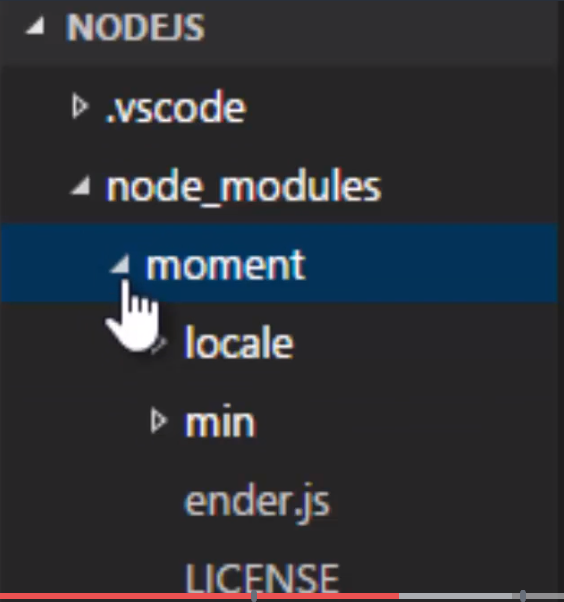
**But the idea is that** **NPM, the programme installed on my computer, will go to the NPM registry, download the files that involve moment, which means I'll have a module because all the NPM packages are structured as Node modules. I'll have a module available to me now in my app that I can just go out and use the require function to grab.**

**But I'd like to make sure that I note, what I'm using in my app within this package.json file, that NPM looks at. So** **--save will save a reference in that package.json, to this dependency.**

I hit enter, and it goes and downloads from the Internet the package.

Two things happen.

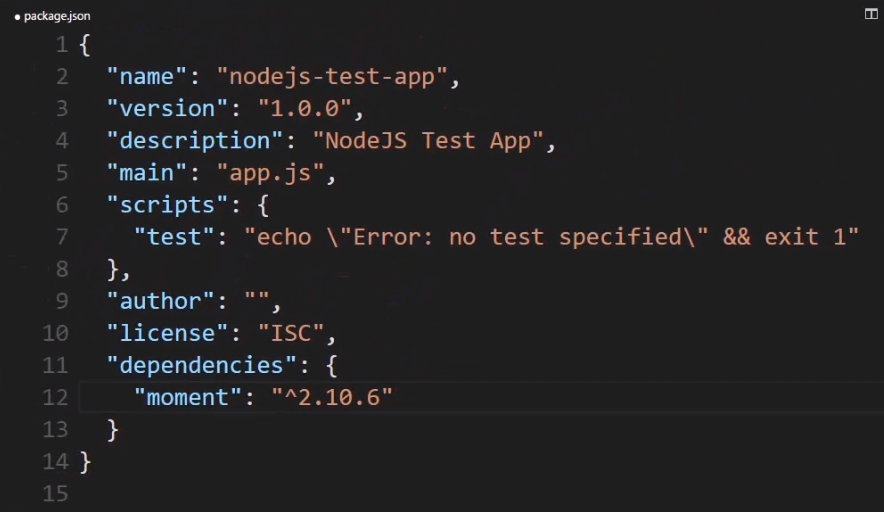
**One, a new folder is created, automatically, in my app called node\_modules. This is standard. This is how we get external modules and pull them into Node.**



in this node\_modules folder it downloaded moment. Which is just a bunch of Javascript files. It also has some documentation source code and other things.

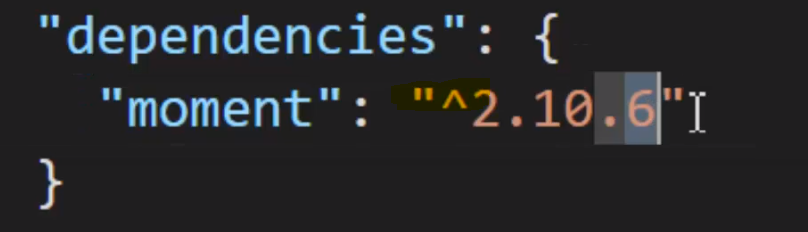
It's good to note that we're going to use this for Node, but, a lot of people use NPM even when they're not building a Node app. When they're just building a Javascript app that runs in the browser with maybe a different server technology. Because this is a great way to still keep track of what you're using in your app. So NPM has grown not just to be for Node, but really for anyone doing anything in Javascript on any platform. It's just a great way to manage things,

so you might instal Node on your computer, use it to download and use other code, but then end up actually using that code somewhere else. Not in Node itself. But not only was this downloaded, package.json, was automatically updated for me with another entry. Dependencies, and again this is just Json



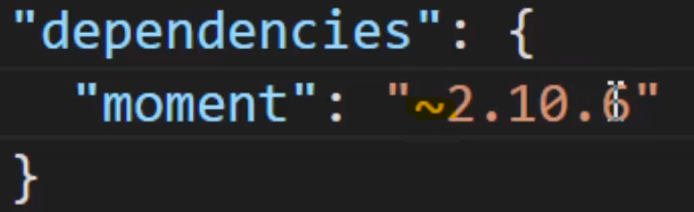
It said the name of the NPM package, and then a character, a caret, and then a version.

**The caret is very important.**

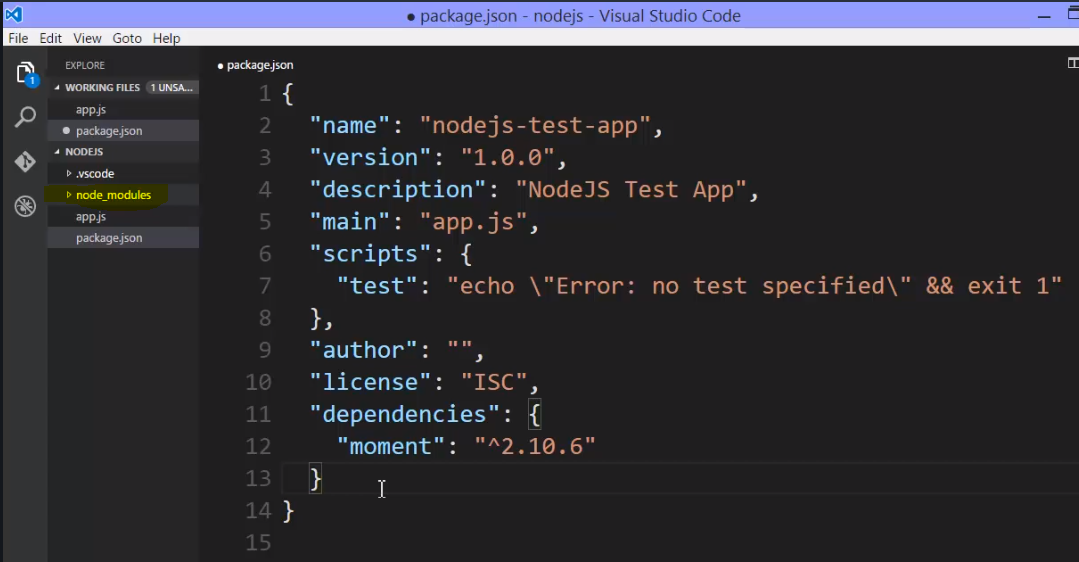


**This means that when we go to update(npm update) to the next version, if a new version of moment comes out, the caret means "NPM, it's okay to automatically update me." To anything within this major release, the two. So if, uh, 2.10.7 comes out, automatically update me. If a 2.11 comes out, automatically update me. But if it changes to a three, don't update me. So in other words, any minor or patch change is okay.**

**If you change this to a tilde,**



as sometimes you'll see, **that means only give me patches. Don't give me even minor updates. So in that way you can say how careful you want to be about updating to new versions of the packages that you're using.**



**But the really great thing, is that now I have this module available to me. And I have the dependency listed. That means I could go and give this package.json to someone else and they'd know what NPM packages are needed, to run this app.**

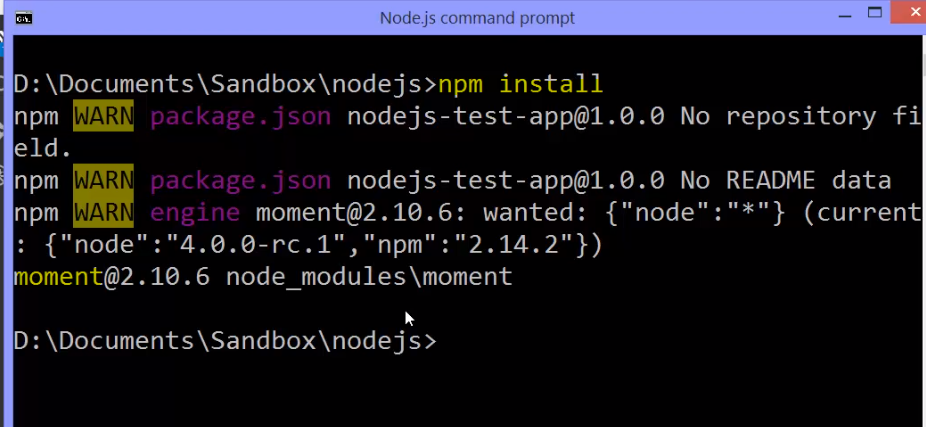
So for example

**if I just delete this Node module's folder. This is often what you'll get as a developer. Whatever Js files or the app, and the package.json. To save a lot of space in our, let's say our code repos if we're using GitHub or some other form of storing your code, we don't store the folder Node modules. Because we don't need to.**

**We're not editing that content. That's packages being downloaded and used. We only ever need to push package.json, into whatever repository of code we're using, to share with other coders, and all you have to do is here, in the command line, say npm instal, by itself.**

**That's it. And what that will do is it will look at package.json, and instal all of the dependencies, so it looks and it sees moment as a dependency, and it downloads it. And there it is again.**

**And it would have done that for all dependencies that I have listed. So that's a terrific way to keep track of, what do I need. What am I using in this app**



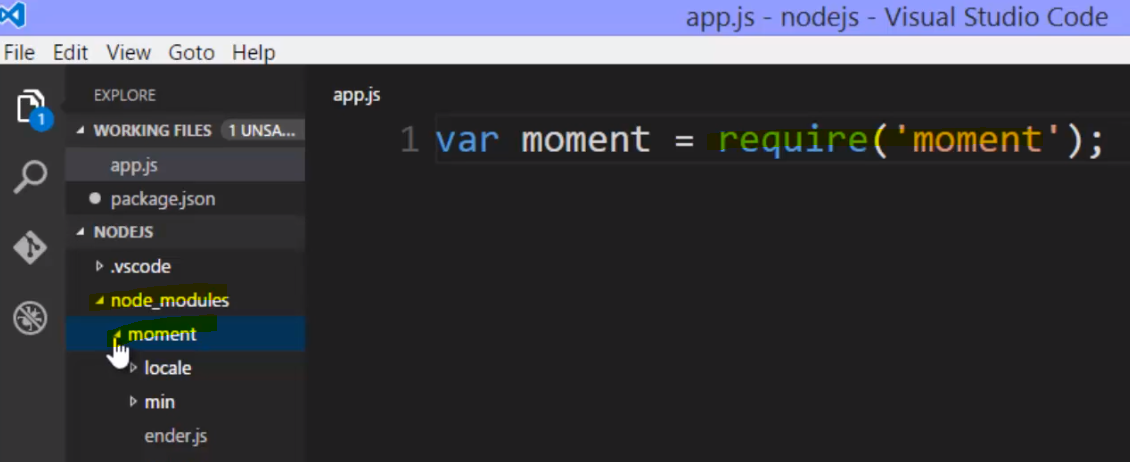
Now **how do we get access to this module?**

Well that's the great thing.

**It's a feature built into the require function. Remember how we said that require, has different capabilities as far as where it looks for things. If I do a ./ or ../, I'm saying a local module. Something in my folder structure.**

**If I just give it a name, it will first look for something in the Node core. Is something there in the Node core that matches but if it doesn't it does some other things. It looks other places and one of the places it looks, is for a node\_modules folder, in your app. And if there's node\_modules folder in your app, then it looks for that package there. It actually looks elsewhere as well(packages install globally in your computer appdata folder in c:/user)**

but we'll get there later.



**So we've already seen that we have the special package.json file, and isn't really used by Node itself but rather by NPM. The Node package manager. That programme on your computer. And when you install a package, downloading it from the NPM registry automatically, you can use --save in order to automatically save that reference to the package.json, or you could build this package.json yourself, or receive it from someone else and use NPM instal by itself. Just that phrase to automatically download all the dependencies. Now we called them dependencies because we use them in our app. My app won't run without this package.**

**However there are other dependencies. Dependencies in which I'm using only during development. My app will run without them but I need them during my development process. We've called those development dependencies, and there's a way to specify those using NPM and package.json as well.**

To see a really good example of this, I'll go to the command line and we'll do NPM install and then we'll get from the NPM registry a package called jasmine-node.

Now Jasmine is a tool to help us write automated test, automatically test my software as I build it.

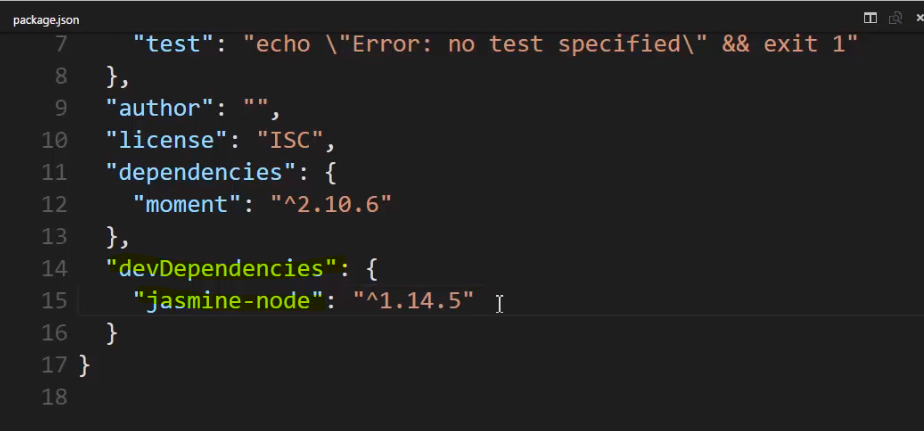
Now we're not going to focus on that in this course, but this is a really good example of a development dependency, a dev dependency.

**I don't need the test for the app to actually run, but I do need it and want to use it while building my app, and other developers who might work on this with me should also have this installed. So I can either add this directly to the package.json or say --save-dev.**

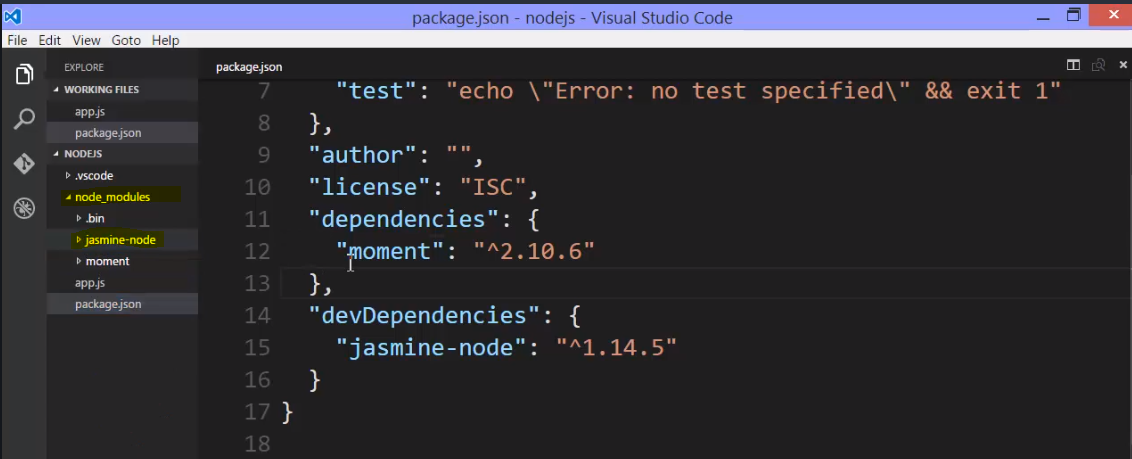
**That's the difference. Instead of --save, which saves it to a dependencies area in package.json, I can just say --save-dev.**



look at my package.json, I'll see that I have a dev dependencies entry now in the JSON file,



it's not something that's need for me to run the app, but it is still pulled into Node modules. The same location. There's a Jasmine-node folder now in node\_modules.



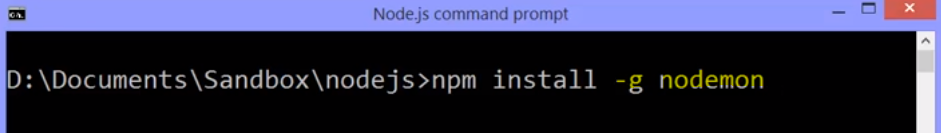
**I can make sure to tell the difference between what needs to go up on the server, for example, and what doesn't, in order for the app to actually run, and in some cases,** **when you deploy Node, when you actually push it up to a server, you don't push the Node modules at all.** In fact on the best servers you don't.

**There are utilities, server utilities that are already built in that will look at your dependencies, and essentially run the npm install after you push up your code.**

**So ultimately the server where you place your code will do the same thing we just did. Install all the Node modules when you first deploy your code. Push your code up to the server, and make sure that, it will have node\_modules folder what is needed.**

**It can ignore the dev dependencies and the server can choose to just install the dependencies.**

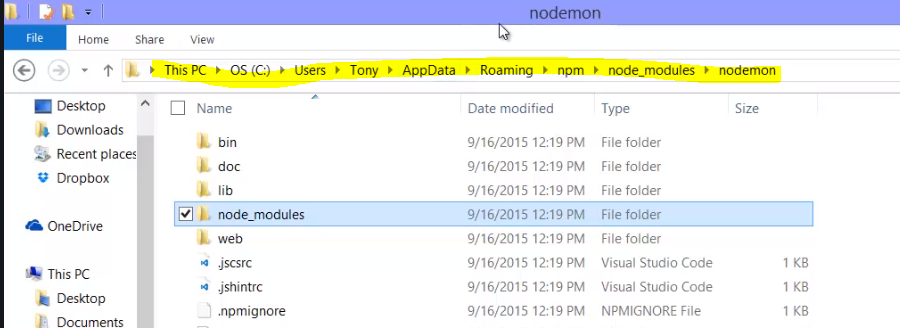
**There's one other kind of package that we might use. It's a package that again, doesn't have to be there for our app to run. And it is something that we're using for development. But it's something that we're using for development on all of our Node applications or might use on all of our Node applications. Not just this one particular.**

****

**-g, that means globally install this module. In other words, don't download into that node\_modules folder for this particular app, but rather put it in a location that NPM and Node can access it from anywhere on my computer from any app that I'm working on.**

**you might end up with some problems installing something globally from a permissions standpoint on your computer. Especially if you're using Linux or Mac, and there's information on that, that you can find on the NPM JS website, and around Googling on the internet for globally installed packages that cause some kind of permissions problem on your computer.**

now it's available globally. Where did it install it? Well that will depend on your setup and operating system. In my case, here on Windows, it installed it deep down inside the data specific to the user that I'm logged in as on my Windows computer, and there's a node\_modules folder found deep down in there, and I have nodemon installed.

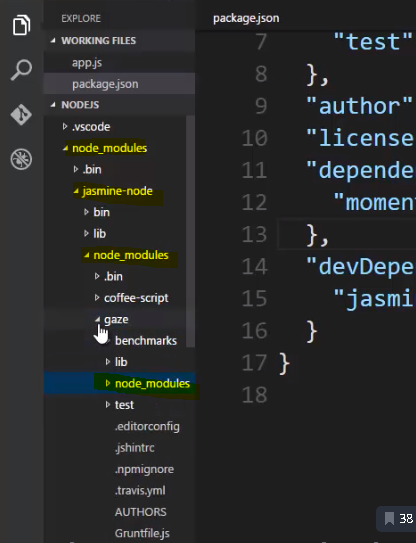


**So there are various places where a package, a module could be installed. You might have a local module, you might have one downloaded from NPM installed locally in a local node\_modules folder. You might have something installed in a global node\_modules folder. And all those places are places where Node can go and look for things.**

Now one other thing worth noticing is that nodemon itself has its own Node modules folder. And actually, if we were to look over in Jasmine-mode, it also does.

So let's pop back over there.

Node modules, Jasmine-node has a Node modules folder, and some of the dependencies that Jasmine has, has their own Node modules folder. Which are its dependencies. So we can see that whatever's installed into the node\_modules folder, comes along with its own dependencies for it to run. For Jasmine-node to run, it needs these, and for this to run, it needs these et cetera.



**This is actually something that is a bit criticized with NPM.**

Every package contains its own dependencies and then dependencies inside there contain their own dependencies. Things that they need to run. So a lot of times we'll have things duplicated around here within the node\_modules folder.

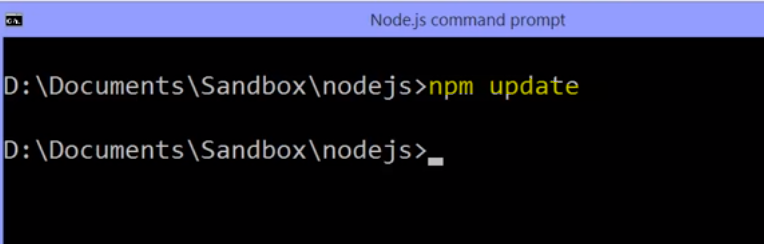
**That's a big reason why we don't push this whole thing to our server or into our code repositories 'cause this can get pretty big,**

but it's okay because it allows us to manage and deal with different dependencies using different diversions of the same dependency and a lot of problems that you can have with package management systems. So by simply saying each package will contain what it needs, it will be self-contained. And we avoid really having to think too hard about what packages depend on what other packages have the same dependencies.

We just say, the folder works, and that's why Node made that choice, and it works, and it works well.

What if the version of one of my dependencies is now old? What if there's a newer version out?

**Well the till day and the kerat take care of what version are allowed to be updated. I don't have to think too hard. I can just go to the command line and say NPM update.**



Run that, and it looks at all the dependencies that I have installed that is the code I have downloaded, and referenced in package.json. Checks for newer versions that match what I said were allowed up update, and then updates them. So if there had been a newer version of one of these, it would have downloaded it overwriting the version that's in node\_modules. So that's a very simple way to make sure as I'm developing, that I'm always using the latest version of other people's code that I'm using as part of my app.

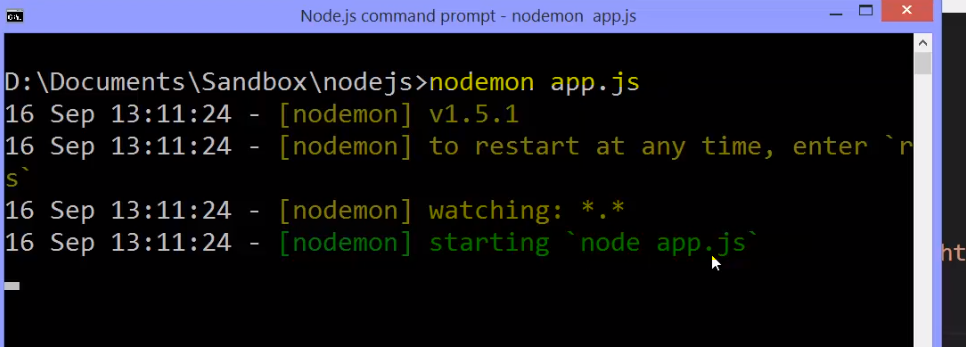
So let's use nodemon. We installed it globally. Which means that because, when I went and looked at the documentation for nodemon, after it's installed globally, I can just use it in the command line.

**The reason being that nodemon comes with what's called a command line interface utility.**

**Often you'll see that NPM packages will have something that lets you use them in the command line. And if they have it and you have it installed, then you should just be able to use it in the command line. Just like I did Node itself.**

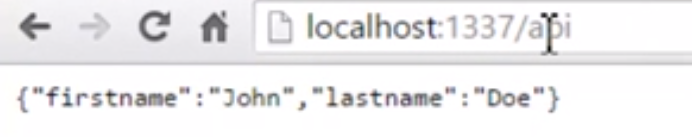
**In fact, I can just say nodemon app.js. And it starts the command Node app.js. But inside of it, nodemon or Node monitor watches the files in your app using built in functionality from the FS module within node, and it watches these files and waits for them to change,**

**and if any of the files change, nodemon automatically cancels Node and runs that Node command again. Pointing at the JavaScript file that I gave nodemon.**

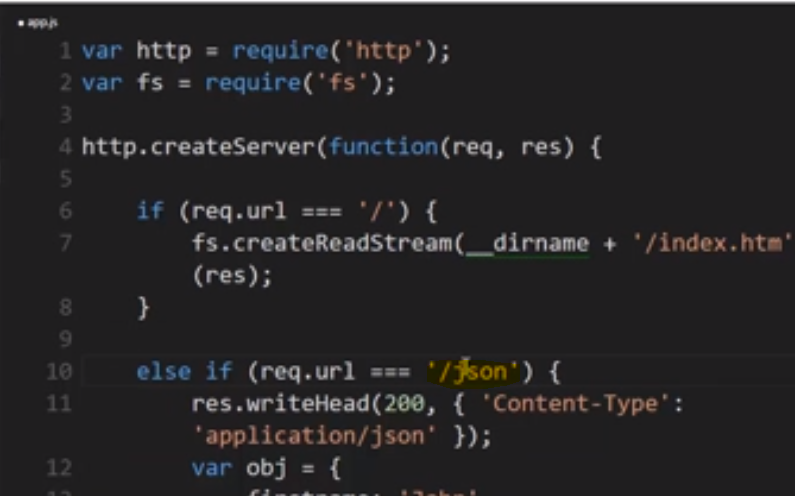


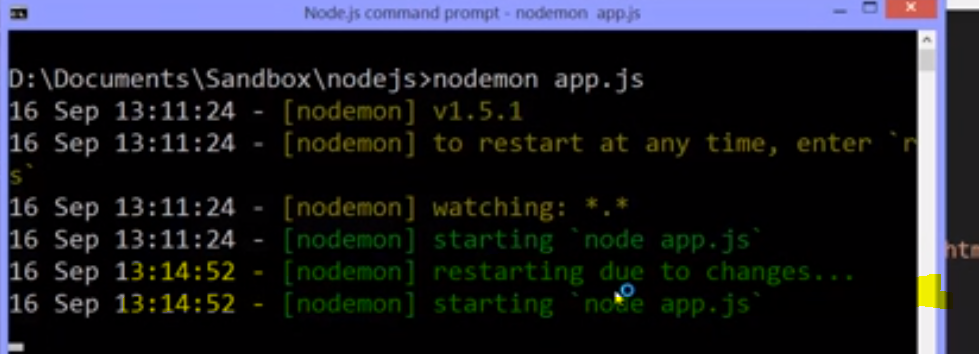
So I gave nodemon app.js. So it ran Node app.js, and if any of the files in this folder that I'm pointing at that I'm sitting on in the command line change, it will stop Node and then run Node app.js again. Exactly what we had to do this whole time.

**Lets see an example**

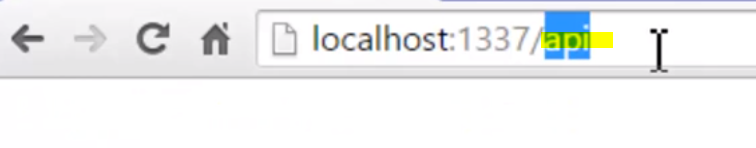


let's say example, I have /api and let's say I change this URL. As I'm doing this during development. So I changed this, let's say, to /json. I save the file and if I go back, we can see that nodemon said restarting due to changes, and then ran Node app.js again. I didn't have to do that manually.

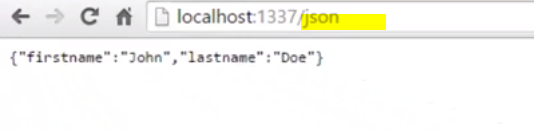




And now if I refresh, I get a 404,



and I can go to /json.

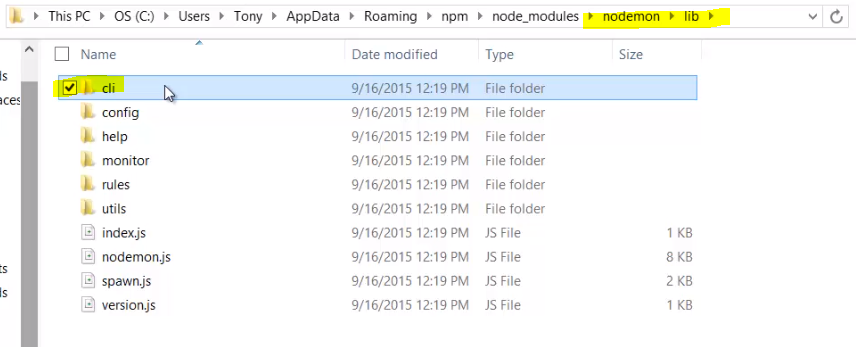


**So all nodemon did was automate whenever I save a file, stopping and restarting Node.**

**Now I don't have to use nodemon. It's not required for my app to run, and it really isn't a dev dependency because you can do development without it. But I've installed it globally so that I can use it when I want to on any of the apps that I want to,**

Again, you don't have to, you could manually stop and start Node if you like, but we're going to use it so we don't have to stop and think about it. And I like using it during real development.

quick note on the command line interface part of nodemon. You can find it in its source code. It's in a folder called cli. Cli stands for command line interface.



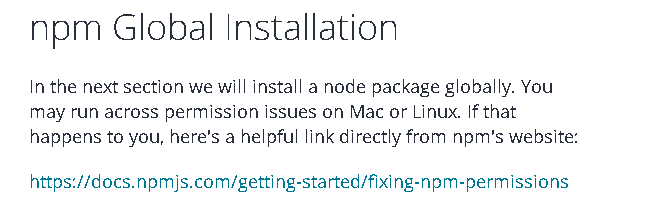
And you'll actually see this quite a bit throughout packages in the NPM registry. In fact, if you see a packaged name that has a -cli in it, that means, most likely, that it's a utility meant to be used in the command line. So just be aware of that because sometimes you'll have a utility with a particular name and then that same name with a -cle. Which means that you have a utility probably that you're going to use in code and then another version of that utility that's meant to be used just typing it into the command line.

**So this is the idea of NPM. Useful things, other people's code, available to me out there on the internet. And I can download it from the NPM registry. Register it in the package.json file as dependencies or dev dependencies. They're in the node\_modules folder. So I can grab them with the require function.**

**This is how Node caused such a huge explosion of open source code because it made it so easy to grab and use. I have access to this entire library of 10s of thousands of other people's code modules.**

**Pretty neat right?**

**Some of them are new, some of them are incredibly popular and have been around a long time. The idea is I can easily say this is what I'm using, instal it, update it, and just focus on building good software.**

**NPM Global installation** 

**Using other’s people code**

Just a quick note. It's important to remember that when using NPM and when downloading code from the NPM Registry, that you aren't just using other people's code. Anyone can push code into NPM. So it's very important to be sure that it's something that will work for you. The more popular a package, the more likely that it's good.

You can go in and look usually on GitHub to see issues and see what kind of problems people are having. And you shouldn't be afraid to open up the source code of what you're about to use, because you're going to be putting it inside your app, inside your software that either you're building or maybe someone's paying you to build. So don't be afraid of that, it's just JavaScript code.

But remember that it's just other people's code, and also that while semantic versioning is great, it's up to the people submitting the code to follow the semantic versioning rules properly. And not everyone always does. In fact, we all make mistakes. So be judicious in using things. Test your software well, make good choices, be careful as far as what you use before you build lots of code depending on a package that maybe ultimately you don't want to use after all.

Test out packages before you start integrating them heavily in your software. Alright, just a quick note but, all that said, **the package registry is unbelievable. It is one of the greatest things to ever happen in the open source community, and that's a great reason to be a fan of Node.js.**