

# COMP 1842

Week6 Intro to NPM

Matt Prichard

# Introduction

- Command line / terminal
- Node.js
- NPM
- SASS module

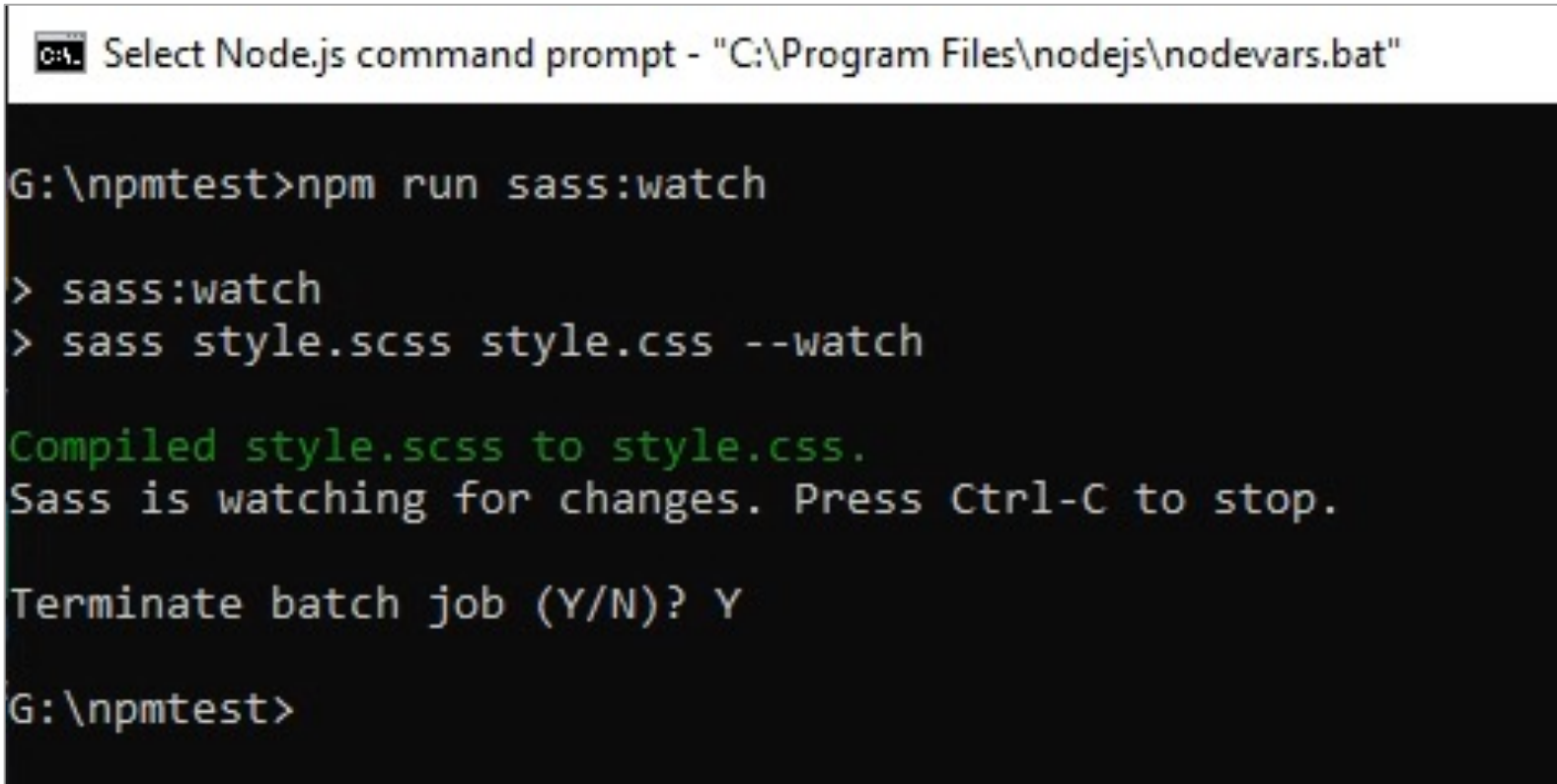


# Overview

- This is a beginners guide to npm or Node Package Manager.
- Modern [“back-of-the-front-end” development](#)—which npm is a part of—seems complex because it’s one name for lots of interconnected tools.
- *“...a front-of-the-front-end developer determines the look and feel of a button, while a back-of-the-front-end developer determines what happens when that button is clicked.”*
- npm ( always lower case) is a collection of technologies.

# Command line

I'm afraid we will be spending sometime here ☹



```
C:\> Select Node.js command prompt - "C:\Program Files\nodejs\nodevars.bat"

G:\npmtest>npm run sass:watch

> sass:watch
> sass style.scss style.css --watch

Compiled style.scss to style.css.
Sass is watching for changes. Press Ctrl-C to stop.

Terminate batch job (Y/N)? Y

G:\npmtest>
```

# terms

- npm, which is in a category known as “package management” software.
- And there’s [Node](#) itself, which is so tricky to explain succinctly It can be described by paraphrasing [Douglas Adams](#): *it’s a programming language that’s almost—but not quite—entirely like JavaScript.*

# npm manages project tools

When you type `npm install` into the command line it can install tools to help you do a wide variety of things in your project, like process your code (e.g., turn Sass code into CSS).

This list often includes tools like [Babel](#) (for compiling JavaScript), [Sass](#) (for compiling CSS), [webpack](#) (for asset bundling), [Vite](#) (for development servers and other tooling), [PostCSS](#) (for transforming one syntax into another); [Autoprefixer](#) (which can be a PostCSS plugin for CSS vendor prefixes); [TypeScript](#) (for additional JavaScript syntax); [ESlint](#) (for checking code quality); [Prettier](#) (for formatting code), and testing libraries like [Jest](#) or [Cypress](#).

# Command line – shudder☹

- The “command line” and the “terminal” are technically two different and distinct things, but are often used interchangeably. You may also hear the command line called a “shell” or see it abbreviated as “CLI” which is short for “command line interface.”
- Pedantic distinctions aside, the terms are often used to mean pretty much the same thing. So just to keep things as simple as possible, we’ll be using them interchangeably from here on.

# tip

- It's common convention to prefix commands with a \$ character—but it's a confusing convention. That's because there's no need to type it. It's literally not part of the command. Instead, \$ signifies a command that's meant to be run in a terminal.
- So here's the first rule to know about working with the command line: if you find yourself typing or copying an instruction that includes the \$ character, know that there is no need to include it in your work

```
## No need to copy the $  
$ npm run build
```



# What's it for?

The command line isn't exactly for writing code. As the name "command line" implies, it's for writing *commands*. But generally speaking, code in a terminal is written differently than it is in a code editor. Instead, you use the terminal to boss your computer around with commands you want it to run *immediately*.

# What's it for...2

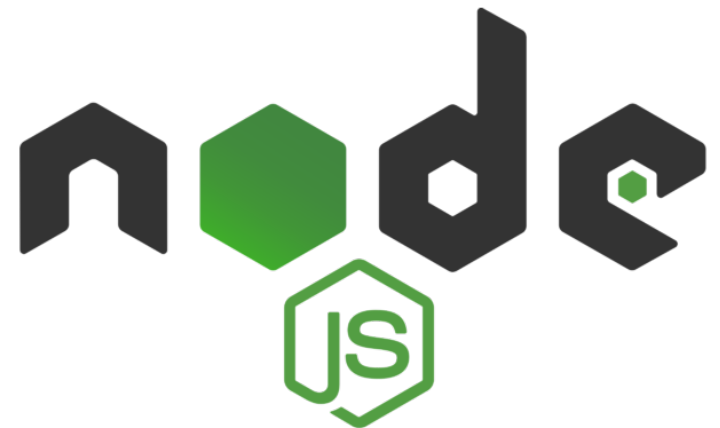
- It grants you god-like system privileges
- The command line is what is referred to as a “privileged environment.” it just means it's a place where there are very few restrictions on what you are allowed to do.
- Whatever command you type, as far as it's valid, is executed immediately and, often, irreversibly.
- It's capable of interacting with the hidden files your operating system tries to prevent you from editing.
- That makes it a bit risky, yes, but it also makes it very powerful, and the perfect choice for certain tasks and projects.

# Speed

- The command line enables npm to generate tons of files with incredible speed. The ability to run a single command that installs, updates, or deletes those files together in one fell swoop at high speed makes the terminal the fastest, most efficient tool for many jobs.
- it downloads and transmits files between computers—generally much, much faster than, say, using a browser to do it.

# Node and how it relates to npm

- Node is JavaScript, but as a server-side language.
- This is possible because of V8, Chromium's JavaScript engine, which can run on its own, outside the confines of the browser.
- Node and browser-based JavaScript can be very different, and have different capabilities, though both are JavaScript at their core.
- You don't need to know Node to use npm. (TFFT) ☺



# Node is JavaScript, but without the browser

- JavaScript doesn't need a browser anymore in order to run.
- Client-side languages (HTML, CSS, JavaScript),
- Server-side languages, like PHP, Ruby, or Python run purely on a server instead of the browser, and generally have much broader and more powerful capabilities.
- [Ryan Dahl](#) is credited with the invention of Node circa 2009 from a group who liked how fast JavaScript is (especially compared to PHP and Ruby), and they wanted to have JavaScript everywhere, not just in a browser.

# How Node works

JavaScript  
V8  
Engine



TECHBLUNT.COM

- Node is essentially JavaScript as a server-side language
- that runs outside of the browser.
- Under the bonnet each browser has its own individual JavaScript engine.
- The JavaScript engine in Chromium-based browsers is called V8 and is by far the most popular JavaScript engine, but now days the engine used in Chrome is a lot like the engine that runs in Firefox, which is a lot like Safari, and so on.

Side note: Firefox's JavaScript engine is named SpiderMonkey – cool 😊

# V8



- It turns out, you can take the JavaScript engine out of a browser, and with some modification, run it on its own—kind of like if you decided to pull the stereo out of a car, tinker a bit, and make it into a stereo system for your home instead. V8 can function perfectly fine as a standalone unit in any environment.
- In other words: V8 makes it possible to run JavaScript anywhere. That's why we have “Node” JavaScript and “browser-based” JavaScript.

# Differences

- Node and the JavaScript we're used to running in the browser are both similar and very different from each other.
- While both are JavaScript at their core, and while the language and syntax is the same, many staples of JavaScript in the browser (like the window or document, and even alert) are not present in a purely server-side Node environment.
- There is no window, of course, when the language is just running on its own, and not in a browser.
- We still have `console.log` though

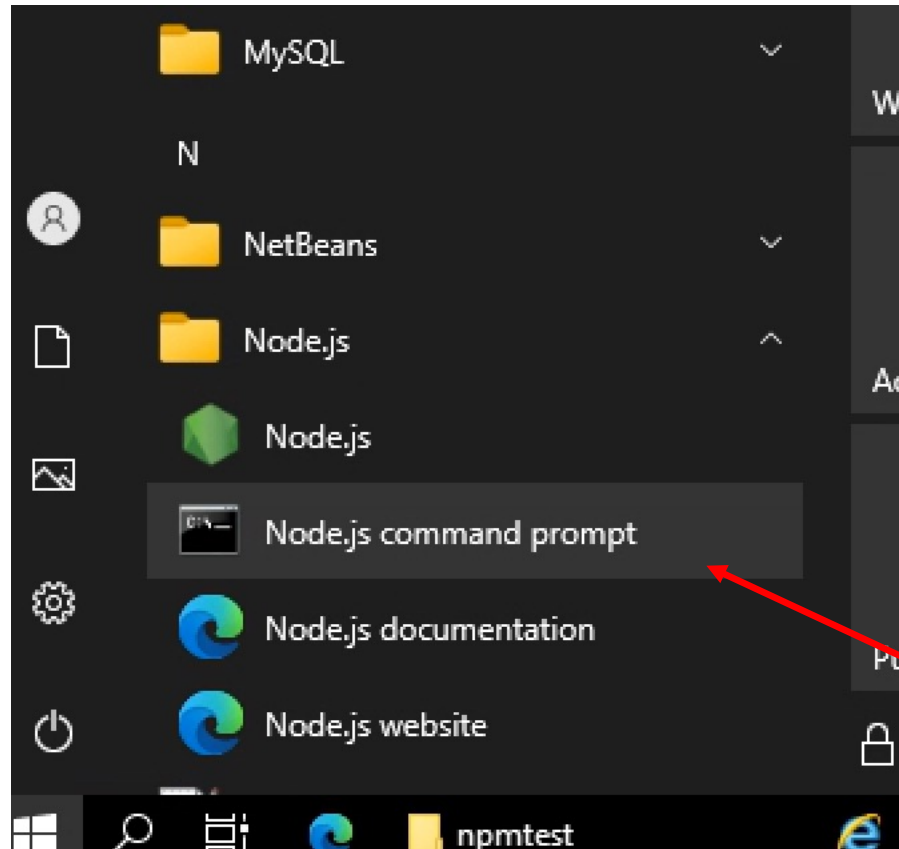


A side-by-side comparison of two musical instruments. On the left is a double bass, a large, dark brown wooden instrument with a curved body and a long neck. On the right is an electric guitar, a smaller, light-colored wooden instrument with a solid body and a long neck. The text "The same but different !" is overlaid in the center.

The same but different !

The same, but different (Photos: [Wikimedia Commons](#), [Unplash](#))

# Running Node locally



- For these examples I am running the files from my G Drive
- You can install Node and npm on your own machines, but I can't directly support that.\
- In the labs locate and open the Node.js command prompt

# Install check

```
Node.js command prompt - "C:\Program Files\nodejs\node.exe"
Your environment has been set up correctly.

G:\>node -v
v16.6.2

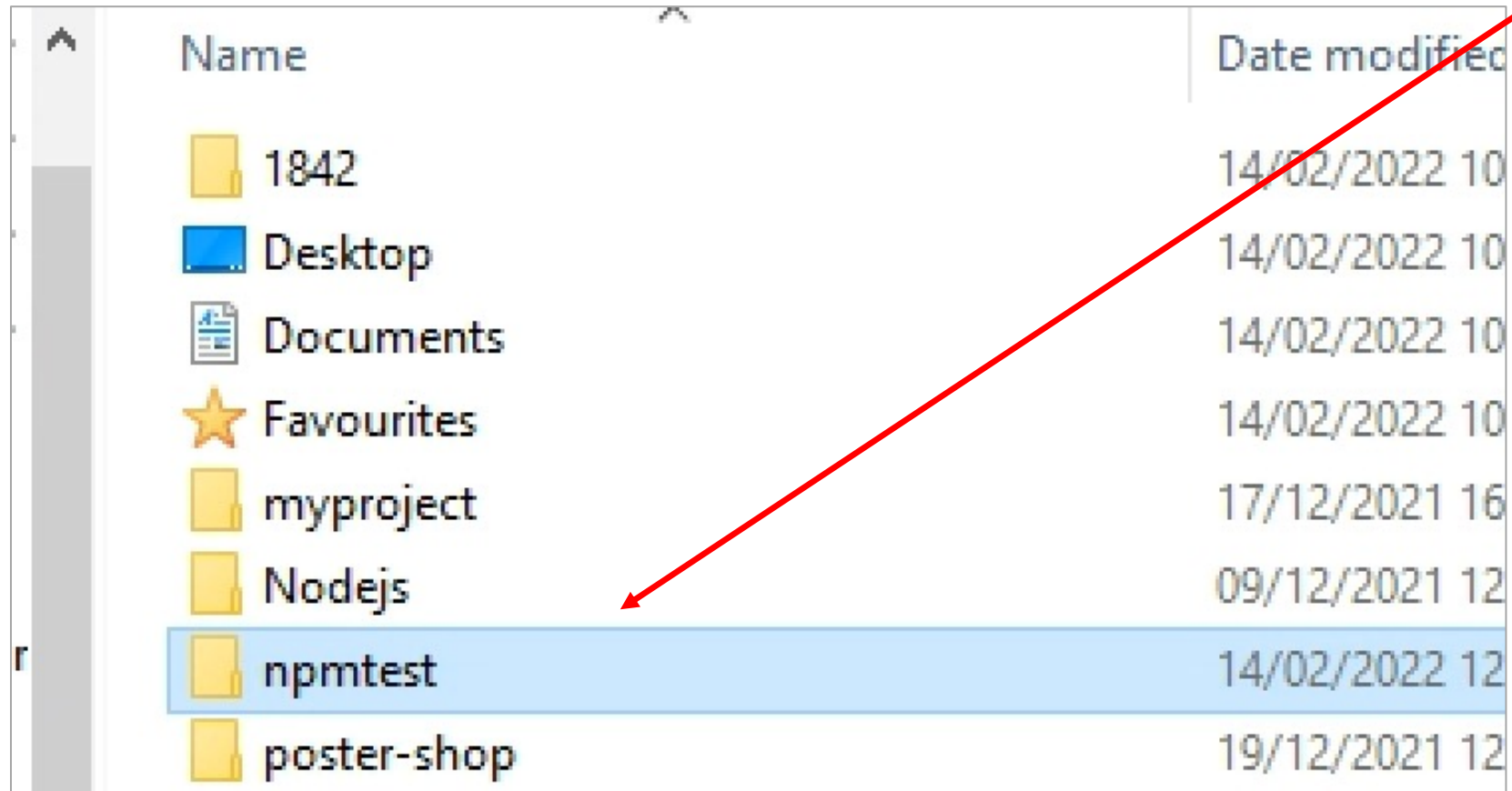
G:\>npm -v
7.20.3

G:\>_
```

Type `node -v` and press enter, this will give the current version of node that is installed, do the same with

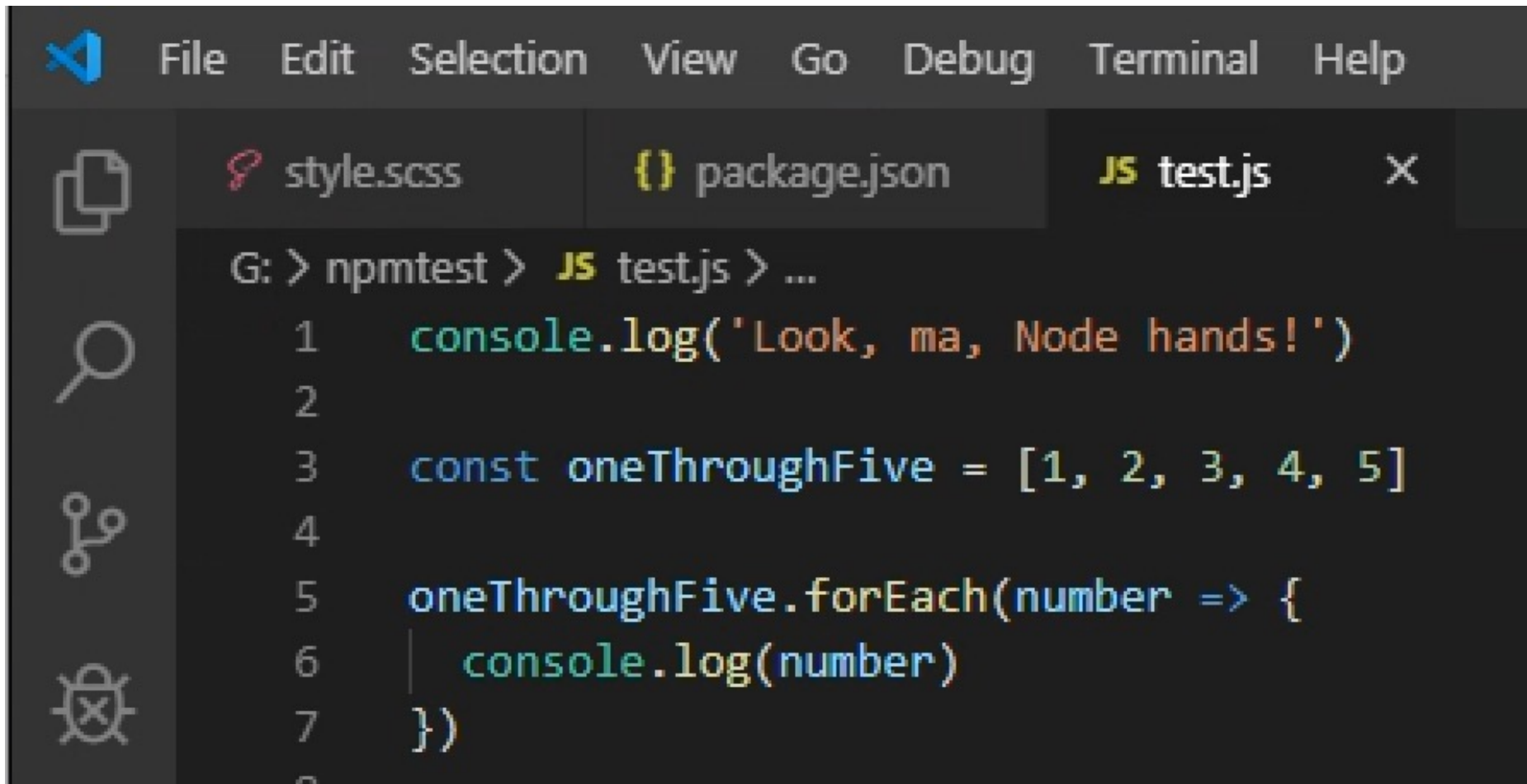
`npm -v`

# G Drive – new folder npmtest



Name	Date modified
1842	14/02/2022 10
Desktop	14/02/2022 10
Documents	14/02/2022 10
Favourites	14/02/2022 10
myproject	17/12/2021 16
Nodejs	09/12/2021 12
npmtest	14/02/2022 12
poster-shop	19/12/2021 12

# Create test.js and save to the new folder

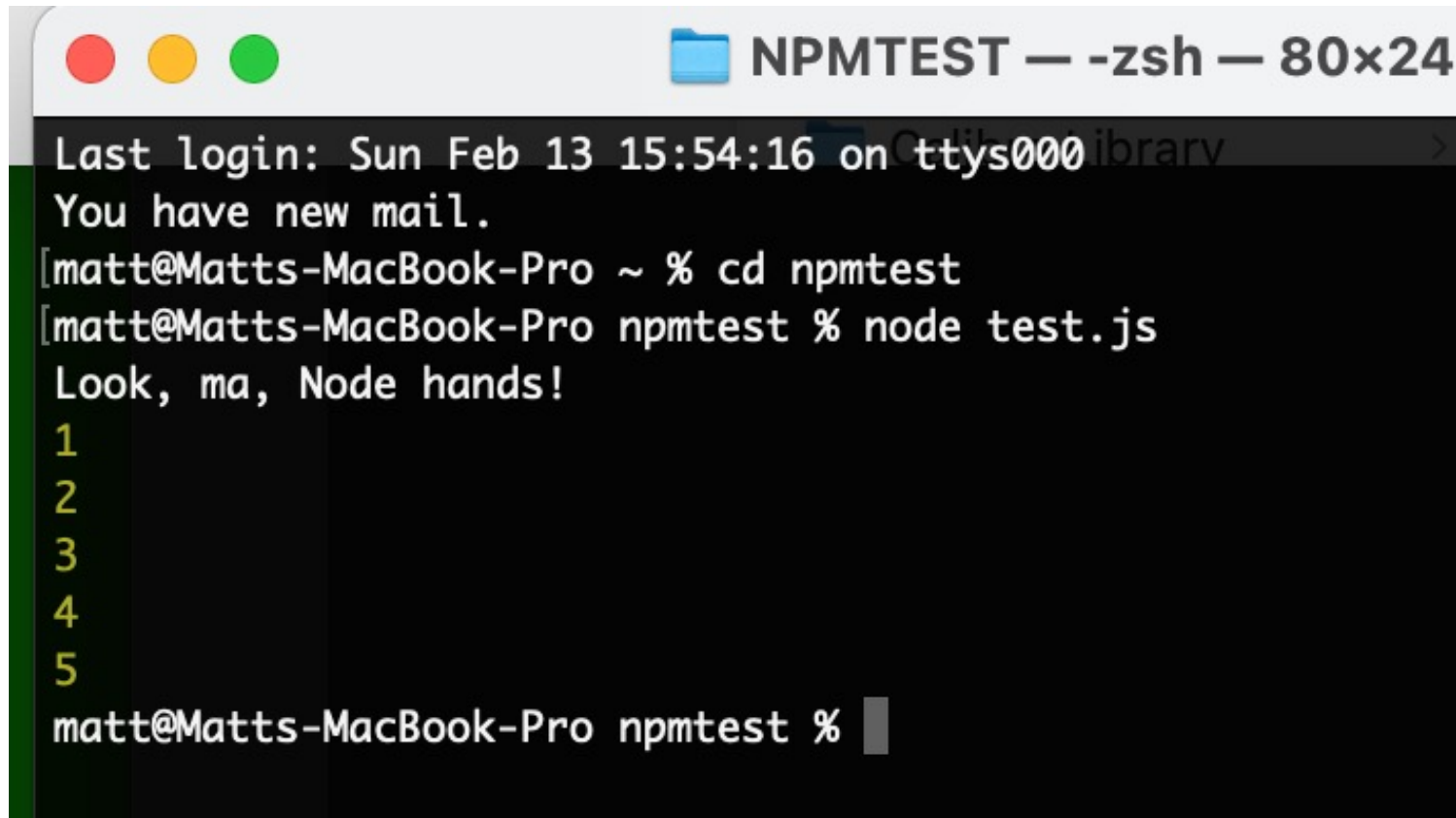


The screenshot shows the Visual Studio Code interface. The top menu bar includes File, Edit, Selection, View, Go, Debug, Terminal, and Help. The Explorer sidebar on the left shows a file named 'test.js' with a JavaScript icon. The editor window displays the following code:

```
G: > npmtest > JS test.js > ...  
1 console.log('Look, ma, Node hands!')  
2  
3 const oneThroughFive = [1, 2, 3, 4, 5]  
4  
5 oneThroughFive.forEach(number => {  
6   | console.log(number)  
7 })  
8
```

# Run node

Open the Node command prompt, navigate to where the file is (using `cd npmtest` or “change directory to npmtest”), and run `node test.js` to get the following output.

A screenshot of a macOS terminal window titled "NPMTEST — -zsh — 80x24". The window shows the following text: "Last login: Sun Feb 13 15:54:16 on ttys000", "You have new mail.", "[matt@Matts-MacBook-Pro ~ % cd npmtest", "[matt@Matts-MacBook-Pro npmtest % node test.js", "Look, ma, Node hands!", and a list of numbers 1 through 5 on the left side. The prompt "matt@Matts-MacBook-Pro npmtest %" is visible at the bottom with a cursor.

```
NPMTEST — -zsh — 80x24
Last login: Sun Feb 13 15:54:16 on ttys000
You have new mail.
[matt@Matts-MacBook-Pro ~ % cd npmtest
[matt@Matts-MacBook-Pro npmtest % node test.js
Look, ma, Node hands!
1
2
3
4
5
matt@Matts-MacBook-Pro npmtest %
```

# Installing at home

<https://docs.npmjs.com/downloading-and-installing-node-js-and-npm>

RTFM

# Installing the SASS package

Make sure you are still in the npmtest directory

- `cd ..` the parent of the current directory.
- `ls` used to list the contents of a directory
- `cd npmtest` change directory to npmtest

Once you are in the 'npmtest' directory run this :

```
npm install sass
```



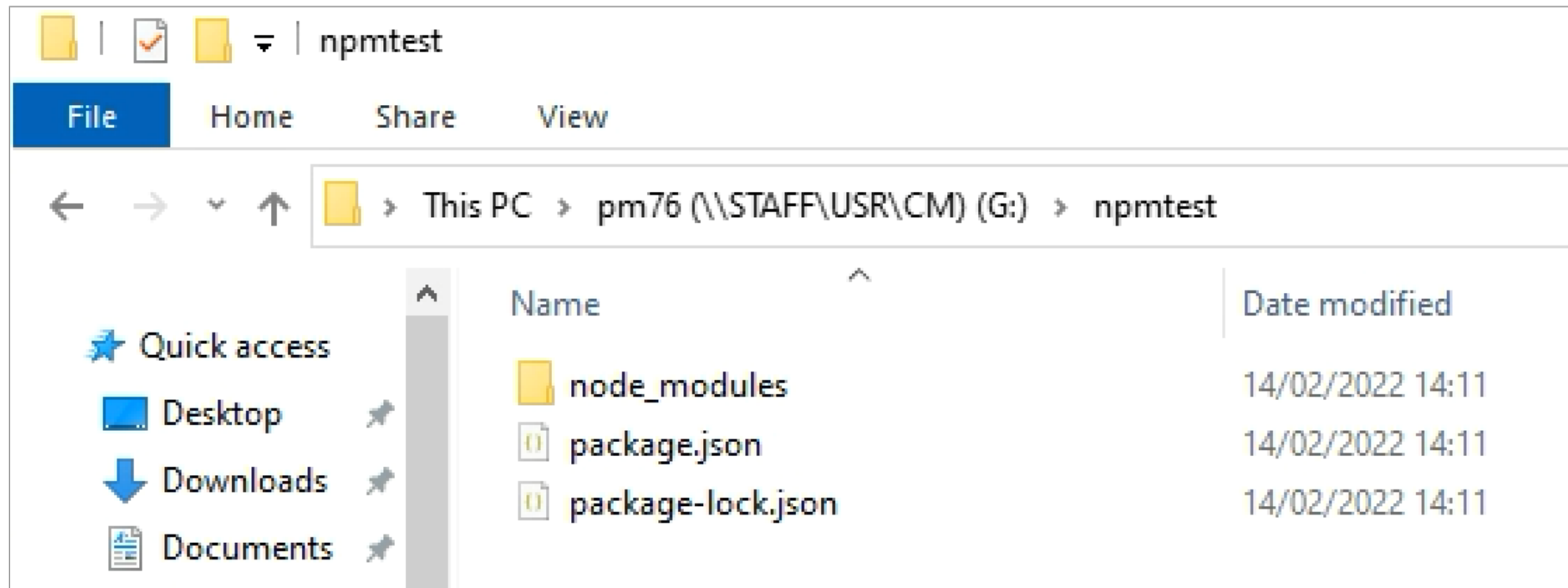
You should see something like this in the command prompt

```
Node.js command prompt - "C:\Program Files\nodejs\nodevars.bat"
Your environment has been set up for using Node.js 16.6.2 (32-bit)
G:\>cd npmtest
G:\npmtest>npm install sass
added 17 packages, and audited 18 packages in 2s
2 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
G:\npmtest>_
```

Open your npmtest folder and you should have this

A folder and 2 JSON files

This is the SASS package that npm installed



# What happens when you install a package

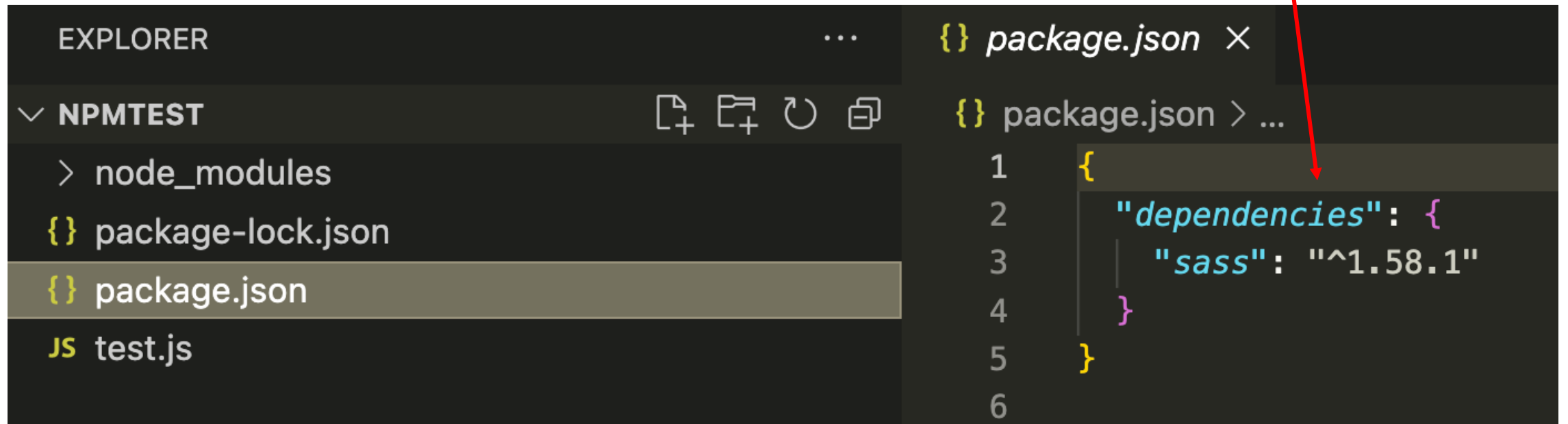
- When you install (or uninstall, or update) a package, npm does most, if not all, of the following four things:
  1. Updates the package.json file in your project, if needed;
  2. updates the package-lock.json file (called the “lockfile”) that contains all of the technical specifics;
  3. installs the actual package files—and any other packages the original package might depend on (inside of the node\_modules folder); and
  4. runs an audit of the installed packages.

# package.json and package-lock.json

- These two JSON files work together to ensure an accurate record of all the dependencies in your project (and all of their dependencies, and so on). The difference is a little technical, but loosely explained: the lockfile is the in-depth, precise snapshot of the project's dependency tree, and package.json is a high level overview, which can also contain other things. The main packages you install may be listed in package.json, but package-lock.json is where the entire dependency tree is tracked.
- The lockfile is also never supposed to be updated by hand; only by npm. So be sure to avoid mistaking the lockfile with the package.json file.

# Package.json

We can see the dependency “sass” and a version number



The image shows a screenshot of the Visual Studio Code editor. On the left, the 'EXPLORER' sidebar shows a project named 'NPMTEST' with files: 'node\_modules', 'package-lock.json', 'package.json' (selected), and 'test.js'. On the right, the 'package.json' file is open, showing a JSON object with a 'dependencies' section. A red arrow points from the text above to the 'sass' dependency entry. The code is as follows:

```
1 {  
2   "dependencies": {  
3     "sass": "^1.58.1"  
4   }  
5 }  
6
```

# node\_modules

- node\_modules is where all the actual package code lives; it's where your installed Node packages and all the stuff that makes them run actually get installed. If you open up the folder right now as you're following along, you'll find a sass folder, but alongside several other folders as well.
- The reason for the additional folders is that when you install a package, it may need other packages to run properly (as Sass clearly does). So, npm automatically does the hard work of finding and installing all of those dependencies as well.
- We don't need to go in here really 😊 (TFFT) again.

# npm commands

- reopen the package.json file in your npmtest folder, and you won't see much right now; just a dependencies property, with only one dependency so far.
- One of the most interesting bits is an optional, but extremely useful property called scripts. The scripts object in your package.json file allows you to create commands you can run in that project to handle various tasks for you, either as a one-shot, or a continuously running process.
- We don't have any scripts to run yet, but let's fix that!



```
{ } package.json ×  
{ } package.json > ...  
1   {  
2     "dependencies": {  
3       "sass": "^1.58.1"  
4     }  
5   }  
6
```

# scripts

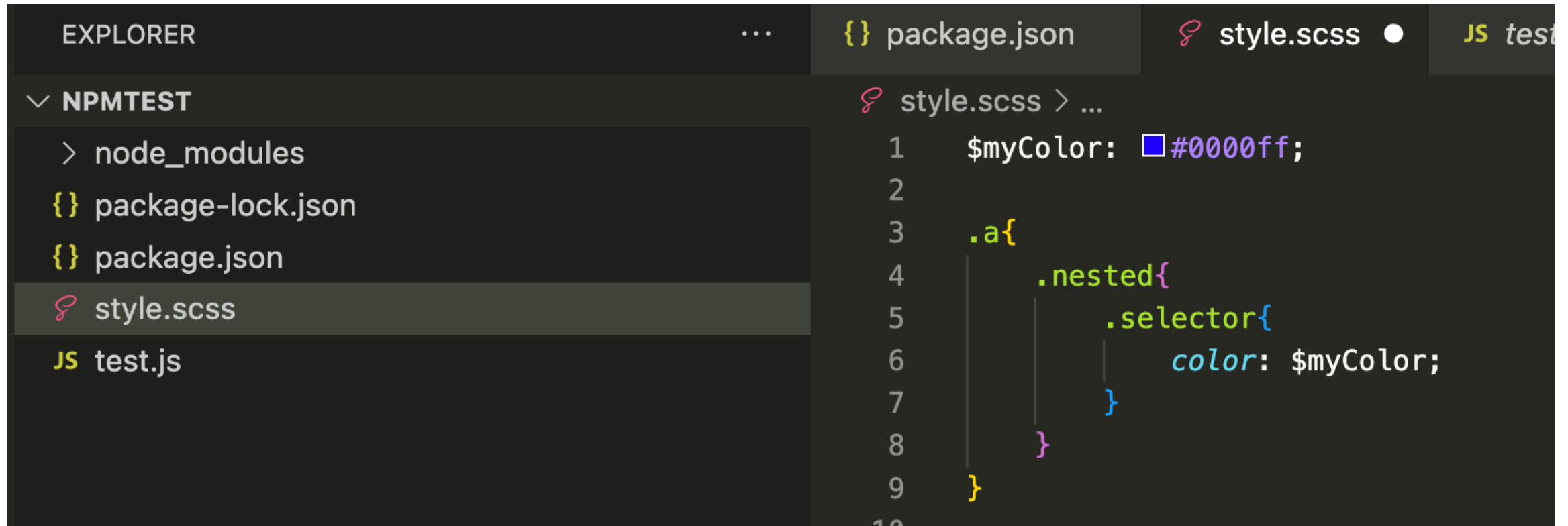
- Inside of the scripts section of the package.json file, we have access to all of our installed packages. Even though we are unable to simply type sass commands in the terminal right now, we can run sass commands as part of an npm script.
- Start by adding this block of code into your package.json file, right after the opening { curly brace:
- This gives us access to an npm run sass:build script, which will compile Sass into CSS for us.

```
{ } package.json > ...  
1   {  
    |   Debug  
2   |   "scripts": {  
3   |     "sass:build": "sass style.scss style.css"  
4   |   },  
5   |   "dependencies": {  
6   |     "sass": "^1.58.1"  
    |   }  
  }
```



# Create a simple sass file

Copy this into a document and save as style.scss into the npmtest folder



The screenshot shows the Visual Studio Code interface. On the left, the 'EXPLORER' sidebar displays the file structure of a project named 'NPMTEST'. The files listed are 'node\_modules', 'package-lock.json', 'package.json', 'style.scss' (which is selected and highlighted), and 'test.js'. On the right, the 'style.scss' file is open in the editor, showing the following SCSS code:

```
1  $myColor: #0000ff;
2
3  .a{
4      .nested{
5          .selector{
6              color: $myColor;
7          }
8      }
9  }
```

# Run the new command

```
npm run sass:build
```

Terminal

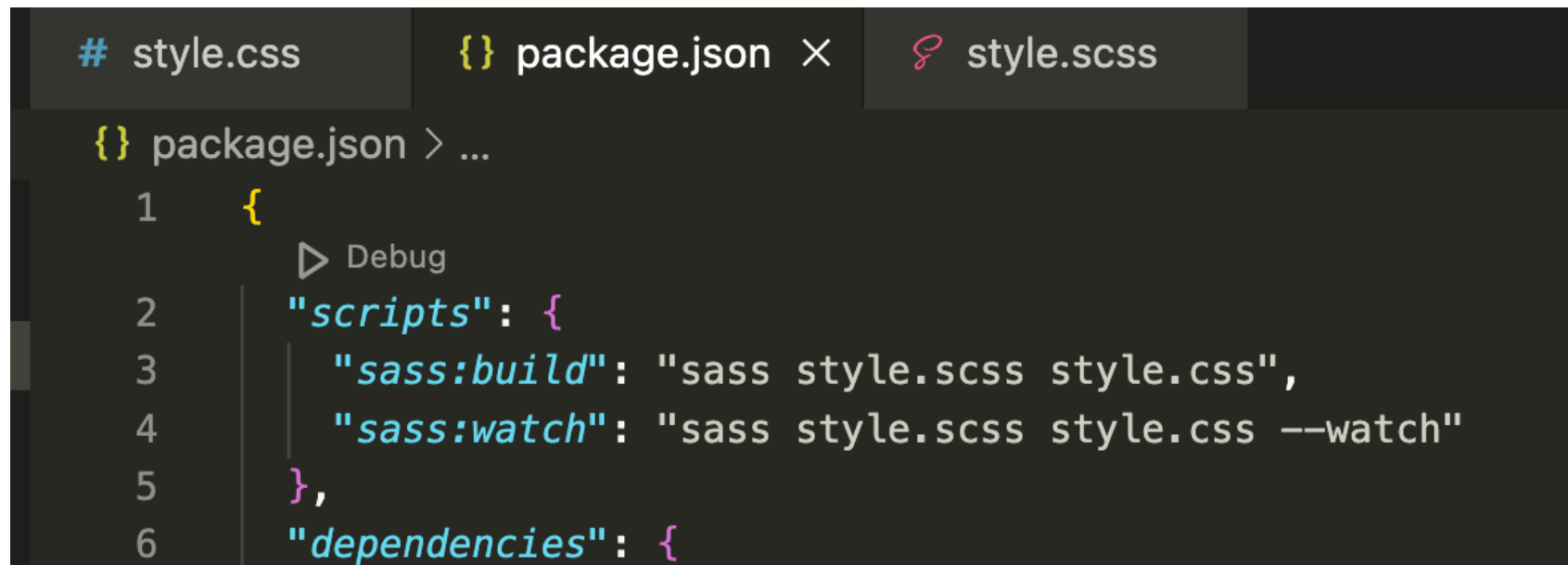
Once this task has completed, you should see two new files appear almost instantly in your project folder: `style.css` and `style.css.map`.



# Creating a development-only command

We'll get tired of running that command over and over as we're developing. So, let's set up a second command that tells Sass to watch the file for us, and re-compile it automatically any time we save changes!

Add line 4 to package.json, notice the comma on line 3.

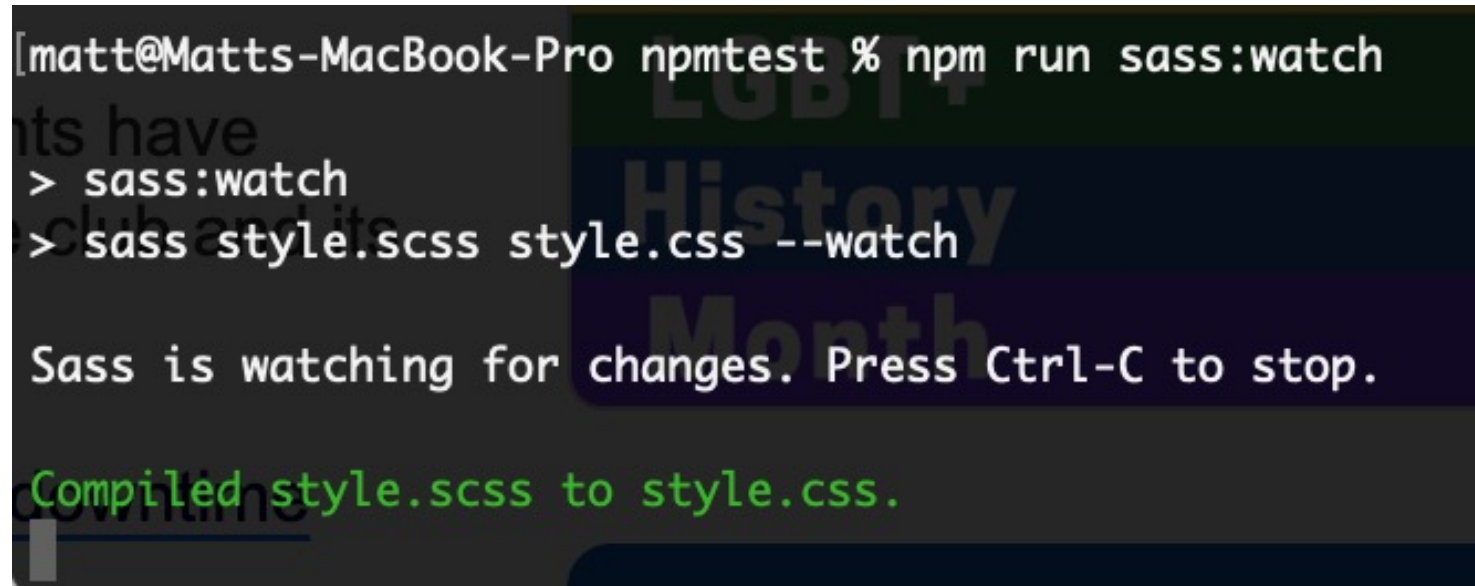


The screenshot shows a code editor with three tabs at the top: `# style.css`, `{ } package.json` (which is active), and `🔗 style.scss`. The `package.json` file is open, showing a JSON object with a `scripts` property. The code is as follows:

```
{ } package.json > ...  
1  {  
    ▶ Debug  
2    "scripts": {  
3      "sass:build": "sass style.scss style.css",  
4      "sass:watch": "sass style.scss style.css --watch"  
5    },  
6    "dependencies": {
```

# Run sass:watch

In the command prompt type `npm run sass:watch`

A terminal window screenshot with a dark background. The prompt is [matt@Matts-MacBook-Pro npmtest %]. The user enters 'npm run sass:watch'. The prompt changes to '>'. The user enters 'sass:watch'. The prompt changes to '>'. The user enters 'sass style.scss style.css --watch'. The terminal outputs 'Sass is watching for changes. Press Ctrl-C to stop.' followed by 'Compiled style.scss to style.css.' in green text. There is a faint 'LGBT+ History Month' watermark in the background.

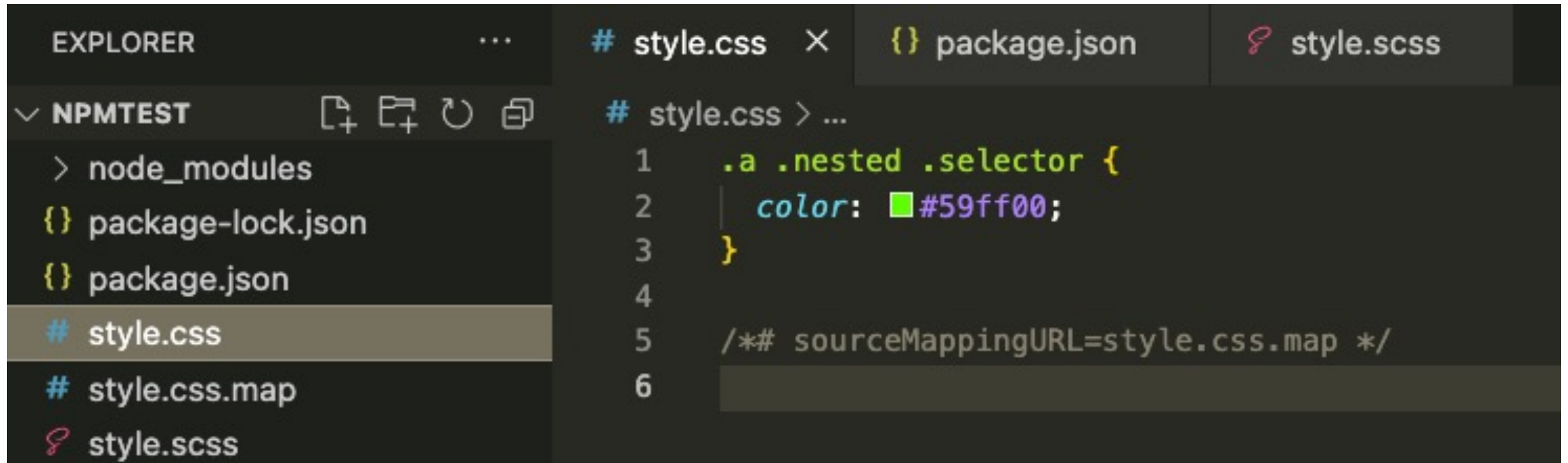
```
[matt@Matts-MacBook-Pro npmtest % npm run sass:watch
> sass:watch
> sass style.scss style.css --watch

Sass is watching for changes. Press Ctrl-C to stop.

Compiled style.scss to style.css.
```

If you open your style.scss file now, make a change, and save it, you should see a message automatically pop up in the terminal confirming that the Sass has re-compiled into CSS:

Here we can see the automatically updated css file,  
I changed it to green



```
EXPLORER
  NPMTEST
    > node_modules
    {} package-lock.json
    {} package.json
    # style.css
    # style.css.map
    style.scss

# style.css
1  .a .nested .selector {
2    color: #59ff00;
3  }
4
5  /*# sourceMappingURL=style.css.map */
6
```

# Summary

We have looked at using npm to install a package and edit the the package to automate a task for us.

We have seen how to use node js to run some basic JS outside of the browser.

# Labs

Work through from slide 18 onwards checking it all works and you understand what is going on.

We will be using npm throughout the rest of the module.

Try installing node.js and npm on your home machine if you want, here's where to begin

<https://docs.npmjs.com/downloading-and-installing-node-js-and-npm>

# Sources

<https://css-tricks.com/a-complete-beginners-guide-to-npm/>

<https://docs.npmjs.com/downloading-and-installing-node-js-and-npm>

<https://www.w3schools.com/nodejs/default.asp>



