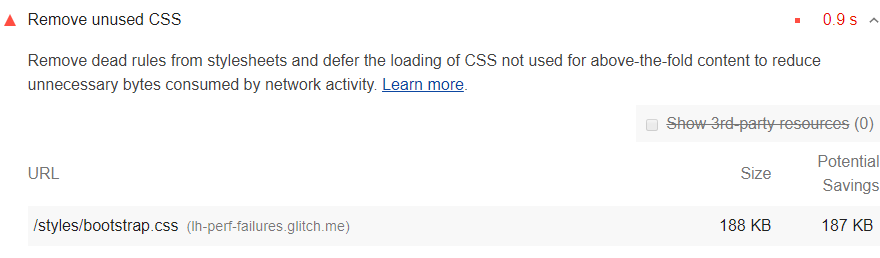
**Remove unused CSS**

May 2, 2019 • Updated May 29, 2020

Appears in: [Performance audits](https://web.dev/lighthouse-performance)

The Opportunities section of your Lighthouse report lists all stylesheets with unused CSS with a potential savings of 2 KiB or more. Remove the unused CSS to reduce unnecessary bytes consumed by network activity:



**How unused CSS slows down performance** [**#**](https://web.dev/unused-css-rules/?utm_source=lighthouse&utm_medium=devtools#how-unused-css-slows-down-performance)

Using a <link> tag is a common way to add styles to a page:

<!doctype html>  
<html>  
 <head>  
 <link href="main.css" rel="stylesheet">  
 ...

The main.css file that the browser downloads is known as an external stylesheet, because it's stored separately from the HTML that uses it.

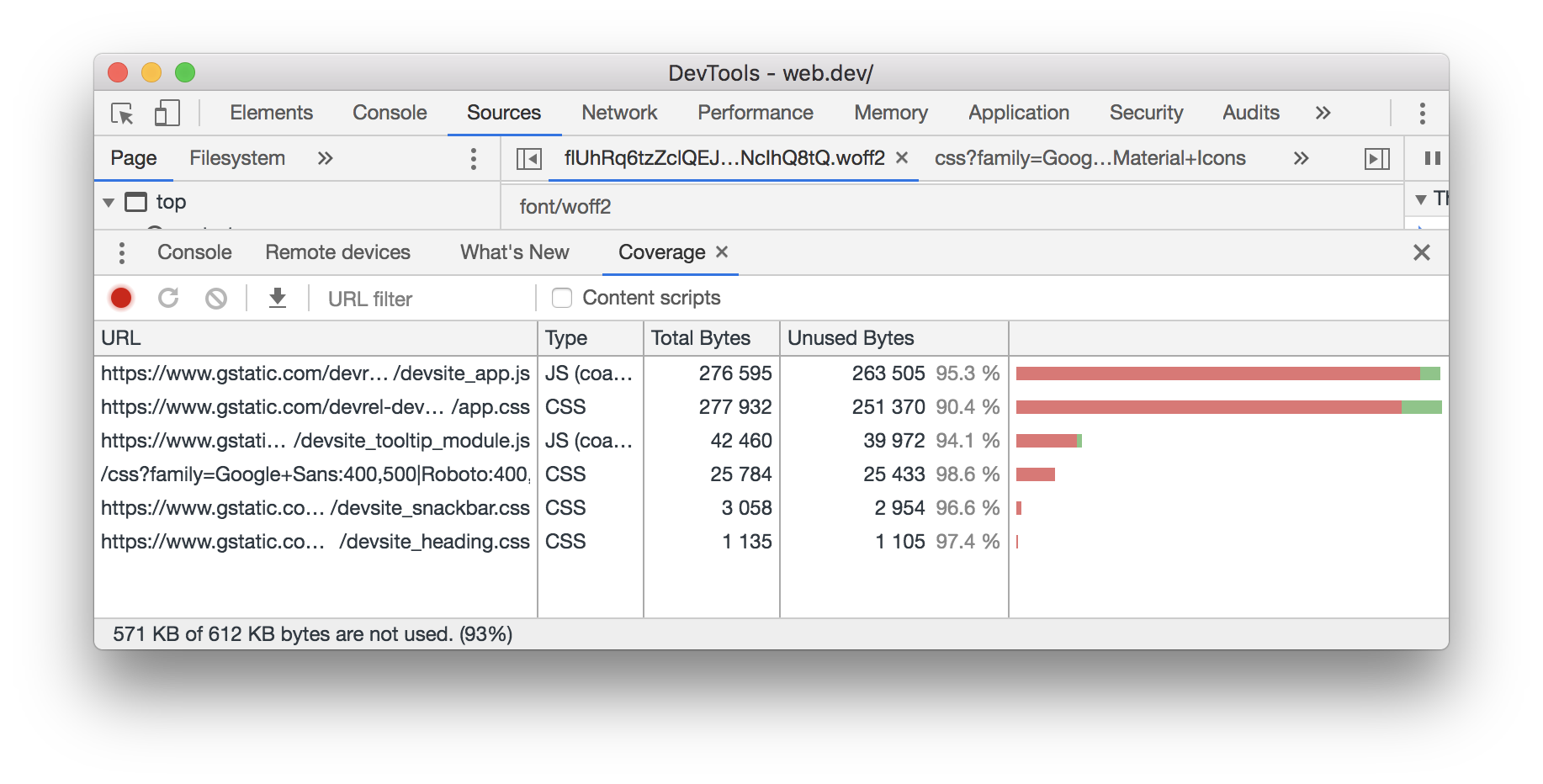
By default, a browser must download, parse, and process all external stylesheets that it encounters before it can display, or render, any content to a user's screen. It wouldn't make sense for a browser to attempt to display content before the stylesheets have been processed, because the stylesheets may contain rules that affect the styling of the page.

Each external stylesheet must be downloaded from the network. These extra network trips can significantly increase the time that users must wait before they see any content on their screens.

Unused CSS also slows down a browser's construction of the [render tree](https://developers.google.com/web/fundamentals/performance/critical-rendering-path/render-tree-construction). The render tree is like the DOM tree, except that it also includes the styles for each node. To construct the render tree, a browser must walk the entire DOM tree, and check which CSS rules apply to each node. The more unused CSS there is, the more time that a browser might potentially need to spend calculating the styles for each node.

**How to detect unused CSS** [**#**](https://web.dev/unused-css-rules/?utm_source=lighthouse&utm_medium=devtools#coverage)

The Coverage tab of Chrome DevTools can help you discover critical and uncritical CSS. See [View used and unused CSS with the Coverage tab](https://developers.google.com/web/tools/chrome-devtools/css/reference#coverage).

Chrome DevTools: Coverage tab.

You can also extract this information from Puppeteer. See [page.coverage](https://github.com/GoogleChrome/puppeteer/blob/master/docs/api.md#pagecoverage).

**Inline critical CSS and defer non-critical CSS** [**#**](https://web.dev/unused-css-rules/?utm_source=lighthouse&utm_medium=devtools#inline-critical-css-and-defer-non-critical-css)

Similar to inlining code in a <script> tag, inline critical styles required for the first paint inside a <style> block at the head of the HTML page. Then load the rest of the styles asynchronously using the preload link.

Consider automating the process of extracting and inlining "Above the Fold" CSS using the [Critical tool](https://github.com/addyosmani/critical/blob/master/README.md).

Learn more in [Defer non-critical CSS](https://web.dev/defer-non-critical-css).

**Stack-specific guidance** [**#**](https://web.dev/unused-css-rules/?utm_source=lighthouse&utm_medium=devtools#stack-specific-guidance)

**Drupal** [**#**](https://web.dev/unused-css-rules/?utm_source=lighthouse&utm_medium=devtools#drupal)

Consider removing unused CSS rules and only attach the needed Drupal libraries to the relevant page or component in a page. See the [Defining a library](https://www.drupal.org/docs/8/creating-custom-modules/adding-stylesheets-css-and-javascript-js-to-a-drupal-8-module#library) for details.

**Joomla** [**#**](https://web.dev/unused-css-rules/?utm_source=lighthouse&utm_medium=devtools#joomla)

Consider reducing, or switching, the number of [Joomla extensions](https://extensions.joomla.org/) loading unused CSS in your page.

**WordPress** [**#**](https://web.dev/unused-css-rules/?utm_source=lighthouse&utm_medium=devtools#wordpress)

Consider reducing, or switching, the number of [WordPress plugins](https://wordpress.org/plugins/) loading unused CSS in your page.

# Find Unused JavaScript And CSS With The Coverage Tab

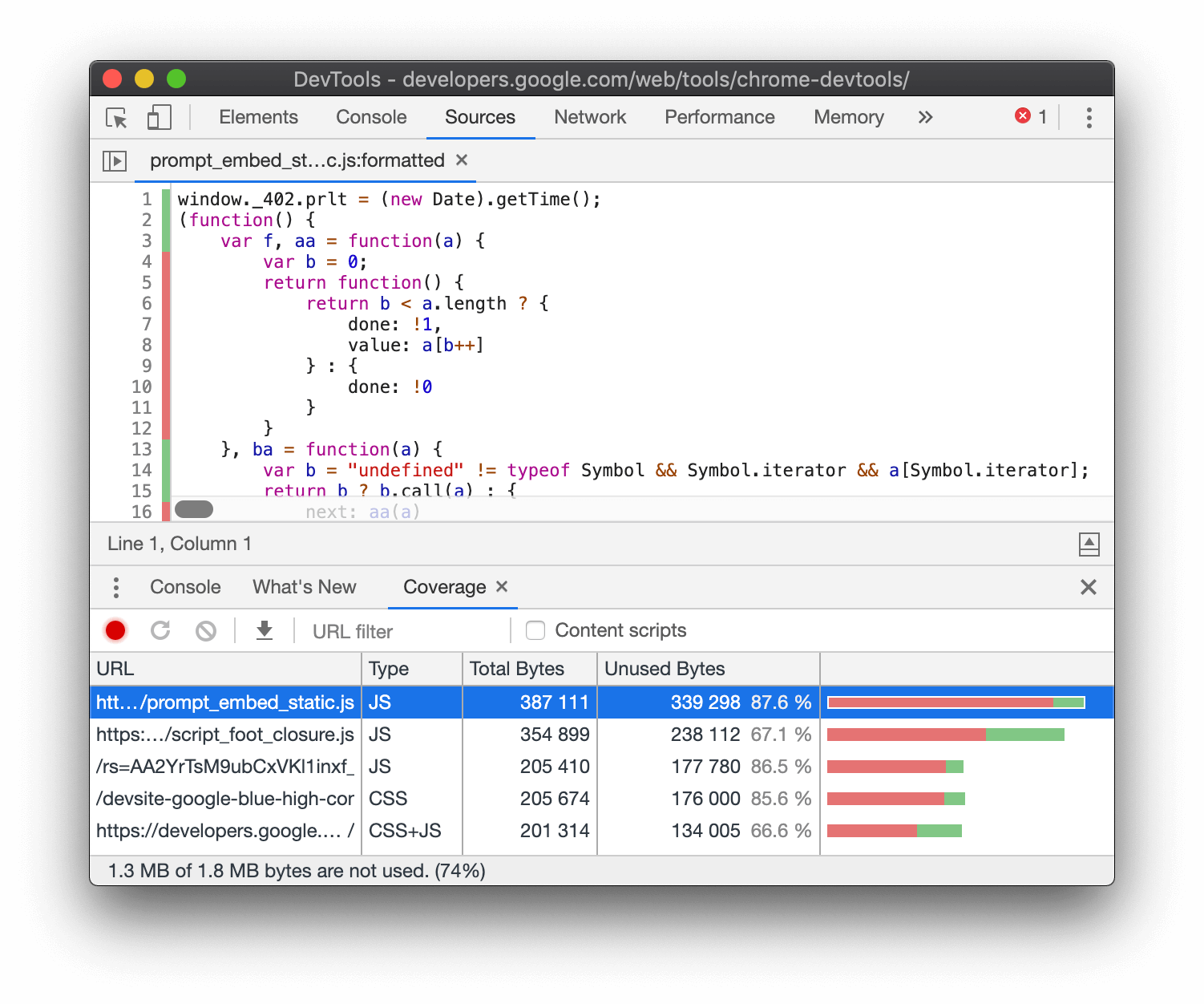
Published on Tuesday, July 9, 2019 • Updated on Friday, July 10, 2020

[](https://developer.chrome.com/authors/kaycebasques/)

[Kayce Basques](https://developer.chrome.com/authors/kaycebasques/)

Technically, I'm a writer

The Coverage tab in Chrome DevTools can help you find unused JavaScript and CSS code. Removing unused code can speed up your page load and save your mobile users cellular data.

**Figure 1**. Analyzing code coverage.

Caution

A word of warning. Finding unused code is relatively easy. But refactoring a codebase so that each page only ships the JavaScript and CSS that it needs can be difficult. This guide doesn't cover how to refactor a codebase to avoid unused code because these refactors depend highly on your technology stack.

## [#](https://developer.chrome.com/docs/devtools/coverage/#overview) Overview

Shipping unused JavaScript or CSS is a common problem in web development. For example, suppose that you want to use [Bootstrap's button component](https://getbootstrap.com/docs/4.3/components/buttons/) on your page. To use the button component you need to add a link to Bootstrap's stylesheet in your HTML, like this:

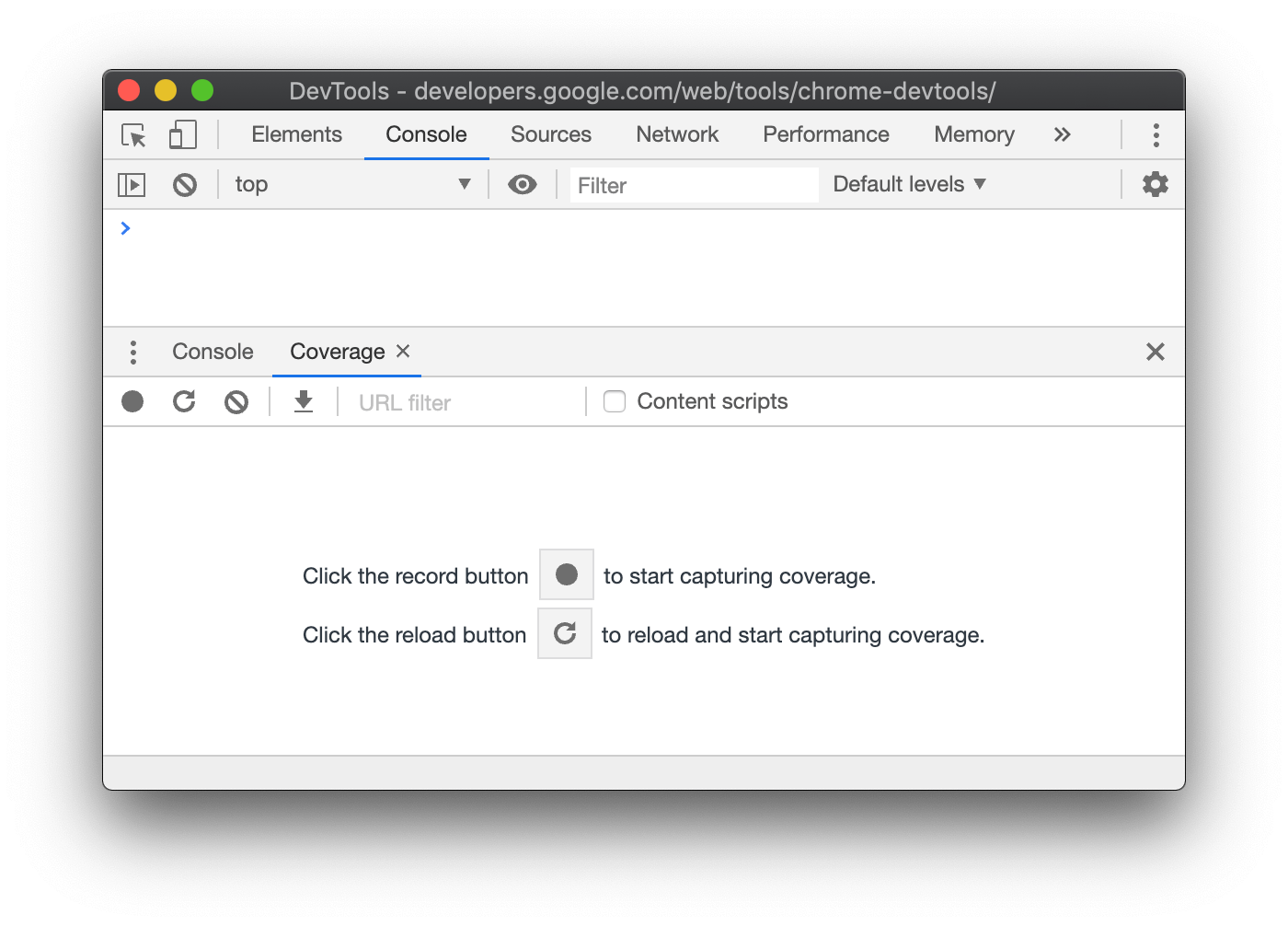
...  
<head>  
 ...  
 <link rel="stylesheet"  
 href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"  
 integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"  
 crossorigin="anonymous">  
 ...  
</head>  
...

This stylesheet doesn't just include the code for the button component. It contains the CSS for all of Bootstrap's components. But you're not using any of the other Bootstrap components. So your page is downloading a bunch of CSS that it doesn't need. This extra CSS is a problem for the following reasons:

* The extra code slows down your page load. See [Render-Blocking CSS](https://developers.google.com/web/fundamentals/performance/critical-rendering-path/render-blocking-css/).
* If a user accesses the page on a mobile device, the extra code uses up their cellular data.

## [#](https://developer.chrome.com/docs/devtools/coverage/#open) Open the Coverage tab

1. [Open the Command Menu](https://developer.chrome.com/docs/devtools/command-menu/).
2. Start typing coverage, select the **Show Coverage** command, and then press Enter to run the command. The **Coverage** tab opens in the **Drawer**.

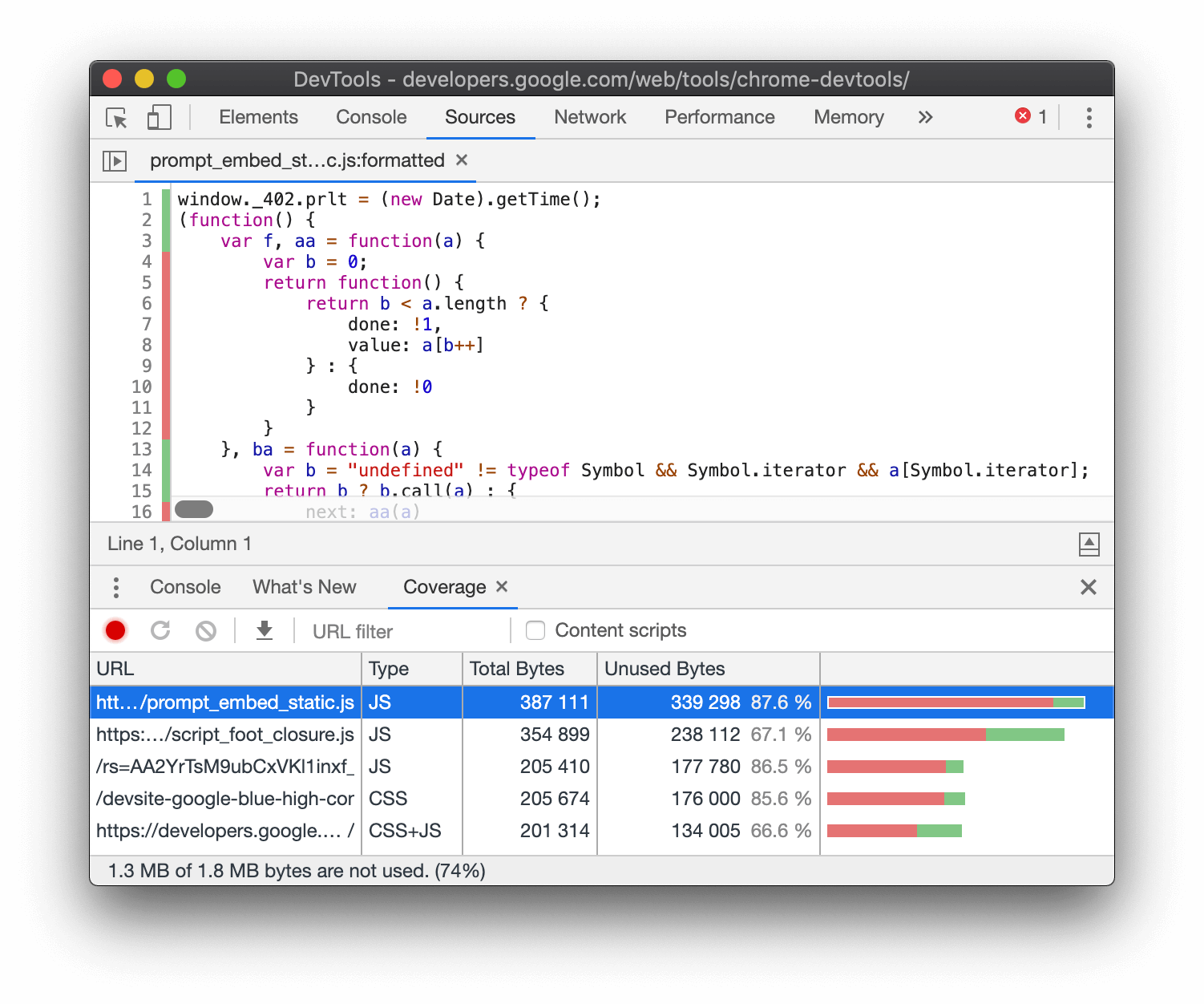
**Figure 2**. The **Coverage** tab.

## [#](https://developer.chrome.com/docs/devtools/coverage/#record) Record code coverage

1. Click one of the following buttons in the **Coverage** tab:
   * Click **Start Instrumenting Coverage And Reload Page** Reloadif you want to see what code is needed to load the page.
   * Click **Instrument Coverage** Recordif you want to see what code is used after interacting with the page.
2. Click **Stop Instrumenting Coverage And Show Results** Stopwhen you want to stop recording code coverage.

## [#](https://developer.chrome.com/docs/devtools/coverage/#analyze) Analyze code coverage

The table in the **Coverage** tab shows you what resources were analyzed, and how much code is used within each resource. Click a row to open that resource in the **Sources** panel and see a line-by-line breakdown of used code and unused code.

**Figure 3**. A code coverage report.

* The **URL** column is the URL of the resource that was analyzed.
* The **Type** column says whether the resource contains CSS, JavaScript, or both.
* The **Total Bytes** column is the total size of the resource in bytes.
* The **Unused Bytes** column is the number of bytes that weren't used.
* The last, unnamed column is a visualization of the **Total Bytes** and **Unused Bytes** columns. The red section of the bar is unused bytes. The green section is used bytes.

# Remove Unused CSS and Dead Rules From StyleSheets

When optimizing your website for faster loading, I am sure you have encountered a red flag in [Google Page Speed Insights](https://developers.google.com/speed/pagespeed/insights/)

Remove dead rules from stylesheets and defer the loading of CSS not used for above-the-fold content to reduce unnecessary bytes consumed by network activity.

Although it may look like the CSS file size is only a few KB and not even 1 MB, don’t be deluded by the size alone. All of these things add up and may slow down your website, so it is recommended to remove unused CSS and dead rules from your stylesheets, make the stylesheet file leaner, minify it and perfect it for loading faster on slow connections. A faster website has better chances of [ranking higher](https://cultofweb.com/blog/ranking-factors-google-search/), so you should take the loading time of your website very, very seriously.

### What is Unused CSS and Dead Rules?

Unused CSS or “dead rules” is nothing but CSS code which is not being used on your webpage. If you remove this code, it will make the CSS file leaner, lightweight and help improve the page’s loading time to some extent.

Let’s take an example to understand this.

Let’s say you have the following CSS code in your stylesheet

.red\_highlight\_box {

padding: 10px;

background-color: black;

color: white;

cursor: pointer;

max-width: 85%;

margin: 0 auto;

border-bottom: 2px solid white;

//border-radius: 10px;

font-family: 'Secular One', Arial, Helvetica, sans-serif;

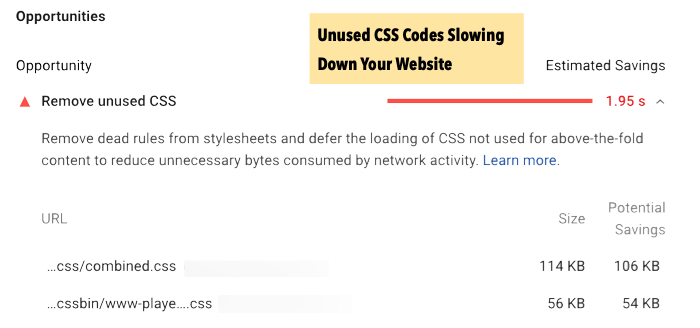
}

The above code is for the class name “red\_highlight\_box”. It is possible you are not going to use this class in your website’s theme and it is also not being used by any existing page on the site.

In this case, the above code is Unused CSS or a Dead Rule or “Bloated Code”. It’s doing nothing but slowing down your website and you should definitely consider removing it from the stylesheet file once and for all.

#### A Word of caution.

Commenting out the Unused CSS is not the same as removing it altogether. When you comment out the Unused CSS code, the code is still there and adds up to the file size. The goal of unused css cleanup is to reduce the file size of the CSS file for faster loading, rendering in the user’s browser. So commenting out unnecessary CSS code is not recommended, you should permanently remove bloated code from your CSS file



Imagine if you have 5 stylesheets being called from every page of your website and each of these stylesheets is filled with dead rules and code which is not being used in your website’s HTML at all. This is quite normal if you are using a Free WordPress theme or a premium theme because the CSS contains everything that your theme “may” require. The developer of the HTML template or WordPress theme has considered all the possibilities, widgets, sections, fonts, styling, background color and hence written a code which will satisfy all the possibilities.

**But in reality, you aren’t going to use all the possibilities.** Practice has shown that you will use less than 30% of all the possibilities coded in the CSS file and hence, there would be a lot of Unused CSS and Dead rules in stylesheets which would unnecessarily increase the loading time of your website’s pages.

### Remove Unused CSS From StyleSheets Using Unused-CSS.Com

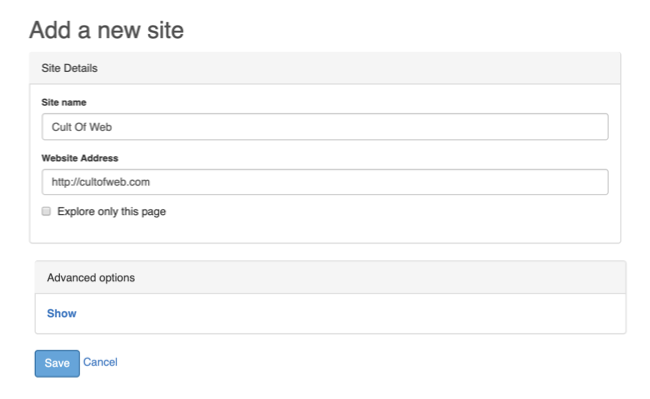
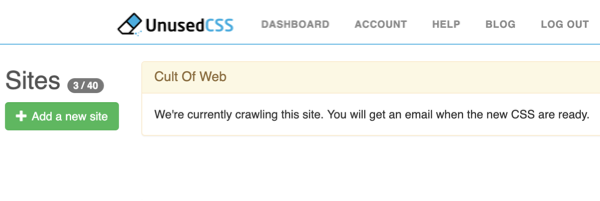
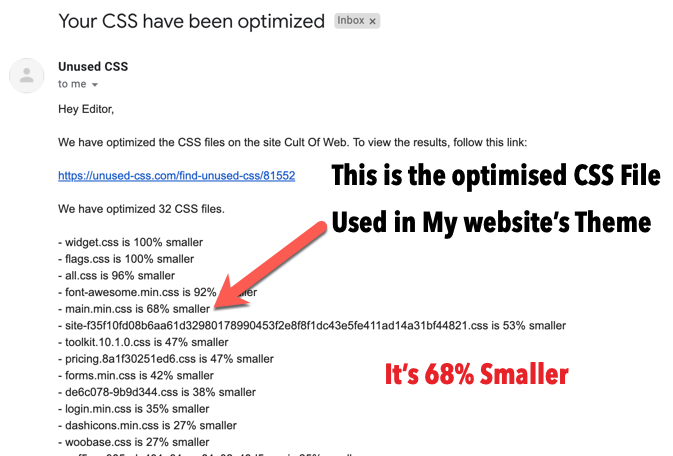
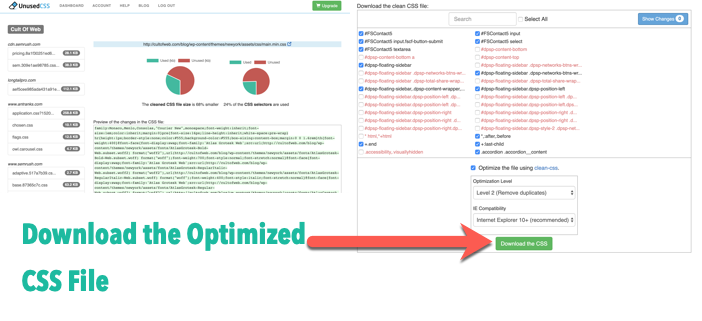
There are many tools to merge CSS files, minify them or remove dead rules from stylesheets. But I highly recommend using [Unused-Css.Com](https://unused-css.com/). In fact, it is the only tool that worked for me every time without any failure. I have tested it on multiple websites and the output CSS file which I got has never failed me.

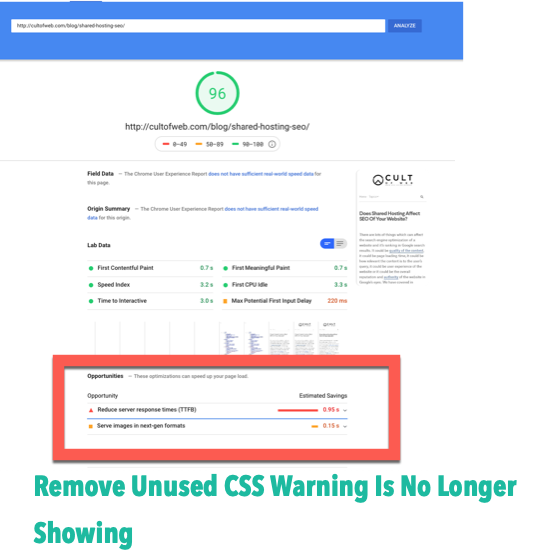
I have given [Purify CSS](https://purifycss.online/) a try but for larger sites with lot of CSS code, the output wasn’t correct. Some of the CSS codes were removed from the output file which are in use on some other page.

The best way to go about this process is not to manually remove unused CSS from your website’s stylesheet files. Because, when you are doing it manually, there is always a possibility of human error. It’s also not scalable and quite laborious to manually inspect each and every class and find in which pages or scenarios they are being used. So manual way is not full proof and we will leave it out.

You need an online tool that will crawl your entire website, consider all the pages and extract only the used CSS in every page. Once it has extracted the usable CSS on every page on your entire site, it will combine it and give you the final output file, which is the clean CSS – free from bloated code or dead rules.

Here is how you do it.

1. Signup for a free account at [Unused-css.com](https://unused-css.com). The free plan let’s you clean the CSS for one website only with a maximum of 100 pages. My website had more than 4000 pages so I took the monthly plan since I had to optimize CSS files in bulk for multiple sites.
2. Add your website(s) and set up a scan. Their Crawler will crawl your entire website and all the pages, so make sure your robots.txt or other scripts are not blocking their crawler in any way.  
   
3. You will soon get an email when their crawler has finished crawling all the sites and has optimized all the CSS files used in your entire website. Don’t worry, all these files won’t be combined into one, you can selectively download each optimized CSS File on your own and then decide to merge it with another, totally depending on your individual requirement.
4. Login to your Unused-CSS account, select the project and you would see all the CSS Files listed in the left sidebar. In the left sidebar, first, select the CSS file which is being used by your website’s theme or HTML template (it should be somewhere down the bottom in the left sidebar). Next, click “Download CSS” and you get the minified CSS file with no bloated code or unused code.
5. Upload this new CSS file in your website’s theme directory. Don’t delete the original CSS file yet, instead upload this new one and call the new optimized CSS file from your website’s head section. Once that is done, go to [Google page speed insights](https://developers.google.com/speed/pagespeed/insights/) and run the diagnostics again.Use random pages on your website, it’s totally fine since Unused-CSS.com has scanned your entire site and combined all the active CSS classes being used into one file. You should see the warning – **Remove unused CSS and remove dead rules from stylesheets** no more showing up under the “Opportunities” section in Google page speed insights.



Of course, you don’t have to use a paid tool to get this done. You can use Chrome Dev tools and do it manually or [defer Unused CSS.](https://developers.google.com/web/tools/lighthouse/audits/unused-css)

But after giving Unused-CSS.com a try for 3 of my sites, I am completely satisfied with the output. The premium plan comes with scheduled weekly scans and up to 5000 pages per website, which is more than sufficient to update your CSS files, in case, it needs to be done at a routine interval.

### What if my Website’s Theme Updates? Do I have to Remove Unused CSS All Over Again?

Yes, you have to do this again and again if you are using a free WordPress theme or a free HTML template. This is because when your theme receives a new update, it will overwrite the code in your style.css file.

One good way to avoid this process is to rename your file as “main.css” or “somethingelse.css” and call this file from the head section manually. Keep the default style.css file as it is and make sure it is not being called in your website’s source code. You can [dequeue stylesheets](https://codex.wordpress.org/Function_Reference/wp_dequeue_style) from your theme’s functions.php file (if you are using WordPress as a content management system)

That way, when the theme has a new update, it won’t overwrite the optimized CSS file which is free from bloated code.

My suggestion would be not to use Free WordPress themes. Use a premium WordPress theme that comes with developer support or hand-code your own WordPress theme from scratch, so you know how everything fits and which files are being called from which sections.

**How Do You Remove Unused CSS From a Site?**

[](https://css-tricks.com/author/chriscoyier/)

[Chris Coyier](https://css-tricks.com/author/chriscoyier/) on Nov 19, 2019 (Updated on Nov 29, 2019)

Here’s what I’d like you to know upfront: **this is a hard problem.** If you’ve landed here because you’re hoping to be pointed at a tool you can run that tells you exactly what CSS you can delete from your project, well… there are tools out there, but I’m warning you to be very careful with them because [none of them can ever tell you the complete story](https://css-tricks.com/heres-the-thing-about-unused-css-tools/).

I know what you want. You want to run the tool, delete what it tells you, and you have a faster site in 2.2 minutes. I’m sorry, but I’m going to disappoint you.

I think you should have a healthy level of skepticism for any tool like that. None of them are exactly *lying* to you — they often just don’t have enough information to give you results that are safe and actionable. That’s not to say you can’t use them or it can’t be done. Let’s take a walk.

**The motivation**

I imagine the #1 driver for the desire to remove unused CSS is this:

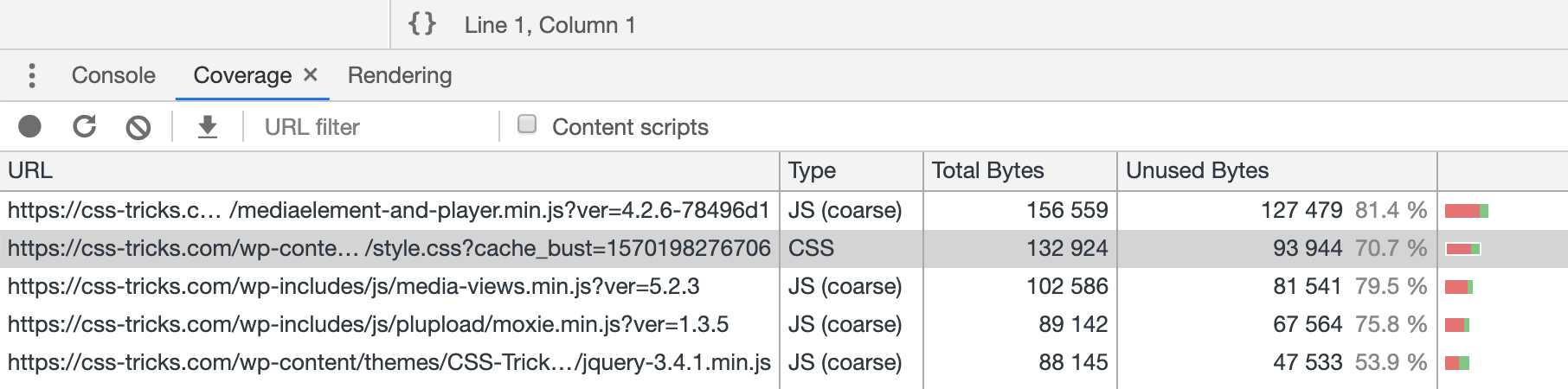
**You used a CSS framework (e.g. Bootstrap), included the framework’s entire CSS file, and you only used a handful of the patterns it provides.**

I can empathize with that. CSS frameworks often don’t provide simple ways to opt-in to only what you are using, and customizing the source to work that way might require a level of expertise that your team doesn’t have. That might even be the reason you reached for a framework to begin with.

Say you’re loading 100 KB of CSS. I’d say that’s a lot. (As I write, this site has ~23 KB, and there are quite a lot of pages and templates. I don’t do anything special to reduce the size.) You have a suspicion, or some evidence, that you aren’t using a portion of those bytes. I can see the cause for alarm. If you had a 100 KB JPG that you could compress to 20 KB by dropping it onto some tool, that’s awesome and totally worth it. **But the gain in doing that for CSS is even more important because CSS is loaded in the head and is render blocking**. The JPG is not.

**😬 Looking at “coverage”**

Chrome’s DevTools has a “Coverage” tab that will tell you how much of your CSS and JavaScript is in use. For example, if I visit the homepage of CSS-Tricks right now…



It tells me that 70.7% of my style.css file is unused. I imagine it’s right, and that the rest of the CSS is used elsewhere. I didn’t just dump a big style library onto this site; I wrote each line of that by hand, so I have my doubts that more than 2/3 of it is unused globally.

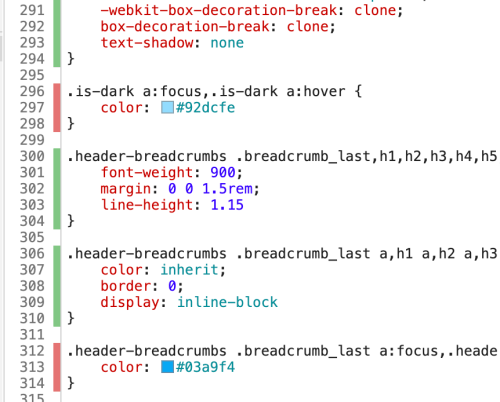
I assumed I could start “recording” then click around different areas of the site and watch that unused number go down as different pages with different HTML are rendered, but alas, when the page refreshes, so does the Coverage tab. It’s not very useful in getting a multi-page look at CSS coverage, unless you have a Single Page App I guess?

I hate to say it but I find looking at code coverage pretty useless. For me, it paints a dire picture of all this unused code on the site, which preys upon my doubts, but all I can do is worry about it.

This might be the very thing that’s given you the idea that unused CSS needs to be discovered and deleted in the first place.

**My primary concern**

My biggest concern is that you look at something like code coverage and see your unused lines:



And you go, *Perfect! I’ll delete that CSS!* And you do, only to find out it wasn’t unused at all and you caused big styling problems throughout the site. Here’s the thing: you don’t actually know if a CSS selector is unused unless you:

1. check coverage on every single page of your entire site…
2. while executing all JavaScript…
3. under every possible combination of state…
4. in every possible combination of media queries you’ve used.

Checking your homepage doesn’t count. Checking all your top-level pages doesn’t count. You gotta dig through *every* page, including states that aren’t always top-of-mind, not to mention all of the edge-case scenarios. Otherwise, you might end up deleting the dropdown styling for the credit card choice dropdown in the pop-up modal that appears for users with a disabled account who’ve logged in during their grace period that also have a gift card to apply.

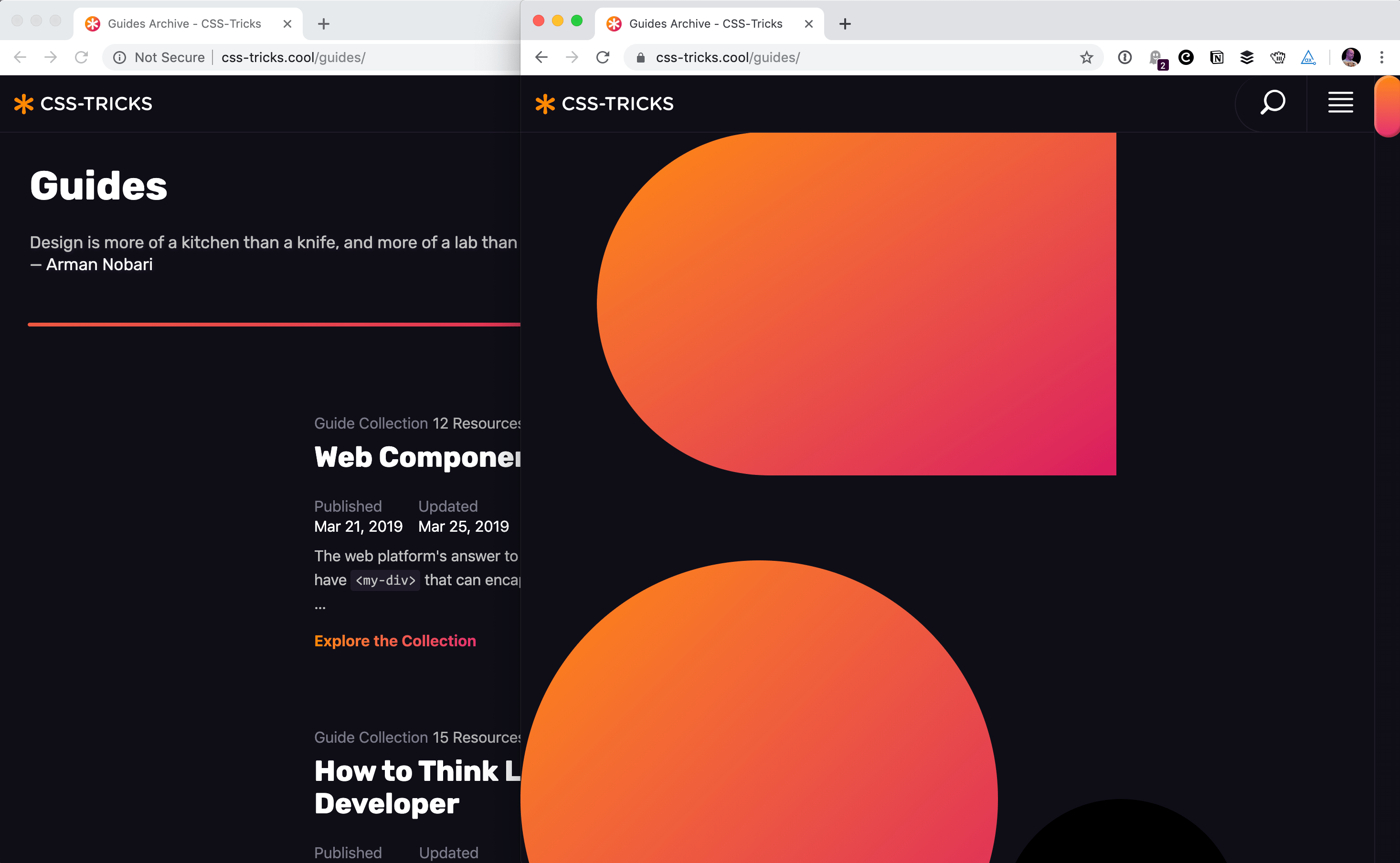
This is too complex for automated tooling to promise their approach works perfectly, particularly when factoring in the unknowns of browser context (different screen sizes, different capabilities, different browsers) and third parties.

Here’s an example of my concern playing out:

**PurifyCSS Online takes some URLs and instantly provides a copy-pasteable chunk of CSS to use**

Here’s me dropping my css-tricks.com into [PurifyCSS Online](https://purifycss.online/) and getting new CSS.

Oooops!

On the left, CSS-Tricks as normal. On the right, I applied the new “purified” CSS, which deleted a bunch of CSS necessary for other pages.

It gave me the opportunity to put in other URLs (which is nice) but there are tens of thousands of URLs on CSS-Tricks. Many of them are fairly similar, but all of them have the potential of having selectors that are used. I get the impression it didn’t execute JavaScript, because anything that came onto the page via JavaScript was left unstyled. It even deleted my :hover states.

Perhaps you can see why my trust in these tools is so low.

**Part of a build process**

[PurifyCSS](https://github.com/purifycss/purifycss) is probably more regularly used as a build process tool rather than the online interface. Their docs have instructions for Grunt, Gulp, and webpack. For example, globbing files to check and process them:

var content = ['\*\*/src/js/\*.js', '\*\*/src/html/\*.html'];

var css = ['\*\*/src/css/\*.css'];

var options = {

// Will write purified CSS to this file.

output: './dist/purified.css'

};

purify(content, css, options);

This gives you a lot more opportunity for accuracy. That content blob could be a list of every single template, partial, and JavaScript file that builds your site. That might be a pain to maintain, but you’ll certainly get more accuracy. It doesn’t account for content in data stores (e.g. this blog post that lives in a database) and third-party JavaScript, but maybe that doesn’t matter to you or you can account for it some other way.

PurgeCSS, a competitor to PurifyCSS, [warns](https://www.purgecss.com/comparison) about its comparison technique:

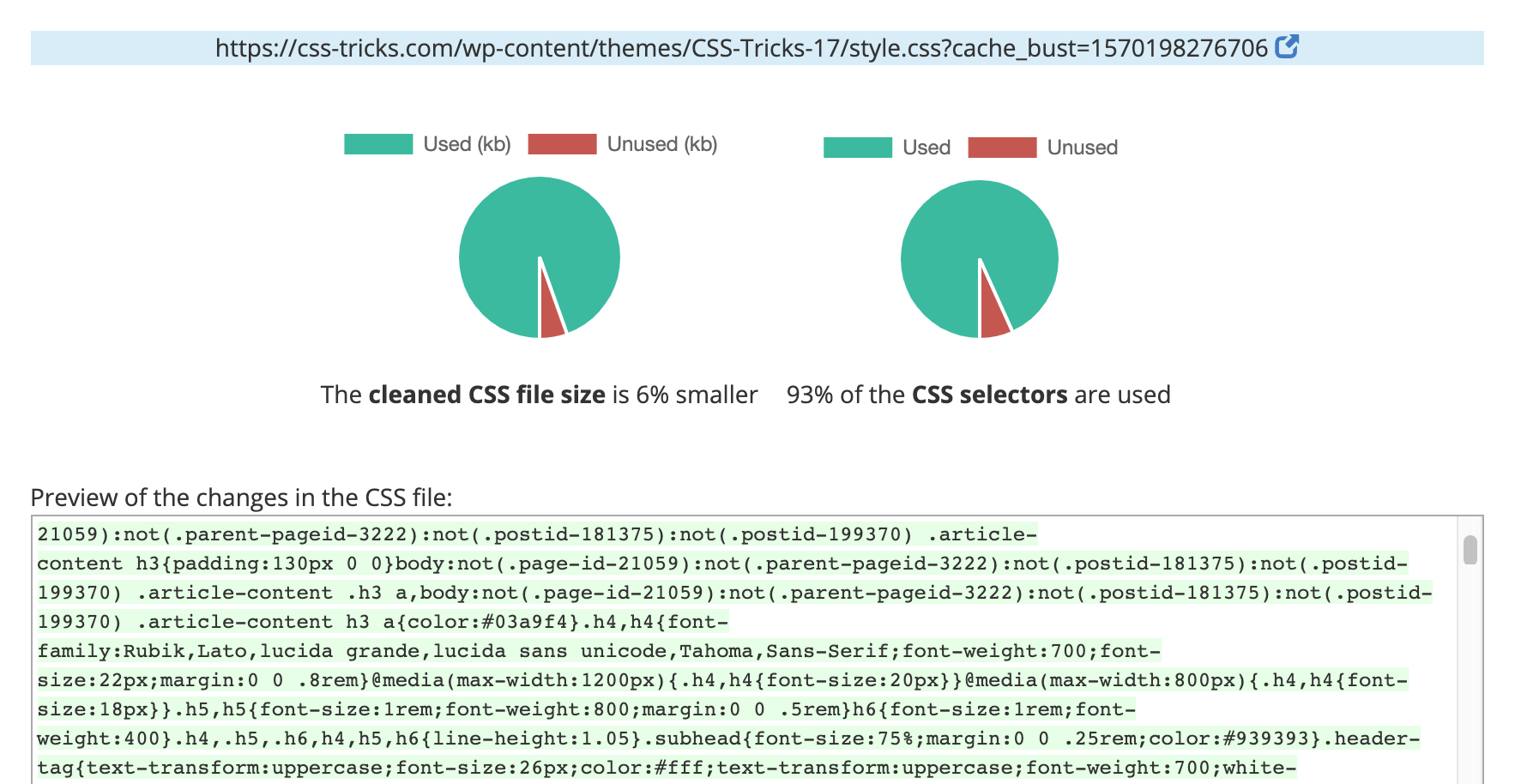
PurifyCSS can work with any file type, not just HTML or JavaScript. PurifyCSS works by looking at all of the words in your files and comparing them with the selectors in your CSS. Every word is considered a selector, which means that a lot of selectors can be erroneously consider used. For example, you may happen to have a word in a paragraph that matches a selector in your CSS.

So keep that in mind as well. It’s dumb in the way it compares potential selector matches, which is both clever and dangerous.

**UnusedCSS is an online service that crawls your site for you**

Manually configuring a tool to look at every page on your site from every angle is certainly a chore and something that will need to be kept in sync day-to-day as your codebase evolves. Interestingly, the online service [UnusedCSS](https://unused-css.com/) tries to overcome this burden by crawling the site itself based on a single URL you give it.

I signed up for the **paid** service and pointed it at CSS-Tricks. I admit, with just a glance at the results, it feels a lot more accurate to me:

It’s telling me I’m using 93% of my CSS, which feels more inline to me as hand-author of all the CSS on this site.

It also lets you download the cleaned file and offers lots of customization, like checking/unchecking selectors you actually want/don’t want (e.g. you see a class name it doesn’t think you need, but you know for sure you actually do need it) as well as prefixing and removing duplicate selectors.

I enjoyed the increased accuracy of the online crawling service, **but there was a lot of noise, and I also can’t see how I’d incorporate it practically into a day-to-day build and release process**.

**Tooling is generally used post-processing**

Say your CSS is built with Less or Sass, then uses a postprocessor to compile it into CSS. You’d probably incorporate automated unused CSS cleaning at the very end of whatever other CSS preprocessing you do. Like…

1. Sass
2. PostCSS / Autoprefixer
3. **[ Clean Unused CSS ]**
4. Production CSS

That both makes sense and is slightly funny to me. You don’t actually *fix* the styling that generates unused CSS. Instead, you just wipe it away at the end of the build. I suppose JavaScript has been doing that kind of thing with tree shaking for a while, so there is a precedent, but it still feels weird to me because a CSS codebase is so directly hands-on. This setup almost encourages you to dump CSS wherever because there is no penalty for overdoing. It removes any incentive to understand how CSS is applied and used.

**PurgeCSS is another tool that takes explicit input and gives you the results**

[PurgeCSS](https://www.purgecss.com/) is another player in the unused CSS market. One tangential thing I like about it is that it [clearly explains how it differs](https://www.purgecss.com/comparison) from other tools. For example, compared to PurifyCSS:

The biggest flaw with PurifyCSS is its lack of modularity. However, this is also its biggest benefit. PurifyCSS can work with any file type, not just HTML or JavaScript. PurifyCSS works by looking at all of the words in your files and comparing them with the selectors in your CSS. Every word is considered a selector, which means that a lot of selectors can be erroneously consider used. For example, you may happen to have a word in a paragraph that matches a selector in your CSS.

PurgeCSS fixes this problem by providing the possibility to create an extractor. An extractor is a function that takes the content of a file and extracts the list of CSS selectors used in it. It allows a perfect removal of unused CSS.

PurgeCSS seems like the big dog at the moment. Lots of people are using it and writing about it.

* [Nghia Pham wrote about](https://medium.com/dwarves-foundation/remove-unused-css-styles-from-bootstrap-using-purgecss-88395a2c5772) how to use it specifically with Bootstrap,
* [Greg Kohn wrote a post](https://www.viget.com/articles/a-better-approach-for-using-purgecss-with-tailwind/) warning that it doesn’t delete selectors in unusual circumstances with whitelists.
* [Flavio Copes wrote about](https://flaviocopes.com/tailwind-setup/) running it with npm scripts and PostCSS.
* [Sarah Dayan carefully details](https://frontstuff.io/how-i-dropped-250-kb-of-dead-css-weight-with-purgecss) how it works with Tailwind.

Despite PurgeCSS needing special configuration to work with Tailwind, it seems like Tailwind and PurgeCSS are two peas in a pod. In fact, [their docs recommend using them together](https://tailwindcss.com/docs/controlling-file-size/) and provides [a CLI](https://www.purgecss.com/cli) for using it in a build process.

I believe the gist of it is this: Tailwind produces this big CSS file full of utility selectors. But they don’t intend for you to use the entire thing. You use these utility selectors in your HTML to do all your styling, then use PurgeCSS to look at all your HTML and shake out the unused utility selectors in your production CSS.

Still, it will be an ongoing maintenance issue to teach it about every single template on your site — JavaScript, HTML, or otherwise — while manually configuring anything that relies on third-party resources and knowing that any data that comes from a data store probably cannot be looked at during a build process, making it something to account for manually.

**My favorite technique: have someone who is really familiar with your CSS codebase be aware of the problem and aim to fix it over time**

Perhaps this feels like the approach of an old-timer who needs to get with the times, but hey, this just feels like the most practical approach to me. Since this problem is so hard, I think hard work is the answer to it. It’s understanding the problem and working toward a solution over time. A front-end developer that is intimately involved in your front end will have an understanding about what is used and usused in CSS-land after time and can whittle it down.

An extreme testing approach I’ve seen is using a (i.e. background-image: url(/is-this-being-used.gif?selector);) in the CSS block and then checking server logs over time to see if that image has been accessed. If it is accessed, it was used; if not, it wasn’t.

But perhaps my favorite tool in the potential toolbox is this:

**Visual regression testing**

You screenshot as much of your site as possible — like all of the most important pages and those pages manipulated into different states — plus across different browsers and screen sizes. Those screenshots are created from your master branch on Git.

Then, before any branches gets merged into Master, you take all those screenshots of them and compare those to the screenshots in master. Not manually, but programmatically.

That’s exactly what [Percy](https://percy.io/) does, so watch this:

There have been other stabs at visual regression testing tools over the years, but Percy is the only one I’ve seen that makes clear sense to me. I don’t just need to take screenshots; I want them compared so I can see visual differences between them. I don’t just want to see the differences; I want to approve or disapprove them. I also want that approval to block or allow merges and I want to be able to control the browser before the screenshot is taken. I don’t want to manually update the comparison images. That’s all bread-and-butter Percy stuff.

**Full disclosure:** Percy has sponsored things here on CSS-Tricks here before — including that video above — but not this post.

**The relation to Atomic CSS and CSS-in-JS**

I’m sure there are lots of people reading this that would say: I don’t have unused CSS because the tooling I use generates the exact CSS it needs and nothing more.

Hey, that’s kinda cool.

Maybe that’s [Atomizer](https://acss.io/guides/atomizer.html). Maybe that’s [Tachyons](https://tachyons.io/) that you also run through [UnCSS](https://github.com/uncss/uncss) and you are super careful about it. Maybe it’s the [Tailwind](https://tailwindcss.com/) + [PurgeCSS](https://www.purgecss.com/) combo that’s all the rage right now.

Maybe you tackle styles some other way. If you’re tightly coupling JavaScript components and styles, like React and Emotion, or even just using CSS modules with whatever, less unused CSS is an advantage of CSS-in-JS. And because tree-shaking and code-splitting come along for the ride in many JavaScript-based build processes, you not only have less CSS but only load what you need at the moment. [There are tradeoffs](https://css-tricks.com/the-differing-perspectives-on-css-in-js/) to all this though.

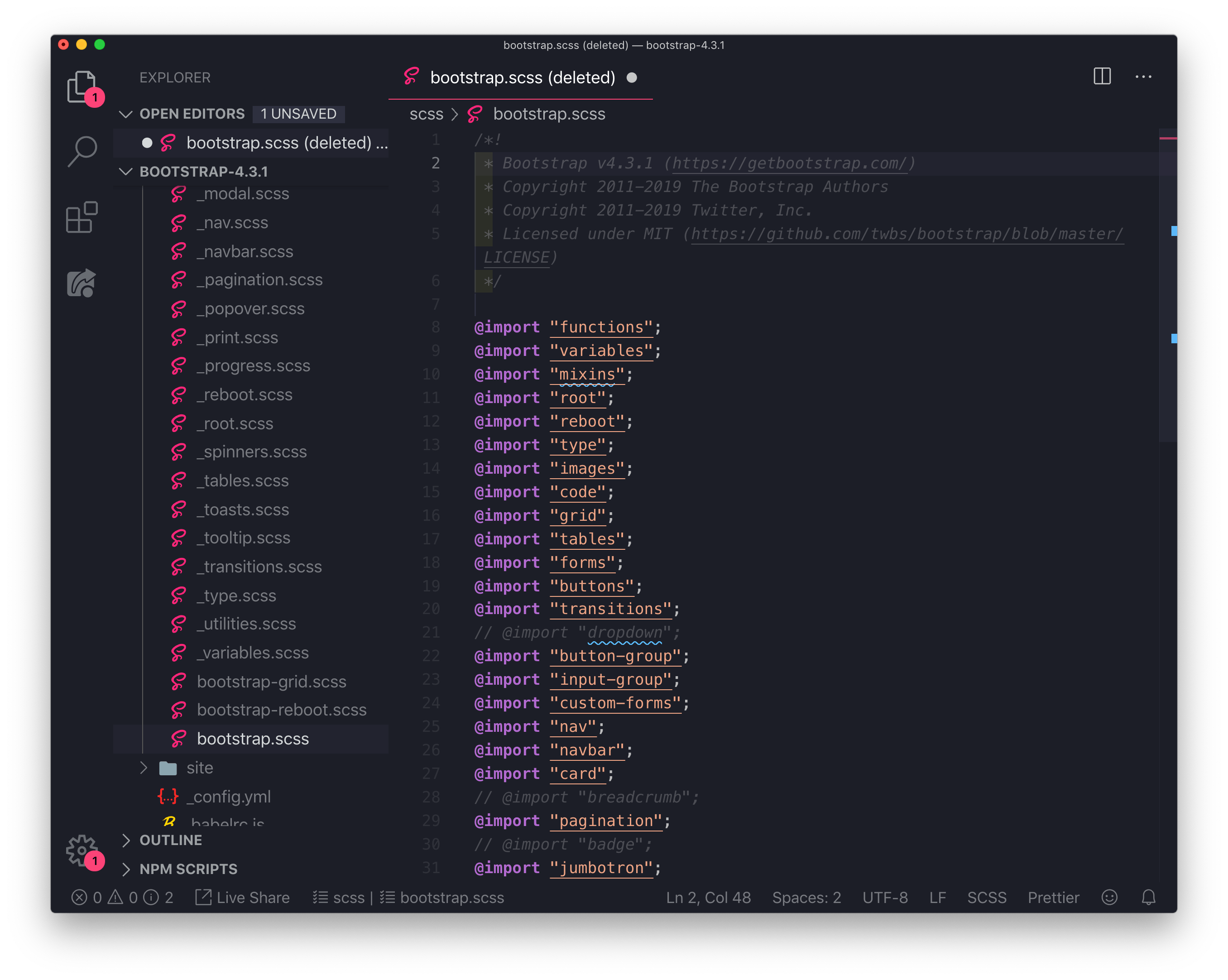
**How do you avoid unused CSS in future projects?**

I think the future of styling is an intentional split between global and componentized styles. Most styles are scoped to components, but there are global styling choices that are made that take clear advantage of the cascade (e.g. global typography defaults).

If most styling is left scoped to components, I think there is less opportunity for unused styles to build up as it’s much easier to wrap your mind around a small block of HTML and a small block of CSS that directly relate to each other. And when components die or evolve, the styling dies or evolves with it. CSS bundles are made from components that are actually used.

CSS-in-JS solutions naturally head in this direction as styles are bound to components. That’s the main point, really. But it’s not required. I like the generic approach of [CSS modules](https://github.com/css-modules/css-modules), which is pretty much entirely for style scoping and doesn’t mandate that you use some particular JavaScript framework.

If all that seems theoretical or out-of-reach, and you just have a Bootstrap site where you’re trying to reduce the size of all that Bootstrap CSS, I’d recommend starting by using Bootstrap from the source instead of the final default distributed bundle. The source is SCSS and built from a bunch of high-level includes, so if you don’t need particular parts of Bootstrap, you can remove them.

Removing dropdowns, badges, and breadcrumbs from Bootstrap before the build.