



24

2013

24 WAYS

Credits

24 ways is the advent calendar for web geeks. For twenty-four days each December we publish a daily dose of web design and development goodness to bring you all a little Christmas cheer.

- 24 ways is brought to you by Perch CMS
- Produced by Drew McLellan, Brian Suda, Anna Debenham and Owen Gregory.
- Designed by Paul Robert Lloyd.
- eBook published by edgeofmyseat.com and produced by Rachel Andrew.
- Possible only with the help and dedication of our authors.

2013

Scourge of browser vendors everywhere, WaSP buzzed its last in March. Dave Shea's CSS Zen Garden celebrated its tenth anniversary in May, and Google Glass was released. Ever broad in its interests, 24 ways tamed Grunt, URLs and GitHub Pages, encouraged readers to write and publish books, and leavened all that with goodies on project management, web typography and SVG.

URL Rewriting for the Fearful	5
Make Your Browser Dance	21
Coding Towards Accessibility	33
Git for Grown-ups	48
JavaScript: Taking Off the Training Wheels	58
Levelling Up	67
Animating Vectors with SVG	76

Kill It With Fire! What To Do With Those Dreaded	
FAQs	84
Keeping Parts of Your Codebase Private on GitHub	95
Why Bother with Accessibility?.....	104
Grunt for People Who Think Things Like Grunt are Weird and Hard	124
The Responsive Hover Paradigm	147
Data-driven Design with an Annual Survey.....	163
Home Kanban for Domestic Bliss	177
In Their Own Write: Web Books and their Authors.....	185
Credits and Recognition	203
Project Hubs: A Home Base for Design Projects.....	214
Get Started With GitHub Pages (Plus Bonus Jekyll)	224
How to Write a Book	245
Untangling Web Typography	255
Managing a Mind	262
Bringing Design and Research Closer Together	271
The Command Position Principle	289
Run Ragged	297

1. URL Rewriting for the Fearful

Drew McLellan

24ways.org/201301

I think it was Marilyn Monroe who said, “If you can’t handle me at my worst, please just fix these rewrite rules, I’m getting an internal server error.” Even the blonde bombshell hated configuring URL rewrites on her website, and I think most of us know where she was coming from.

The majority of website projects I work on require some amount of URL rewriting, and I find it mildly enjoyable — I quite like a good rewrite rule. I suspect you may not share my glee, so in this article we’re going to go back to basics to try to make the whole rigmarole more understandable.

When we think about URL rewriting, usually that means adding some rules to an `.htaccess` file for an Apache web server. As that’s the most common case, that’s what I’ll be sticking to here. If you work with a different server,

there's often documentation specifically for translating from Apache's `mod_rewrite` rules. I even found an **automatic converter for nginx**.

This isn't going to be a comprehensive guide to every URL rewriting problem you might ever have. That would take us until Christmas. If you consider yourself a trial-and-error dabbler in the HTTP 500-infested waters of URL rewriting, then hopefully this will provide a little bit more of a basis to help you figure out what you're doing. If you've ever found yourself staring at the white screen of death after screwing up your `.htaccess` file, don't worry. As Michael Jackson once insipidly whined, you are not alone.

THE BASICS

Rewrite rules form part of the Apache web server's configuration for a website, and can be placed in a number of different locations as part of your *virtual host* configuration. By far the simplest and most portable option is to use an `.htaccess` file in your website root. Provided your server has `mod_rewrite` available, all you need to do to kick things off in your `.htaccess` file is:

```
RewriteEngine on
```

The general formula for a rewrite rule is:

```
RewriteRule URL/to/match URL/to/use/if/it/matches  
[options]
```

When we talk about URL rewriting, we're normally talking about one of two things: **redirecting** the browser to a different URL; or **rewriting** the URL internally to use a particular file. We'll look at those in turn.

Redirects

Redirects match an incoming URL, and then redirect the user's browser to a different address. These can be useful for maintaining legacy URLs if content changes location as part of a site redesign. Redirecting the old URL to the new location makes sure that any incoming links, such as those from search engines, continue to work.

In 1998, Sir Tim Berners-Lee wrote that **Cool URIs don't change**, encouraging us all to go the extra mile to make sure links keep working forever. I think that sometimes it's fine to move things around — especially to correct bad URL design choices of the past — provided that you can do so while keeping those old URLs working. That's where redirects can help.

A redirect might look like this

```
RewriteRule ^article/used/to/be/here.php$ /article/now/  
lives/here/ [R=301,L]
```

Rewriting

By default, web servers closely map page URLs to the files in your site. On receiving a request for `http://example.com/about/history.html` the server goes to the configured folder for the `example.com` website, and then goes into the `about` folder and returns the `history.html` file.

A rewrite rule changes that process by breaking the direct relationship between the URL and the file system. “When there’s a request for `/about/history.html`” a rewrite rule might say, “use the file `/about_section.php` instead.”

This opens up lots of possibilities for creative ways to map URLs to the files that know how to serve up the page. Most MVC frameworks will have a single rule to rewrite *all* page URLs to one single file. That file will be a script which kicks off the framework to figure out what to do to serve the page.

```
RewriteRule ^for/this/url/$ /use/this/file.php [L]
```

MATCHING PATTERNS

By now you’ll have noted the weird `^` and `$` characters wrapped around the URL we’re trying to match. That’s because what we’re actually using here is a pattern. Technically, it is what’s called a Perl Compatible Regular Expression (PCRE) or simply a *regex* or *regexp*. We’ll call it a pattern because we’re not animals.

What are these patterns? If I asked you to enter your credit card expiry date as *MM/YY* then chances are you'd wonder what I wanted your credit card details for, but you'd know that I wanted a two-digit month, a slash, and a two-digit year. That's not a regular expression, but it's the same idea: using some placeholder characters to define the *pattern* of the input you're trying to match.

We've already met two regexp characters.

`^`

Matches the beginning of a string

`$`

Matches the end of a string

When a pattern starts with `^` and ends with `$` it's to make sure we match the complete URL start to finish, not just part of it. There are lots of other ways to match, too:

`[0-9]`

Matches a number, 0–9. `[2-4]` would match numbers 2 to 4 inclusive.

`[a-z]`

Matches lowercase letters a–z

`[A-Z]`

Matches uppercase letters A–Z

`[a-z0-9]`

Combining some of these, this matches letters a-z and numbers 0-9

These are what we call character groups. The square brackets basically tell the server to match from the selection of characters within them. You can put any specific characters you're looking for within the brackets, as well as the ranges shown above.

However, all these just match one single character. `[0-9]` would match 8 but not 84 — to match 84 we'd need to use `[0-9]` twice.

`[0-9][0-9]`

So, if we wanted to match 1984 we could do this:

`[0-9][0-9][0-9][0-9]`

...but that's getting silly. Instead, we can do this:

`[0-9]{4}`

That means any character between 0 and 9, four times. If we wanted to match a number, but didn't know how long it might be (for example, a database ID in the URL) we could use the `+` symbol, which means *one or more*.

`[0-9]+`

This now matches 1, 123 and 1234567.

Putting it into practice

Let's say we need to write a rule to match article URLs for this website, and to rewrite them to use */article.php* under the hood. The articles all have URLs like this:

```
2013/article-title/
```

They start with a year (from 2005 up to 2013, currently), a slash, and then have a URL-safe version of the article title (a *slug*), ending in a slash. We'd match it like this:

```
^[0-9]{4}/[a-z0-9-]+/$
```

If that looks frightening, don't worry. Breaking it down, from the start of the URL (^) we're looking for four numbers ([0-9]{4}). Then a slash — that's just literal — and then anything lowercase a-z or 0-9 or a dash ([a-z0-9-]) one or more times (+), ending in a slash (/ \$).

Putting that into a rewrite rule, we end up with this:

```
RewriteRule ^[0-9]{4}/[a-z0-9-]+/$ /article.php
```

We're getting close now. We can match the article URLs and rewrite them to use *article.php*. Now we just need to make sure that *article.php* knows which article it's supposed to display.

Capturing groups, and replacements

When rewriting URLs you'll often want to take important parts of the URL you're matching and pass them along to the script that handles the request. That's usually done by adding those parts of the URL on as *query string* arguments. For our example, we want to make sure that *article.php* knows the year and the article title we're looking for. That means we need to call it like this:

```
/article.php?year=2013&slug=article-title
```

To do this, we need to mark which parts of the pattern we want to reuse in the destination. We do this with round brackets or parentheses. By placing parentheses around parts of the pattern we want to reuse, we create what's called a *capturing group*. To capture an important part of the source URL to use in the destination, surround it in parentheses.

Our pattern now looks like this, with parentheses around the parts that match the year and slug, but ignoring the slashes:

```
^([0-9]{4})/([a-z0-9-]+)/$
```

To use the capturing groups in the destination URL, we use the dollar sign and the number of the group we want to use. So, the first capturing group is \$1, the second is \$2 and so on. (The \$ is unrelated to the end-of-pattern \$ we used before.)

```
RewriteRule ^([0-9]{4})/([a-z0-9-]+)/$  
/article.php?year=$1&slug=$2
```

The value of the year capturing group gets used as \$1 and the article title slug is \$2. Had there been a third group, that would be \$3 and so on. In regexp parlance, these are called *back-references* as they refer back to the pattern.

OPTIONS

Several brain-taxing minutes ago, I mentioned some options as the final part of a rewrite rule. There are **lots of options** (or *flags*) you can set to change how the rule is processed. The most useful (to my mind) are:

R=301

Perform an HTTP 301 redirect to send the user's browser to the new URL. A status of 301 means a resource has moved permanently and so it's a good way of both redirecting the user to the new URL, and letting search engines know to update their indexes.

L

Last. If this rule matches, don't bother processing the following rules.

Options are set in square brackets at the end of the rule. You can set multiple options by separating them with commas:

```
RewriteRule ^([0-9]{4})/([a-z0-9-]+)/$  
/article.php?year=$1&slug=$2 [L]
```

or

```
RewriteRule ^about/([a-z0-9-]).jsp/$ /about/$1/  
[R=301,L]
```

COMMON PITFALLS

Once you've built up a few rewrite rules, things can start to go wrong. You may have been there: a rule which looks perfectly good is somehow not matching. One common reason for this is hidden behind that `[L]` flag.

`L` for Last is a useful option to tell the rewrite engine to stop once the rule has been matched. This is what it does — the remaining rules in the `.htaccess` file are then ignored. However, once a URL has been rewritten, the entire set of rules are then run *again* on the new URL. If the new URL matches any of the rules, that too will be rewritten and on it goes.

One way to avoid this problem is to keep your 'real' pages under a folder path that will never match one of your rules, or that you can exclude from the rewrite rules.

USEFUL SNIPPETS

I find myself reusing the same few rules over and over again, just with minor changes. Here are some useful examples to refer back to.

Excluding a directory

As mentioned above, if you're rewriting lots of fancy URLs to a collection of real files it can be helpful to put those files in a folder and exclude it from rewrite rules. This helps solve the issue of rewrite rules reapplying to your newly rewritten URL. To exclude a directory, put a rule like this at the top of your file, before your other rules. Our files are in a folder called `_source`, the dash in the rule means *do nothing*, and the L flag means the following rules won't be applied.

```
RewriteRule ^_source - [L]
```

This is also useful for excluding things like CMS folders from your website's rewrite rules

```
RewriteRule ^perch - [L]
```

Adding or removing www from the domain

Some folk like to use a `www` and others don't. Usually, it's best to pick one and go with it, and redirect the one you don't want. On this site, we don't use `www.24ways.org` so we redirect those requests to `24ways.org`.

This uses a RewriteCond which is like an if for a rewrite rule: “If this condition matches, then apply the following rule.” In this case, it’s if the HTTP HOST (or domain name, basically) matches this pattern, then redirect everything:

```
RewriteCond %{HTTP_HOST} ^www.24ways.org$ [NC]
RewriteRule ^(.*)$ http://24ways.org/$1 [R=301,L]
```

The [NC] flag means ‘no case’ — the match is case-insensitive. The dots in the domain are escaped with a backslash, as a dot is a regular expression character which means *match anything*, so we escape it because we literally mean a dot in this instance.

Removing file extensions

Sometimes all you need to do to tidy up a URL is strip off the technology-specific file extension, so that */about/history.php* becomes */about/history*. This is easily achieved with the help of some more rewrite conditions.

```
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_FILENAME}.php -f
RewriteRule ^(.+)$ $1.php [L,QSA]
```

This says *if* the file being asked for isn’t a file (!-f) *and if* it isn’t a directory (!-d) *and if* the file name plus .php is an actual file (-f) then rewrite by adding .php on the end. The QSA flag means ‘query string append’: append the existing query string onto the rewritten URL.

It's these sorts of more generic catch-all rules that you need to watch out for when your `.htaccess` gets rerun after a successful match. Without care they can easily rematch the newly rewritten URL.

LOGGING FOR WHEN IT ALL GOES WRONG

Although not possible within your `.htaccess` file, if you have access to your Apache configuration files you can enable rewrite logging. This can be useful to track down where a rule is going wrong, if it's matching incorrectly or failing to match. It also gives you an overview of the amount of work being done by the rewrite engine, enabling you to rearrange your rules and maximise performance.

```
RewriteEngine On
RewriteLog "/full/system/path/to/rewrite.log"
RewriteLogLevel 5
```

To be doubly clear: this will not work from an `.htaccess` file — it needs to be added to the main Apache configuration files. (I sometimes work using MAMP PRO locally on my Mac, and this can be pasted into the snappily named **Customized virtual host general settings** box in the **Advanced** tab for your site.)

The white screen of death

One of the most frustrating things when working with rewrite rules is that when you make a mistake it can result in the server returning an HTTP 500 Internal Server Error. This in itself isn't an error message, of course. It's more of a notification that an error has occurred. The real error message can usually be found in your Apache error log.

If you have access to your server logs, check the Apache error log and you'll usually find a much more descriptive error message, pointing you towards your mistake. (Again, if using MAMP PRO, go to **Server, Apache** and the **View Log** button.)

IN CONCLUSION

Rewriting URLs can be a bear, but the advantages are clear. Keeping a tidy URL structure, disconnected from the technology or file structure of your site can result in URLs that are easier to use and easier to maintain into the future.

If you're redesigning a site, remember that *cool URLs don't change*, so budget some time to make sure that any content you move has a rewrite rule associated with it to keep any links working.

Further reading

To find out more about URL rewriting and perhaps even learn more about regular expressions, I can recommend the following resources.

- From the horse's mouth, the **Apache mod_rewrite documentation**
- Particularly useful with that documentation is the **RewriteRule Flags listing**
- You may wish to don sunglasses to follow the otherwise comprehensive **Regular-Expressions.info** tutorial
- Friend of 24 ways, **Neil Crosby** has a **mod_rewrite Beginner's Guide** which I've found handy over the years.

As noted at the start, this isn't a fully comprehensive guide, but I hope it's useful in finding your feet with a powerful but sometimes annoying technology. Do you have useful snippets you often use on projects? Feel free to share them in the comments.

ABOUT THE AUTHOR



Drew McLellan is lead developer on your favourite small CMS, Perch. He is Director and Senior Developer at UK-based web development agency edgeofmyseat.com, and formerly Group Lead at the Web Standards Project. When not publishing 24 ways, Drew keeps a [personal site](#) covering web development issues and themes, [takes photos](#) and [tweets a lot](#).

2. Make Your Browser Dance

Ruth John

24ways.org/201302

It was a crisp winter's evening when I pulled up alongside the pier. I stepped out of my car and the bitterly cold sea air hit my face. I walked around to the boot, opened it and heaved out a heavy flight case. I slammed the boot shut, locked the car and started walking towards the venue.

This was it. My first gig. I thought about all those weeks of preparation: editing video clips, creating 3-D objects, making coloured patterns, then importing them all into software and configuring effects to change as the music did; targeting frequency, beat, velocity, modifying size, colour, starting point; creating playlists of these... and working out ways to mix them as the music played.

This was it. This was me **VJing**.

This was all a lifetime (well a decade!) ago.

When I started web designing, VJing took a back seat. I was more interested in interactive layouts, semantic accessible HTML, learning all the IE bugs and mastering the quirks that CSS has to offer. More recently, I have been excited by background gradients, 3-D transforms, the `@keyframe` directive, as well as new APIs such as `getUserMedia`, `indexedDB`, the `Web Audio API`

But wait, have I just come full circle? Could it be possible, with these wonderful new things in technologies I am already familiar with, that I could VJ again, right here, in a browser?

Well, there's only one thing to do: let's try it!

LET'S TAKE TO THE DANCE FLOOR

Over the past couple of years working in `The Lab` I have learned to take a much more iterative approach to projects than before. One of my new favourite methods of working is to create a **proof of concept** to make sure my theory is feasible, before going on to create a full-blown product. So let's take the same approach here.

The main VJing functionality I want to recreate is manipulating visuals in relation to sound. So for my POC I need to create a visual, with parameters that can be changed, then get some sound and see if I can analyse that sound to detect some data, which I can then use to manipulate the visual parameters. Easy, right?

So, let's start at the beginning: creating a simple visual. For this I'm going to create a CSS animation. It's just a funky `i` element with the opacity being changed to make it flash.

See the Pen [Creating a light](#) by Rumyra (@Rumyra) on [CodePen](#)

A note about prefixes: I've left them out of the code examples in this post to make them easier to read. Please be aware that you may need them. I find a great resource to find out if you do is [caniuse.com](#). You can also check out all the code for the examples in [this article](#)

START THE MUSIC

Well, that's pretty easy so far. Next up: loading in some sound. For this we'll use the **Web Audio API**. The Web Audio API is based around the concept of nodes. You have a source node: the sound you are loading in; a destination node: usually the device's speakers; and any number of processing nodes in between. All this processing that goes on with the audio is sandboxed within the `AudioContext`.

So, let's start by initialising our audio context.

```
var contextClass = window.AudioContext;
if (contextClass) {
  //web audio api available.
  var audioContext = new contextClass();
} else {
```

```
//web audio api unavailable
//warn user to upgrade/change browser
}
```

Now let's load our sound file into the new context we created with an XMLHttpRequest.

```
function loadSound() {
    //set audio file url
    var audioFileUrl = '/octave.ogg';
    //create new request
    var request = new XMLHttpRequest();
    request.open("GET", audioFileUrl, true);
    request.responseType = "arraybuffer";

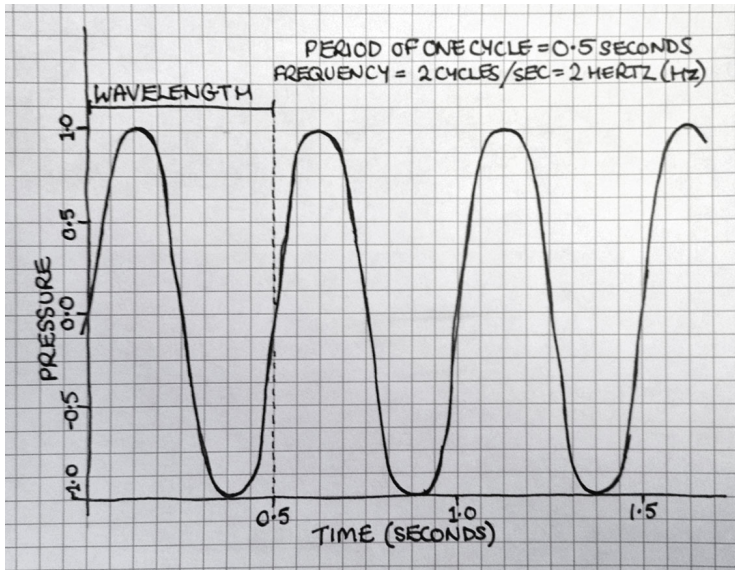
    request.onload = function() {
        //take from http request and decode into buffer
        context.decodeAudioData(request.response,
function(buffer) {
            audioBuffer = buffer;
        });
    }
    request.send();
}
```

Phew! Now we've loaded in some sound! There are plenty of things we can do with the Web Audio API: increase volume; add filters; spatialisation. If you want to dig deeper, the O'Reilly *Web Audio API* book by Boris Smus is available to read online free.

All we really want to do for this proof of concept, however, is analyse the sound data. To do this we really need to know what data we have.

LEARNING THE STEPS

Let's take a minute to step back and remember our school days and science class. I'm sure if I drew a picture of a sound wave, we would all start nodding our heads.



The sound you hear is caused by pressure differences in the particles in the air. Sound pushes these particles together, causing vibrations. Amplitude is basically

strength of pressure. A simple example of change of amplitude is when you increase the volume on your stereo and the output wave increases in size.

This is great when everything is analogue, but the waveform varies continuously and it's not suitable for digital processing: there's an infinite set of values. For digital processing, we need discrete numbers.

We have to sample the waveform at set time intervals, and record data such as amplitude and frequency. Luckily for us, just the fact we have a digital sound file means all this hard work is done for us. What we're doing in the code above is piping that data in the audio context. All we need to do now is access it.

We can do this with the Web Audio API's analysing functionality. Just pop in an analysing node before we connect the source to its destination node.

```
function createAnalyser(source) {  
  //create analyser node  
  analyser = audioContext.createAnalyser();  
  //connect to source  
  source.connect(analyser);  
  //pipe to speakers  
  analyser.connect(audioContext.destination);  
}
```

The data I'm really interested in here is frequency. Later we could look into amplitude or time, but for now I'm going to stick with frequency.

The analyser node gives us frequency data via the `getFrequencyByteData` method.

DON'T FORGET TO COUNT!

To collect the data from the `getFrequencyByteData` method, we need to pass in an empty array (a **JavaScript typed array** is ideal). But how do we know how many items the array will need when we create it?

This is really up to us and how high the resolution of frequencies we want to analyse is. Remember we talked about sampling the waveform; this happens at a certain rate (**sample rate**) which you can find out via the audio context's `sampleRate` attribute. This is good to bear in mind when you're thinking about your resolution of frequencies.

```
var sampleRate = audioContext.sampleRate;
```

Let's say your file sample rate is 48,000, making the maximum frequency in the file 24,000Hz (thanks to a wonderful **theorem from Dr Harry Nyquist**, the maximum frequency in the file is always half the sample rate). The analyser array we're creating will contain frequencies up to this point. This is ideal as the human ear hears the range 0–20,000hz.

So, if we create an array which has 2,400 items, each frequency recorded will be 10Hz apart. However, we are going to create an array which is half the size of the FFT (fast Fourier transform), which in this case is 2,048 which is the default. You can set it via the `fftSize` property.

```
//set our FFT size
analyzer.fftSize = 2048;
//create an empty array with 1024 items
var frequencyData = new Uint8Array(1024);
```

So, with an array of 1,024 items, and a frequency range of 24,000Hz, we know each item is $24,000 \div 1,024 = 23.44\text{Hz}$ apart.

The thing is, we also want that array to be updated constantly. We could use the `setInterval` or `setTimeout` methods for this; however, I prefer the new and shiny `requestAnimationFrame`.

```
function update() {
    //constantly getting feedback from data
    requestAnimationFrame(update);
    analyzer.getBytesFrequencyData(frequencyData);
}
```

PUTTING IT ALL TOGETHER

Sweet sticks! Now we have an array of frequencies from the sound we loaded, updating as the sound plays. Now we want that data to trigger our animation from earlier.

We can easily pause and run our CSS animation from JavaScript:

```
element.style.webkitAnimationPlayState = "paused";  
element.style.webkitAnimationPlayState = "running";
```

Unfortunately, this may not be ideal as our animation might be a whole heap longer than just a flashing light. We may want to target specific points within that animation to have it stop and start in a visually pleasing way and perhaps not smack bang in the middle.

There is no really easy way to do this at the moment as **Zach Saucier** explains in this [wonderful article](#). It takes some jiggery pokery with `setInterval` to try to ascertain how far through the CSS animation you are in percentage terms.

This seems a bit much for our proof of concept, so let's backtrack a little. We know by the animation we've created which CSS properties we want to change. This is pretty easy to do directly with JavaScript.

```
element.style.opacity = "1";  
element.style.opacity = "0.2";
```

So let's start putting it all together. For this example I want to trigger each light as a different frequency plays. For this, I'll loop through the HTML elements and change the `opacity` style if the frequency gain goes over a certain threshold.

```
//get light elements
var lights = document.getElementsByTagName('i');
var totalLights = lights.length;

for (var i=0; i<totalLights; i++) {
    //get frequencyData key
    var freqDataKey = i*8;
    //if gain is over threshold for that frequency animate
    light
    if (frequencyData[freqDataKey] > 160){
        //start animation on element
        lights[i].style.opacity = "1";
    } else {
        lights[i].style.opacity = "0.2";
    }
}
```

See all the code in action [here](#). I suggest viewing in a modern browser :)

Awesome! It is true — we can VJ in our browser!

LET'S DANCE!

So, let's start to expand this simple example. First, I feel the need to make lots of lights, rather than just a few. Also, maybe we should try a sound file more suited to gigs or clubs.

Check it out!

I don't know about you, but I'm pretty excited — that's just a bit of HTML, CSS and JavaScript!

The other thing to think about, of course, is the sound that you would get at a venue. We don't want to load sound from a file, but rather pick up on what is playing in real time. The easiest way to do this, I've found, is to capture what my laptop's mic is picking up and piping that back into the audio context. We can do this by using `getUserMedia`.

Let's include this in this demo. If you make some noise while viewing the demo, the lights will start to flash.

AND RELAX :)

There you have it. Sit back, play some music and enjoy the **Winamp** like experience in front of you.

So, where do we go from here? I already have a wealth of ideas. We haven't started with canvas, SVG or the 3-D features of CSS. There are other things we can detect from the audio as well. And yes, OK, it's questionable whether the browser is the best environment for this. For one, I'm using a whole bunch of nonsensical HTML elements (maybe each animation could be held within a web component in the future). But hey, it's fun, and it looks cool and sometimes I think it's OK to just dance.

ABOUT THE AUTHOR



Ruth John wireframes, designs and codes for **The Lab** at O2 (Telefonica). She also **tweets** and **blogs** a bit too. You can often find her chatting about web development, building apps and how an extra div is not the answer to your styling problems. Either that or the lesser known Thundercats characters.

3. Coding Towards Accessibility

Charlie Perrins

24ways.org/201303

“Can we make it AAA-compliant?” – does this question strike fear into your heart? Maybe for no other reason than because you will soon have to wade through the impenetrable WCAG documentation once again, to find out exactly what AAA-compliant means?

I’m not here to talk about that.

The Web Content Accessibility Guidelines are a comprehensive and peer-reviewed resource which we’re lucky to have at our fingertips. But they are also a pig to read, and they may have contributed to the sense of mystery and dread with which some developers associate the word accessibility.

This Christmas, I want to share with you some thoughts and some practical tips for building accessible interfaces which you can start using today, without having to do a ton of reading or changing your tools and workflow.

But first, let's clear up a couple of misconceptions.

DREARY, FLAT EXPERIENCES

I recently built a front-end framework for the Post Office. This was a great gig for a developer, but when I found out about my client's stringent accessibility requirements I was concerned that I'd have to scale back what was quite a complex set of visual designs.

Sites like Jakob Nielsen's old workhorse useit.com and even the pioneering [GOV.UK](http://gov.uk) may have to shoulder some of the blame for this. They put a premium on usability and accessibility over visual flourish. (Although, in fairness to Mr Nielsen, his new site nngroup.com is really quite a snazzy affair, comparatively.)

Of course, there are other reasons for these sites' aesthetics — and it's not because of the limitations of the form. You can make an accessible site look as glossy or as plain as you want it to look. It's always our own ingenuity and attention to detail that are going to be the limiting factors.

SYNECDOCHE

We must always guard against the tendency to assume that catering to screen readers means we have the whole accessibility ballgame covered.

There's so much more to accessibility than assistive technology, as you know. And within the field of assistive technology there are **plenty of other devices** for us to consider.

Planning to accommodate all these users and devices can be daunting. When I first started working in this field I thought that the breadth of technology was prohibitive. I didn't even know what a screen reader looked like. (I assumed they were big and heavy, perhaps like an old typewriter, and certainly they would be expensive and difficult to fathom.) This is nonsense, of course. Screen reader emulators are readily available as browser extensions and can be activated in seconds. **Chromevox** and **Fangs** are both excellent and you should download one or the other right now.

But the really good news is that you can emulate many other types of assistive technology without downloading a byte. And this is where we move from misconceptions into some (hopefully) useful advice.

THE MOUSE TRAP

The simplest and most effective way to improve your abilities as a developer of accessible interfaces is to unplug your mouse.

Keyboard operation has its own WCAG chapter, because most users of assistive technology are navigating the web using only their keyboards. You can go some way towards putting yourself into their shoes so easily — just by ditching a peripheral.

Learning this was a lightbulb moment for me. When I build interfaces I am constantly flicking between code and the browser, testing or viewing the changes I have made. Now, instead of checking a new element once, I check it twice: once with my mouse and then again without.

DON'T JUST :HOVER

The reality is that when you first start doing this you can find your site becomes unusable straightaway. It's easy to lose track of which element is in focus as you hit the tab key repeatedly.

One of the easiest changes you can make to your coding practice is to add `:focus` and `:active` pseudo-classes to every hover state that you write. I'm still amazed at how

many sites fail to provide a decent focus state for links (and despite previous 24 ways authors in 2007 and 2009 writing on this same issue!).

You may find that in some cases it makes sense to have something other than, or in addition to, the hover state on focus, but start with the hover state that your designer has taken the time to provide you with. It's a tiny change and there is no downside. So instead of this:

```
.my-cool-link:hover {  
  background-color: MistyRose ;  
}
```

...try writing this:

```
.my-cool-link:hover,  
.my-cool-link:focus,  
.my-cool-link:active {  
  background-color: MistyRose ;  
}
```

I've toyed with the idea of making a Sass mixin to take care of this for me, but I haven't yet. I worry that people reading my code won't see that I'm explicitly defining my focus and active states so I take the hit and write my hover rules out longhand.

JAVASCRIPT CAN PLAY, TOO

This was another revelation for me. Keyboard-only navigation doesn't necessitate a JavaScript-free experience, and up-to-date screen readers can execute JavaScript. So we're able to create complex JavaScript-driven interfaces which all users can interact with.

Some of the hard work has already been done for us. First, there are already conventions around keyboard-driven interfaces. Think about the last time you viewed a **photo album** on Facebook. You can use the arrow keys to switch between photos, and the escape key closes whichever lightbox-y UI thing Facebook is showing its photos in this week. Arrow keys (up/down as well as left/right) for progression through content; Escape to back out of something; Enter or space bar to indicate a positive intention — these are established keyboard conventions which we can apply to our interfaces to improve their accessibility.

Of course, by doing so we are improving our interfaces in general, giving all users the option to switch between keyboard and mouse actions as and when it suits them.

Second, **this guy** wants to help you out. Hans Hillen is a developer who has done a great deal of work around accessibility and JavaScript-powered interfaces. Along with The Paciello Group he has created a **version of the**

jQuery UI library which has been fully optimised for keyboard navigation and screen reader use. It's a fantastic reference which I revisit all the time

I'm not a huge fan of the jQuery UI library. It's a pain to style and the code is a bit bloated. So I've not used this demo as a code resource to copy wholesale. I use it by playing with the various components and seeing how they react to keyboard controls. Each component is also fully marked up with the relevant ARIA roles to improve screen reader announcement where possible (more on this below).

Coding for accessibility promotes good habits

This is another observation around accessibility and JavaScript. I noticed an improvement in the structure and abstraction of my code when I started adding keyboard controls to my interface elements.

Your code has to become more modular and event-driven, because any number of events could trigger the same interaction. A mouse-click, the Enter key and the space bar could all conceivably trigger the same open function on a collapsed accordion element. (And you want to keep things DRY, don't you?)

If you aren't already in the habit of separating out your interface functionality into discrete functions, you will be soon.

```
var doSomethingCool = function(){
    // Do something cool here.
}

// Bind function to a button click - pretty vanilla
$('.myCoolButton').on('click', function(){
    doSomethingCool();
    return false;
});

// Bind the same function to a range of keypresses
$(document).keyup(function(e){
    switch(e.keyCode) {
        case 13: // enter
        case 32: // spacebar
            doSomethingCool();
            break;
        case 27: // escape
            doSomethingElse();
            break;
    }
});
```

To be honest, if you're doing complex UI stuff with JavaScript these days, or if you've been building any responsive interfaces which rely on JavaScript, then you are most likely working with an application framework such as **Backbone**, **Angular** or **Ember**, so an abstracted and event-driven application structure will be familiar to you. It should be super easy for you to start helping out your keyboard-only users if you aren't already — just add a few more event bindings into your UI layer!

MANIPULATING THE TAB ORDER

So, you've adjusted your mindset and now you test every change to your codebase using a keyboard as well as a mouse. You've applied all your hover states to `:focus` and `:active` so you can see where you're tabbing on the page, and your interactive components react seamlessly to a mixture of mouse and keyboard commands. Feels good, right?

There's another level of optimisation to consider: manipulating the tab order. Certain DOM elements are naturally part of the tab order, and others are excluded. Links and input elements are the main elements included in the tab order, and static elements like paragraphs and headings are excluded. What if you want to make a static element 'tabbable'?

A good example would be in an expandable accordion component. Each section of the accordion should be separated by a heading, and there's no reason to make that heading into a link simply because it's interactive.

```
<div class="accordion-widget">
  <h3>Tyrannosaurus</h3>
  <p>Tyrannosaurus; meaning "tyrant lizard"...<p>

  <h3>Utahraptor</h3>
  <p>Utahraptor is a genus of theropod dinosaurs...<p>

  <h3>Dromiceiomimus</h3>
```

```
<p>Ornithomimus is a genus of ornithomimid  
dinosaurs...<p>  
</div>
```

Adding the heading elements to the tab order is trivial. We just set their `tabindex` attribute to zero. You could do this on the server or the client. I prefer to do it with JavaScript as part of the accordion setup and initialisation process.

```
$('.accordion-widget h3').attr('tabindex', '0');
```

You can apply this trick in reverse and take elements out of the tab order by setting their `tabindex` attribute to `-1`, or change the tab order completely by using other integers. This should be done with great care, if at all. You have to be sure that the markup you remove from the tab order comes out because it genuinely improves the keyboard interaction experience. This is hard to validate without user testing. The danger is that developers will try to sweep complicated parts of the UI under the carpet by taking them out of the tab order. This would be considered a **dark pattern** — at least on my team!

A FAREWELL ARIA

This is where things can get complex, and I'm no expert on the **ARIA specification**: I feel like I've only dipped my toe into this aspect of coding for accessibility. But, as with WCAG, I'd like to demystify things a little bit to encourage you to look into this area further yourself.

ARIA roles are of most benefit to screen reader users, because they modify and augment screen reader announcements.

Let's take our dinosaur accordion from the previous section. The markup is semantic, so a screen reader that can't handle JavaScript will announce all the content within the accordion, no problem.

But modern screen readers *can deal with JavaScript*, and this means that all the lovely dino information beneath each heading has probably been hidden on `document.ready`, when the accordion initialised. It might have been hidden using `display:none`, which prevents a screen reader from announcing content. If that's as far as you have gone, then you've committed an accessibility sin by hiding content from screen readers. Your user will hear a set of headings being announced, with no content in between. It would sound something like this if you were using Chromevox:

> Tyrannosaurus. Heading Three.
> Utahraptor. Heading Three.
> Dromiceiomimus. Heading Three.

We can add some ARIA magic to the markup to improve this, using the **tablist** role. Start by adding a role of **tablist** to the widget, and roles of **tab** and **tabpanel** to the headings and paragraphs respectively. Set boolean values for **aria-selected**, **aria-hidden** and **aria-expanded**. The markup could end up looking something like this.

```
<div class="accordion-widget" role="tablist">
  <!-- T-rex -->
  <h3 role="tab"
    tabindex="0"
    id="tab-2"
    aria-controls="panel-2"
    aria-selected="false">Utahraptor</h3>
  <p role="tabpanel"
    id="panel-2"
    aria-labelledby="tab-2"
    aria-expanded="false"
    aria-hidden="true">Utahraptor is a genus of theropod
    dinosaurs...</p>
  <!-- Dromiceiomimus -->
</div>
```

Now, if a screen reader user encounters this markup they will hear the following:

```
> Tyrannosaurus. Tab not selected; one of three.  
> Utahraptor. Tab not selected; two of three.  
> Dromiceiomimus. Tab not selected; three of three.
```

You could add arrow key events to help the user browse up and down the tab list items until they find one they like.

Your accordion `open()` function should update the ARIA boolean values as well as adding whatever classes and animations you have built in as standard. Your users know that unselected tabs are meant to be interacted with, so if a user triggers the open function (say, by hitting Enter or the space bar on the second item) they will hear this:

```
> Utahraptor. Selected; two of three.
```

The paragraph element for the expanded item will not be hidden by your CSS, which means it will be announced as normal by the screen reader.

This kind of thing makes so much more sense when you have a working example to play with. Again, I refer you to the fantastic resource that Hans Hillen has put together: this is his take on an **accessible accordion**, on which much of my example is based.

CONCLUSION

Getting complex interfaces right for all of your users can be difficult — there's no point pretending otherwise. And there's no substitute for user testing with real users who

navigate the web using assistive technology every day. This kind of testing can be time-consuming to recruit for and to conduct. On top of this, we now have accessibility on mobile devices to contend with. That's a huge area in itself, and it's one which I have not yet had a chance to research properly.

So, there's lots to learn, and there's lots to do to get it right. But don't be disheartened. If you have read this far then I'll leave you with one final piece of advice: **don't wait.**

Don't wait until you're building a site which mandates AAA-compliance to try this stuff out. Don't wait for a client with the will or the budget to conduct the full spectrum of user testing to come along. Unplug your mouse, and start playing with your interfaces in a new way. You'll be surprised at the things that you learn and the issues you uncover.

And the next time an true accessibility project comes along, you will be way ahead of the game.

ABOUT THE AUTHOR



Charlie Perrins is Technical Director at **Dare**. He's a front-end developer by trade, and a nut for semantic and readable code. He writes and talks about technologies old and new to anyone who'll listen. Most recently he's spoken at events run by Faber & Faber and at Front End London.

Charlie **tweets** pretty regularly, but is an unreliable blogger. His crowning achievement in self-publishing came some five years ago and was entitled simply 'The Bacon Project'.

Photo by Steve Whittington

4. Git for Grown-ups

Emma Jane Westby

24ways.org/201304

You are a clever and talented person. You create beautiful designs, or perhaps you have architected a system that even my cat could use. Your peers adore you. Your clients love you. But, until now, you haven't `*&^#!` been able to make Git work. It makes you angry inside that you have to ask your co-worker, again, for that `*&^#!` command to upload your work.

It's not you. It's Git. Promise.

Yes, this is an article about the popular version control system, Git. But unlike just about every other article written about Git, I'm not going to give you the top five commands that you need to memorize; and I'm not going to tell you all your problems would be solved if only you were using this GUI wrapper or that particular workflow. You see, I've come to a grand realization: when we teach Git, we're doing it wrong.

Let me back up for a second and tell you a little bit about the field of adult education. (Bear with me, it gets good and will leave you feeling both empowered and righteous.) Andragogy, unlike pedagogy, is a learner-driven educational experience. There are six main tenets to adult education:

1. Adults prefer to know why they are learning something.
2. The foundation of the learning activities should include experience.
3. Adults prefer to be able to plan and evaluate their own instruction.
4. Adults are more interested in learning things which directly impact their daily activities.
5. Adults prefer learning to be oriented not towards content, but towards problems.
6. Adults relate more to their own motivators than to external ones.

Nowhere in this list does it include “memorize the five most popular Git commands”. And yet this is how we teach version control: `init`, `add`, `commit`, `branch`, `push`. You’re an expert! Sound familiar? In the hierarchy of learning, memorizing commands is the lowest, or most basic, form of learning. At the peak of learning you are able to not just analyze and evaluate a problem space, but create your own understanding in relation to your existing body of knowledge.

“Fine,” I can hear you saying to yourself. “But I’m here to learn about version control.” Right you are! So how can we use this knowledge to master Git? First of all: I give you permission to use Git as a tool. A tool which you control and which you assign tasks to. A tool like a hammer, or a saw. Yes, your mastery of your tools will shape the kinds of interactions you have with your work, and your peers. But it’s yours to control. Git was written by kernel developers for kernel development. The web world has adopted Git, but it is not a tool designed for us and by us. It’s no Sass, y’know? Git wasn’t developed out of our frustration with managing CSS files in an increasingly complex ecosystem of components and atomic design. So, as you work through the next part of this article, give yourself a bit of a break. We’re in this together, and it’s going to be OK.

We’re going to do a little activity. We’re going to create your perfect Git cheatsheet.

I want you to start by writing down a list of all the people on your code team. This list may include:

- developers
- designers
- project managers
- clients

Next, I want you to write down a list of all the ways you interact with your team. Maybe you're a solo developer and you do all the tasks. Maybe you only do a few things. But I want you to write down a list of all the tasks you're actually responsible for. For example, my list looks like this:

- writing code
- reviewing code
- publishing tested code to your server(s)
- troubleshooting broken code

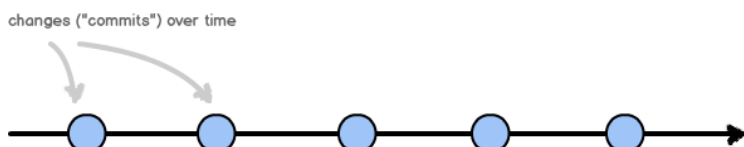
The next list will end up being a series of boxes in a diagram. But to start, I want you to write down a list of your tools and constraints. This list potentially has a lot of noun-like items and verb-like items:

- code hosting system (Bitbucket? GitHub? Unfuddle? self-hosted?)
- server ecosystem (dev/staging/live)
- automated testing systems or review gates
- automated build systems (that **Jenkins** dude people keep referring to)

Brilliant! Now you've got your actors and your actions, it's time to shuffle them into a diagram. There are many popular workflow patterns. None are inherently right or wrong; rather, some are more or less appropriate for what you are trying to accomplish.

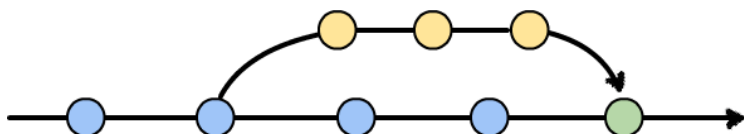
Centralized workflow

Everyone saves to a single place. This workflow may mean no version control, or a very rudimentary version control system which only ever has a single copy of the work available to the team at any point in time.



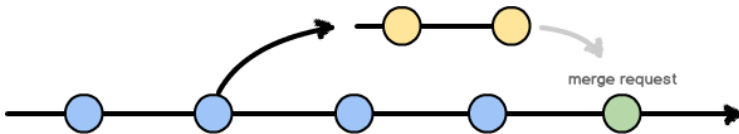
Branching workflow

Everyone works from a copy of the same place, merging their changes into the main copy as their work is completed. Think of the branches as a motorcycle sidecar: they're along for the ride and probably cannot exist in isolation of the main project for long without serious danger coming to either the driver or sidecar passenger. Branches are a fundamental concept in version control — they allow you to work on new features, bug fixes, and experimental changes within a single repository, but without forcing the changes onto others working from the same branch.



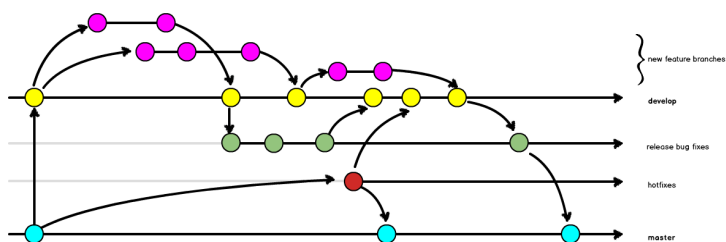
Forking workflow

Everyone works from their own, independent repository. A fork is an exact duplicate of a repository that a developer can make their own changes to. It can be kept up to date with additional changes made in other repositories, but it cannot force its changes onto another's repository. A fork is a complete repository which can use its own workflow strategies. If developers wish to merge their work with the main project, they must make a request of some kind (submit a patch, or a pull request) which the project collaborators may choose to adopt or reject. This workflow is popular for open source projects as it enforces a review process.



Gitflow workflow

A specific workflow convention which includes five streams of parallel coding efforts: master, development, feature branches, release branches, and hot fixes. This workflow is often simplified down to a few elements by web teams, but may be used wholesale by software product teams. The **original article describing this workflow** was written by Vincent Driessen back in January 2010.



But these workflows aren't about you yet, are they? So let's make the connections.

From the list of people on your team you identified earlier, draw a little circle. Give each of these circles some eyes and a smile. Now I want you to draw arrows between each of these people in the direction that code (ideally) flows. Does your designer create responsive prototypes which are pushed to the developer? Draw an arrow to represent this.

Chances are high that you don't just have people on your team, but you also have some kind of infrastructure. Hopefully you wrote about it earlier. For each of the servers and code repositories in your infrastructure, draw a square. Now, add to your diagram the relationships between the people and each of the machines in the infrastructure. Who can deploy code to the live server? How does it really get there? I bet it goes through some kind of code hosting system, such as GitHub. Draw in those arrows.

But wait!

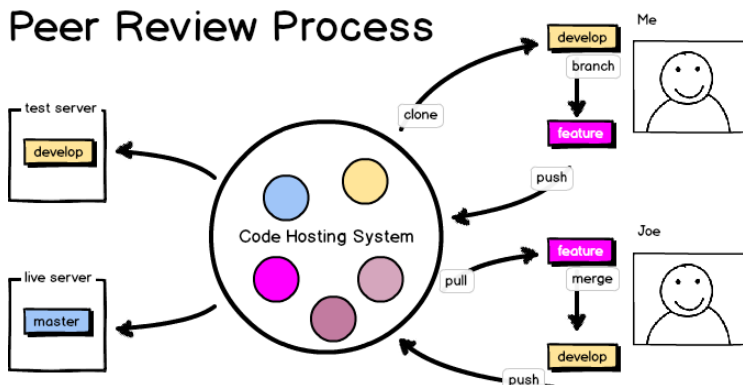
The code that's on your development machine isn't the same as the live code. This is where we introduce the concept of a branch in version control. In Git, a repository contains all of the code (sort of). A branch is a fragment of the code that has been worked on in isolation to the other branches within a repository. Often branches will have elements in common. When we compare two (or more) branches, we are asking about the difference (or *diff*) between these two slivers. Often the master branch is used on production, and the development branch is used on our dev server. The difference between these two branches is the untested code that is not yet deployed.

On your diagram, see if you can colour-code according to the branch names at each of the locations within your infrastructure. You might find it useful to make a few different copies of the diagram to isolate each of the tasks you need to perform. For example: our team has a peer review process that each branch must go through before it is merged into the shared development branch.

Finally, we are ready to add the Git commands necessary to make sense of the arrows in our diagram. If we are bringing code to our own workstation we will issue one of the following commands: `clone` (the first time we bring code to our workstation) or `pull`. Remembering that a repository contains all branches, we will issue the command `checkout` to switch from one branch to another within our own workstation. If we want to share a

particular branch with one of our team mates, we will push this branch back to the place we retrieved it from (the origin). Along each of the arrows in your diagram, write the name of the command you are going to use when you perform that particular task.

Peer Review Process



From here, it's up to you to be selfish. Before asking Git what command it would like you to use, sketch the diagram of what you want. Git is your tool, you are not Git's tool. Draw the diagram. Communicate your tasks with your team as explicitly as you can. Insist on being a selfish adult learner — demand that others explain to you, in ways that are relevant to you, how to do the things you need to do today.

ABOUT THE AUTHOR



Emma Jane Westby is an author, an educator, and a part-time beekeeper. Her latest videos, *Collaborating with Git: Crafting Workflows at the Command Line*, are now available from O'Reilly. You can follow her adventures on Twitter at [@emmajanehw](https://twitter.com/emmajanehw).

5. JavaScript: Taking Off the Training Wheels

Tom Ashworth

24ways.org/201305

JavaScript is the third pillar of front-end web development. Of those pillars, it is both the most powerful and the most complex, so it's understandable that when 24 ways asked, "What one thing do you wish you had more time to learn about?", a number of you answered "JavaScript!"

This article aims to help you feel happy writing JavaScript, and maybe even without libraries like jQuery. I can't comprehensively explain JavaScript itself without writing a book, but I hope this serves as a springboard from which you can jump to other great resources.

WHY LEARN JAVASCRIPT?

So what's in it for you? Why take the next step and learn the fundamentals?

Confidence with jQuery

If nothing else, learning JavaScript will improve your jQuery code; you'll be comfortable writing jQuery from scratch and feel happy bending others' code to your own purposes. Writing efficient, fast and bug-free jQuery is also made much easier when you have a good appreciation of JavaScript, because you can look at what jQuery is really doing. Understanding how JavaScript works lets you write better jQuery because you know what it's doing behind the scenes. When you need to leave the beaten track, you can do so with confidence.

In fact, you could say that jQuery's ultimate goal is not to exist: it was invented at a time when web APIs were very inconsistent and hard to work with. That's slowly changing as new APIs are introduced, and hopefully there will come a time when jQuery isn't needed.

An example of one such change is the introduction of the very useful `document.querySelectorAll`. Like jQuery, it converts a CSS selector into a list of matching elements. Here's a comparison of some jQuery code and the equivalent without.

```
$('.counter').each(function (index) {  
    $(this).text(index + 1);  
});  
  
var counters = document.querySelectorAll('.counter');
```

```
[].slice.call(counters).forEach(function (elem, index) {  
    elem.textContent = index + 1;  
});
```

Solving problems no one else has!

When you have to go to the internet to solve a problem, you're forever stuck reusing code other people wrote to solve a slightly different problem to your own. Learning JavaScript will allow you to solve problems in your own way, and begin to do things nobody else ever has.

Node.js

Node.js is a non-browser environment for running JavaScript, and it can do just about anything! But if that sounds daunting, don't worry: the Node community is thriving, very friendly and willing to help.

I think Node is incredibly exciting. It enables you, with one language, to build complete websites with complex and feature-filled front- and back-ends. Projects that let users log in or need a database are within your grasp, and Node has a great ecosystem of library authors to help you build incredible things. Exciting!

Here's an example web server written with Node. `http` is a module that allows you to create servers and, like jQuery's `$.ajax`, make requests. It's a small amount of

code to do something complex and, while working with Node is different from writing front-end code, it's certainly not out of your reach.

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World');
}).listen(1337);
console.log('Server running at http://localhost:1337/');
```

Grunt and other website tools

Node has brought in something of a renaissance in tools that run in the command line, like **Yeoman** and **Grunt**. Both of these rely heavily on Node, and I'll talk a little bit about Grunt here.

Grunt is a task runner, and many people use it for compiling Sass or compressing their site's JavaScript and images. It's pretty cool. You configure Grunt via the *gruntfile.js*, so JavaScript skills will come in handy, and since Grunt supports plug-ins built with JavaScript, knowing it unlocks the bucketloads of power Grunt has to offer.

WAYS TO IMPROVE YOUR SKILLS

So you know you want to learn JavaScript, but what are some good ways to learn and improve? I think the answer to that is different for different people, but here are some ideas.

Rebuild a jQuery app

Converting a jQuery project to non-jQuery code is a great way to explore how you modify elements on the page and make requests to the server for data. My advice is to focus on making it work in one modern browser initially, and then go cross-browser if you're feeling adventurous. There are many resources for directly comparing jQuery and non-jQuery code, like Jeffrey Way's [jQuery to JavaScript](#) article.

Find a mentor

If you think you'd work better on a one-to-one basis then finding yourself a mentor could be a brilliant way to learn. The JavaScript community is very friendly and many people will be more than happy to give you their time. I'd look out for someone who's active and friendly on Twitter, and does the kind of work you'd like to do. Introduce yourself over Twitter or send them an email. I wouldn't

expect a full tutoring course (although that is another option!) but they'll be very glad to answer a question and any follow-ups every now and then.

Go to a workshop

Many conferences and local meet-ups run workshops, hosted by experts in a particular field. See if there's one in your area. Workshops are great because you can ask direct questions, and you're in an environment where others are learning just like you are — no need to learn alone!

Set yourself challenges

This is one way I like to learn new things. I have a new thing that I'm not very good at, so I pick something that I think is just out of my reach and I try to build it. It's learning by doing and, even if you fail, it can be enormously valuable.

WHERE TO START?

If you've decided learning JavaScript is an important step for you, your next question may well be where to go from here.

I've collected some links to resources I know of or use, with some discussion about why you might want to check a particular site out. I hope this serves as a springboard for you to go out and learn as much as you want.

Beginner

If you're just getting started with JavaScript, I'd recommend heading to one of these places. They cover the basics and, in some cases, a little more advanced stuff. They're all reputable sources (although, I've included something I wrote — you can decide about that one!) and will not lead you astray.

- [jQuery's JavaScript 101](#) is a great first resource for JavaScript that will give you everything you need to work with jQuery like a pro.
- [Codecademy's JavaScript Track](#) is a small but useful JavaScript course. If you like learning interactively, this could be one for you.
- [HTMLDog's JavaScript Tutorials](#) take you right through from the basics of code to a brief introduction to newer technology like Node and Angular. [Disclaimer: I wrote this stuff, so it comes with a hazard warning!]
- The [tuts+ jQuery to JavaScript](#) mentioned earlier is great for seeing how jQuery code looks when converted to pure JavaScript.

Getting in-depth

For more comprehensive documentation and help I'd recommend adding these places to your list of go-tos.

- **MDN:** the Mozilla Developer Network is the first place I go for many JavaScript questions. I mostly find myself there via a search, but it's a great place to just go and browse.
- Axel Rauschmayer's **2ality** is a stunning collection of articles that will take you deep into JavaScript. It's certainly worth looking at.
- Addy Osmani's **JavaScript Design Patterns** is a comprehensive collection of patterns for writing high quality JavaScript, particularly as you (I hope) start to write bigger and more complex applications.

AND FINALLY...

I think the key to learning anything is curiosity and perseverance. If you have a question, go out and search for the answer, even if you have no idea where to start. Keep going and going and eventually you'll get there. I bet you'll learn a whole lot along the way. Good luck!

Many thanks to the people who gave me their time when I was working on this article: Tom Oakley, Jack Franklin, Ben Howdle and Laura Kalbag.

ABOUT THE AUTHOR



Tom is a front-end engineer at Twitter, working on TweetDeck. That means his consumption of Javascript is way higher than the recommended daily amount, bordering on deadly. When not working he plays the tuba in a New Orleans brass band, and enjoys taking the train between Brighton and London.

6. Levelling Up

Ashley Baxter

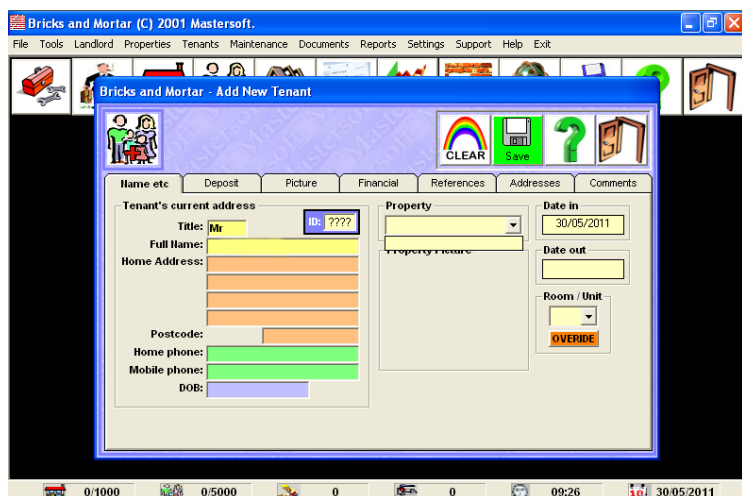
24ways.org/201306

Hello, 24 ways. I'm Ashley and I sell property insurance. I'm interrupting your Christmas countdown with an article about rental property software and a guy, Pete, who selflessly encouraged me to build my first web app. It doesn't sound at all festive, or — considering I've used both “insurance” and “rental property” — interesting, but do stick with me. There's eggnog at the end.

I run a property insurance business, **Brokers Direct**. It's a small operation, but well established. We've been selling landlord insurance on the web for over thirteen years, for twelve of which we have provided our clients with third-party software for managing their rental property portfolios. Free. Of. Charge.

It sounds like a sweet deal for our customers, but it isn't. At least, not any more. The third-party software is victim to years of neglect by its vendor. Its questionable

interface, garish visuals and, ahem, clip art icons have suffered from a lack of updates. While it was never a contender for software of the year, I've steadily grown too embarrassed to associate my business with it.



6-1. The third-party rental property software we distributed

I wanted to offer my customers a simple, clean and lightweight alternative. In an industry that's dominated by dated and bloated software, it seemed only logical that I should build my own rental property tool.

THE LONG LEARNING-TO-CODE SLOG

Learning a programming language is daunting, the source of my frustration stemming from a non-programming background. Generally, tutorials assume a degree of

familiarity with programming, whether it be tools, conventions or basic skills. I had none and, at the time, there was nothing on the web really geared towards a novice. I reached the point where I genuinely thought I was just not cut out for coding. Surrendering to my feelings of self-doubt and frustration, I sourced a local Rails developer, Pete, to build it for me.

Pete brought a pack of index cards to our meeting. Index cards that would represent each feature the rental property software would launch with.



“OK,” he began. “We’ll need a user model, tenant model, authentication, tenant and property relationships...” A dozen index cards with a dozen features lined the coffee

table in a grid-like format. Logical, comprehensible, achievable. Seeing the app laid out in a digestible manner made it seem surmountable. Maybe I *could* do this.

“I’ve been trying to learn Rails...”, I piped up.

I don’t know why I said it. I was fully prepared to hire Pete to do the hard work for me. But Pete, unprompted, gathered the index cards and neatly stacked them together, coasting them across the table towards me. “*You should build this*”.

Pete, a full-time freelance developer at the time, was turning down a paying job in favour of encouraging me to learn to code. Looking back, I didn’t realise how significant this moment was.

That evening, I took Pete’s index cards home to make a start on my app, slowly evolving each of the cards into a working feature. Building the app solo, I turned to **Stack Overflow** to solve the inevitable coding hurdles I encountered, as well as calling on a supportive Rails community. Whether they provided direct solutions to my programming woes, or simply planted a seed on how to solve a problem, I kept coding. Many months later, and after several more doubtful moments, **Lodger** was born.

The screenshot shows the 'LODGER' app interface. At the top is a red navigation bar with tabs for 'PROPERTY OVERVIEW', 'TENANTS', and 'EVENTS'. The 'PROPERTY OVERVIEW' tab is active. Below the navigation bar, the 'Property Details' section shows the address '123 Smith Street, Belfast, BT54 1GU' and a 'Edit' link. To the right are three small images: a building, an interior room, and a document icon. Below this is the 'Current Tenant(s)' section, which features a '+ NEW TENANT' button and a card for 'Jack Osborne' with his email and phone number, and an 'Edit' link. The 'Renting Agreement' section contains a table with columns for PROPERTY, START DATE, END DATE, FREQUENCY, and RENT AMOUNT. The table has one row with the following data: 123 Smith Street, Belfast, BT54 1GU; 23rd Dec 2011; 23rd Dec 2013; Monthly; £500 Due in: 3 weeks. Below the table are links for 'View Contract' and 'Edit Tenant', and summary fields for 'DEPOSIT: £500' and 'TOTAL RENT RECEIVED: £4000'. The 'Previous Tenant(s)' section lists two past tenants: 'Ashley Baxter' (4th Sept 2009 - 4th Sept 2011) and 'David Hughes' (2nd Aug 2002 - 2nd Aug 2009).

6-2. Property overview of my app, Lodger.

IF I CAN DO IT, SO CAN YOU

I misspent a lot of time building Twitter and blogging applications (apparently, all Rails tutorials centre around Twitter and blogging). If I could rewind and impart some advice to myself, this is what I'd say.

There's no magic formula

"I haven't quite grasped Rails routing. I should tackle another tutorial."

Making excuses — or procrastination — is something we are all guilty of. I was waiting for a programming book that would magically deposit a grasp of the entire Ruby syntax in my head. I kept buying books thinking each one would

be the one where it all clicked. I now have a bookshelf full of Ruby material, all of which I've barely read, and none of which got me any closer to launching my web app. Put simply, there's no magic formula.

Break it down

Whatever it is you want to build, break it down into digestible chunks. Taking Pete's method as an example, having an index card represent an individual feature helped me tremendously. Tackle one at a time. Even if each feature takes you a month to build, and you have eight features to launch with, after eight months you'll have your MVP. Remember, if you do nothing each day, it adds up to nothing.

Have a tangible product to build

I have a wonderful habit of writing down personal notes, usually to express my feelings at the time or to log an idea, only to uncover them months or years down the line, long after I forgot I had written them. I made a timely discovery while writing this article, discovering this gem while flicking through a battered Moleskine:

“I don't seem to be making good progress with learning Rails, but development still excites me. I should maybe stop doing tutorials and work towards building a specific app.”

Having a real product to work on, like I did with Lodger, means you have something tangible to apply the techniques you are learning. I found this prevented me from flitting aimlessly between tutorials and books, which is an easy area to accidentally remain in.

Team up

If possible, team up with a designer and create something together. Designers are great at presenting features in a way you'd never have considered. You will learn a lot from making their designs come to life.

YOUR HOMEWORK FOR THE HOLIDAY

Despite having a web app under my belt, I am not a programmer. I tinker with code, piecing enough bits of it together to make something functional. And that's OK! I'm not excusing sloppiness, but if we aimed for perfection every time, we'd never execute any of our ideas.

As the holidays approach and you've exhausted yet another viewing of The Muppet Christmas Carol (or is that just my guilty pleasure at Christmas?), you may have time on your hands. Time to explore an idea you've been sitting on, but — plagued with procrastination and doubt — have yet to bring to life. This holiday, I am here to say to you what Pete said to me.

You should build this.

You don't need to be the next Mark Zuckerberg or Larry Page. You just have to learn enough to get it done.

PS: I lied about the egnogg, but try capturing somebody's attention when you tell them you sell property insurance!

ABOUT THE AUTHOR



An unusual combination of insurer and photographer, Ashley's big girl's job is running insurance broker **Brokers Direct**. She has also been known to photograph weddings under the **Girl With A**

Camera alias. Ashley reserves what little leisure time she has for Xbox, weightlifting and convincing people that working in insurance isn't as mundane as it sounds.

Ashley tweets as **@iamashley** and prefers dogs over cats.

7. Animating Vectors with SVG

Brian Suda

24ways.org/201307

It is almost 2014 and fifteen years ago the W3C started to develop a web-based scalable vector graphics (SVG) format. As web technologies go, this one is pretty old and well entrenched.

See the Pen [yJf1C](#) by Drew McLellan (@drewm) on CodePen

Embed not working on your device? [Try direct.](#)

Unlike rasterized images, SVG files will stay crisp and sharp at any resolution. With high-DPI phones, tablets and monitors, all those rasterized icons are starting to look a bit old and blocky. There are several options to get simpler, decorative pieces to render smoothly and respond to various device widths, shapes and sizes. Symbol fonts are one option; the other is SVG.

I'm a big fan of SVG. SVG is an XML format, which means it is possible to write by hand or to script. The most common way to create an SVG file is through the use of various drawing applications like Illustrator, Inkscape or Sketch. All of them open and save the SVG format.

But, if SVG is so great, why doesn't it get more attention?

The simple answer is that for a long time it wasn't well supported, so no one touched the technology. SVG's adoption has always been hampered by browser support, but that's not the case any more. Every modern browser (at least three versions back) supports SVG. Even IE9.

Although the browsers support SVG, it is implemented in many different ways.

SVG IN HTML

Some browsers allow you to embed SVG right in the HTML: the `<svg>` element. Treating SVG as a first-class citizen works — sometimes. Another way to embed SVG is via the `` element; using the `src` attribute, you can refer to an SVG file. Again, this only works sometimes and leaves you in a tight space if you need to have a fallback for older browsers. The most common solution is to use the `<object>` element, with the `data` attribute referencing the SVG file. When a browser does not support this, it falls back to the content inside the `<object>`. This could be a rasterized fallback ``. This method gets you the best

of both worlds: a nice vector image with an alternative rasterized image for browsers that don't support SVG. The downside is that you need to manage both formats, and some browsers will download both the SVG and the rasterized version, becoming a performance problem.

Alexey Ten came up with a **brilliant little trick** that uses inline SVG combined with an SVG `<image>` element. This has an SVG href pointing to the vector SVG representation and a src attribute to the rasterized version. Older browsers will rewrite the `<image>` element as `` and use the rasterized src attribute, but modern browsers will show the vector SVG.

```
<svg width="96" height="96">
  <image xlink:href="svg.svg" src="svg.png" width="96"
height="96"/>
</svg>
```

It is a great workaround for most situations. You will have to determine the browsers you want or need to support and consider performance issues to decide which method is best for you.

SO IT CAN BE USED IN HTML. WHY?

There are two compelling reasons why vector graphics in the form of icons and symbols are going to be important on the web. With higher resolution screens, going from 72dpi to 200, 300, even over 400dpi, your rasterized

icons are looking a little too blocky. As we zoom and print, we expect the visuals on the site to also stay smooth and crisp.

The other main reason vector graphics are useful is scaling. As responsive websites become the norm, we need a way to dynamically readjust the heights, widths and styles of various elements. SVG handles this perfectly, since vectors remain smooth when changing size.

SVG files are text-based, so they're small and can be zipped nicely. There are also techniques for creating SVG sprites to further squeeze out performance gains. But SVG really shines when you begin to couple it with JavaScript. Since SVG elements are part of the DOM, they can be interacted with just like any other element you are used to.

The folks at **Vox Media** had an ingenious little trick with their SVG for a **Playstation** and **Xbox One** reviews. I've used the same technique for the 24 ways example. Vox Media spent a lot of time creating SVG line art of the two consoles, but once in place the artwork scaled and resized beautifully.

They still had another trick up their sleeves. In their example, they knew each console was line art, so they used SVG's line dash property to simulate the lines being drawn by animating the growth of the line by small percentage increments until the lines were complete.

This is a great example of a situation where the alternatives wouldn't be as straightforward to implement. Using an animated GIF would create a heavy file since it would need to contain all the frames of the animation at a large size to permit scaling; even then, smooth aliasing would be lost. canvas and plenty of JavaScript would be another alternative, but this is a rasterized format. It would need be redrawn at each scale, which is certainly possible, but smoothness would be lost when zooming or printing.

The HTML, SVG and JavaScript for this example is less than 4KB! Let's have a quick look at the code:

```
<script>
var current_frame = 0;
var total_frames = 60;
var path = new Array();
var length = new Array();
for(var i=0; i<4;i++){
    path[i] = document.getElementById('i'+i);
    l = path[i].getTotalLength();
    length[i] = l;
    path[i].style.strokeDasharray = l + ' ' + l;
    path[i].style.strokeDashoffset = l;
}
var handle = 0;

var draw = function() {
    var progress = current_frame/total_frames;
    if (progress > 1) {
        window.cancelAnimationFrame(handle);
```

```

    } else {
      current_frame++;
      for(var j=0; j<path.length;j++){
        path[j].style.strokeDashoffset =
Math.floor(length[j] * (1 - progress));
      }
      handle = window.requestAnimationFrame(draw);
    }
  };
  draw();
</script>

```

First, we need to initialize a few variables to set the current frame, the number of frames, how fast the animation will run, and we get each of the paths based on their IDs. With those paths, we set the dash and dash offset.

```

path[i].style.strokeDasharray = 1 + ' ' + 1;
path[i].style.strokeDashoffset = 1;

```

We start the line as a dash, which effectively makes it blank or invisible.

Next, we move to the `draw()` function. This is where the magic happens. We want to increment the frame to move us forward in the animation and check it's not finished. If it continues, we then take a percentage of the distance based on the frame and then set the dash offset to this new percentage. This gives the illusion that the line is being drawn. Then we have an animation callback, which starts the draw process over again.

That's it! It will work with any SVG `<path>` element that you can draw.

LIBRARIES TO GET YOU STARTED

If you aren't sure where to start with SVG, there are several libraries out there to help. They also abstract all browser compatibility issues to make your life easier.

- Raphaël
- Snap.svg
- svg.js

You can also get most vector applications to export SVG. This means that you can continue your normal workflows, but instead of flattening the image as a PNG or bringing it over to Photoshop to rasterize, you can keep all your hard work as vectors and reap the benefits of SVG.

ABOUT THE AUTHOR



Brian Suda is a master informatician working to make the web a better place little by little everyday. Since discovering the Internet in the mid-90s, Brian Suda has spent a good portion of each day connected to it. His own little patch of Internet is <http://suda.co.uk>, where many of his past projects and crazy ideas can be found.

Photo: Jeremy Keith

8. Kill It With Fire! What To Do With Those Dreaded FAQs

Lisa Maria Martin

24ways.org/201308

In the mid-1640s, a man named Matthew Hopkins attempted to rid England of the devil's influence, primarily by demanding payment for the service of tying women to chairs and tossing them into lakes.

Unsurprisingly, his methods garnered criticism. Hopkins defended himself in *The Discovery of Witches* in 1647, subtitled "Certaine *Queries* answered, which have been and are likely to be objected against MATTHEW HOPKINS, in his way of finding out *Witches*."

Each "querie" was written in the voice of an imagined detractor, and answered in the voice of an imagined defender (always referring to himself as "the discoverer," or "him"):

Quer. 14.

All that the witch-finder doth is to fleece the country of their money, and therefore rides and goes to townes to have employment, and promiseth them faire promises, and it may be doth nothing for it, and possesseth many men that they have so many wizzards and so many witches in their towne, and so hartens them on to entertaine him.

Ans.

You doe him a great deale of wrong in every of these particulars.

Hopkins' self-defense was an early modern English FAQ.

DIGITAL BEGINNINGS

Question and answer formatting certainly isn't new, and stretches back much further than witch-hunt days. But its most modern, most notorious, most reviled incarnation is the internet's frequently asked questions page.

FAQs began showing up on pre-internet mailing lists as a way for list members to answer and pre-empt newcomers' repetitive questions:

The presumption was that new users would download archived past messages through ftp. In practice, this rarely happened and the users tended to post questions to the mailing list instead of searching its archives. Repeating the “right” answers becomes tedious...

When all the users of a system can hear all the other users, FAQs make a lot of sense: the conversation needs to be managed and manageable. FAQs were a stopgap for the technological limitations of the time.

But the internet moved past mailing lists. Online information can be stored, searched, filtered, and muted; we choose and control our conversations. New users no longer rely on the established community to answer their questions for them.

And yet, FAQs are still around. They’re a content anti-pattern, replicated from site to site to solve a problem we no longer have.

WHAT WE HATE WHEN WE HATE FAQs

As someone who creates and structures online content – always with the goal of making that content as useful as possible to people – FAQs drive me absolutely batty. Almost universally, FAQs represent the opposite of useful. A brief list of their sins:

1.

Double trouble

Duplicated content is practically a given with FAQs. They're written as though they'll be accessed in a vacuum – but search results, navigation patterns, and curiosity ensure that users will seek answers throughout the site. Is our goal to split their focus? To make them uncertain of where to look? To divert them to an isolated microcosm of the website? Duplicated content means user confusion (to say nothing of the duplicated workload for maintaining content).

2.

Leaving the job unfinished

Many FAQs fail before they're even out of the gate, presenting a list of questions that's incomplete (too short and careless to be helpful) or irrelevant (avoiding users' real concerns in favor of soundbites). Alternately, if the right questions are there, the answers may be convoluted, jargon-heavy, or otherwise difficult to understand.

3.

Long lists of not-my-question

Getting a single answer often means sifting through a haystack of questions. For each potential question, the

user must read, comprehend, assess, move on, rinse, repeat. That's a lot of legwork for little reward – and a lot of opportunity for mistakes. Users may miss their question, or they may fail to recognize a differently worded version of their question, or they may not notice when their sought-after answer appears somewhere they didn't expect.

4.

The ventriloquist act

FAQs shift the point of view. While websites speak on behalf of the organization (“our products,” “our services,” “you can call us for assistance,” etc.), FAQs speak as the user – “I can't find my password” or “How do I sign up?” Both voices are written from the first-person perspective, but speak for different entities, which is disorienting: it breaks the tone and messaging across the website. It's also presumptuous: why do you get to speak for the user?

These all underscore FAQs' fatal flaw: they are content without context, delivered without regard for the larger experience of the website. You can hear the absurdity in the name itself: if users are asking the same questions so frequently, then there is an obvious gulf between their needs and the site content. (And if not, then we have a labeling problem.)

Instead of sending users to a jumble of maybe-it's-here-maybe-it's-not questions, the answers to FAQs should be found naturally throughout a website. They are not separated, not isolated, not other. They are the content.

To present it otherwise is to create a runaround, and users know it. Jay Martel's parody, "F.A.Q.s about F.A.Q.s" captures the silliness and frustration of such a system:

Q: Why are you so rude?

A: For that answer, you would have to consult an F.A.Q.s about F.A.Q.s about F.A.Q.s. But your time might be better served by simply abandoning your search for a magic answer and taking responsibility for your own profound ignorance.

FAQs aren't magic answers. They don't resolve a content dilemma or even help users. Yet they keep cropping up, defiant, weedy, impossible to eradicate.

WHERE ARE THEY ALL COMING FROM?

Blame it on this: writing is hard. When generating content, most of us do whatever it takes to get some words on the screen. And the format of question and answer makes it easy: a reactionary first stab at content development.

After all, the point of website content is to answer users' questions. So this – to give everyone credit – is a really good move. Content creators who think in terms of questions and answers are actually thinking of their users, particularly first-time users, trying to anticipate their needs and write towards them.

It's a good start. But it's scaffolding: writing that helps you get to the writing you're supposed to be doing. It supports you while you write your way to the heart of your content. And once you get there, you have to look back and take the scaffolding down.

Leaving content in the Q&A format that helped you develop it is missing the point. You're not there to build scaffolding. You have to see your content in its naked purpose and determine the best method for communicating that purpose – and it usually won't be what got you there.

The goal (to borrow a lesson from content management systems) is to separate the content from its presentation, to let the meaning of the content inform its display.

This is, of course, a nice theory.

AN OCCASIONALLY NECESSARY EVIL

I have a lot of clients who adore FAQs. They've developed their content over a long period of time. They've listened to the questions their users are asking. And they've answered them all on a page that I simply cannot get them to part with.

Which means I've had to consider that there may be occasions where an FAQ page is appropriate.

As an example: one of my clients is a financial office in a large institution. Because this office manages several third-party systems that serve a range of niche audiences, they had developed FAQs that addressed hyper-specific instances of dysfunction within systems for different users – à la *"I'm a financial director and my employee submitted an expense report in such-and-such system and it returned such-and-such error. What do I do?"*

Yes, this content could be removed from the question format and rewritten. But I'm not sure it would be an improvement. It won't necessarily resolve concerns about length and searchability, and the different audiences may complicate the delivery. And since the work of rewriting it didn't fit into the client workflow (small team, no writers, pressed for time), I didn't recommend the change.

I've had to make peace with not being to torch all the FAQs on the internet. Some content, like troubleshooting information or complex procedures, may be better in that format. It may be the smartest way for a particular client to handle that particular information.

Of course, this has to be determined on a case-by-case basis, taking into account the amount of content, the subject matter, the skill levels of the content creators, the publishing workflow, and the search habits of the users.

If you determine that an FAQ page is the only way to go, ask yourself:

- Is there a better label or more specific term for the page (support, troubleshooting, product concerns, etc.)?
- Is there way to structure the page, categorize the questions, or otherwise make it easier for users to navigate quickly to the answer they need?
- Is a question and answer format absolutely the best way to communicate this information?

FORM FOLLOWS FUNCTION

Just as a question and answer format isn't necessarily required to deliver the content, neither is it an inappropriate method in and of itself. Content professionals have developed a knee-jerk reaction: *It's an FAQ page! Quick, burn it! Buuuurn it!*

But there's no inherent evil in questions and answers. Framing content in an interrogatory construct is no more a deal with the devil than subheads and paragraphs, or narrative arcs, or bullet points.

Yes, FAQs are riddled with communication snafus. They deserve, more often than not, to be tied to a chair and thrown into a lake. But that wouldn't fix our content problems. FAQs are a shiny and obvious target for our frustration, but they're not unique in their flaws. In any format, in any display, in any kind of page, weak content can rear its ugly, poorly written head.

It's not the Q&A that's to blame, it's bad content. Content without context will always fail users. That's the real witch in our midst.

ABOUT THE AUTHOR



Lisa Maria Martin is a content strategist, information architect, and writer in Washington, DC. In her previous lives, she's been a copywriter, a designer, a journalist, and a lecturer at several universities, extolling the virtues of MLA citation and Oxford commas. When she isn't making an honest living, her writing shows up in lit journals like *Pleiades*, *Puerto del Sol*, *The Indiana Review*, and others. You can read Lisa Maria's every-blue-moon blog at thefutureislikepie.com, or follow her on Twitter @redsesame (but she mostly tweets about doges and Star Trek. Fair warning).

9. Keeping Parts of Your Codebase Private on GitHub

Harry Roberts

24ways.org/201309

Open source is brilliant, there's no denying that, and GitHub has been instrumental in open source's recent success. I'm a keen open-sourcerer myself, and I have a number of projects on GitHub. However, as great as sharing code is, we often want to keep *some* projects to ourselves. To this end, GitHub created private repositories which act like any other Git repository, only, well, private!

A slightly less common issue, and one I've come up against myself, is the desire to only keep *certain parts* of a codebase private. A great example would be my site, **CSS Wizardry**; I want the code to be open source so that people can poke through and learn from it, but I want to

keep any draft blog posts private until they are ready to go live. Thankfully, there is a very simple solution to this particular problem: using multiple remotes.

Before we begin, it's worth noting that you can actually build a **GitHub Pages** site from a private repo. You can keep the entire source private, but still have GitHub build and display a full Pages/Jekyll site. I do this with csswizardry.net. This post will deal with the more specific problem of keeping only certain parts of the codebase (branches) private, and expose parts of it as either an open source project, or a built GitHub Pages site.

N.B. This post requires some basic Git knowledge.

ADDING YOUR PUBLIC REMOTE

Let's assume you're starting from scratch and you currently have no repos set up for your project. (If you do already have your public repo set up, skip to the "Adding your private remote" section.)

So, we have a clean slate: nothing has been set up yet, we're doing all of that now. On GitHub, create two repositories. For the sake of this article we shall call them *site.com* and *private.site.com*. Make the *site.com* repo public, and the *private.site.com* repo private (you will need a paid GitHub account).

On your machine, create the *site.com* directory, in which your project will live. Do your initial work in there, commit some stuff — whatever you need to do. Now we need to link this *local* Git repo on your machine with the *public* repo (remote) on GitHub. We should all be used to this:

```
$ git remote add origin  
git@github.com:[user]/site.com.git
```

Here we are simply telling Git to add a remote called *origin* which lives at `git@github.com:[user]/site.com.git`. Simple stuff. Now we need to push our current branch (which will be *master*, unless you've explicitly changed it) to that remote:

```
$ git push -u origin master
```

Here we are telling Git to push our *master* branch to a corresponding *master* branch on the remote called *origin*, which we just added. The `-u` sets upstream tracking, which basically tells Git to always shuttle code on this branch between the local *master* branch and the *master* branch on the *origin* remote. Without upstream tracking, you would have to tell Git where to push code to (and pull it from) every time you ran the `push` or `pull` commands. This sets up a permanent bond, if you like.

This is really simple stuff, stuff that you will probably have done a hundred times before as a Git user. Now to set up our private remote.

ADDING YOUR PRIVATE REMOTE

We've set up our public, open source repository on GitHub, and linked that to the repository on our machine. All of this code will be publicly viewable on GitHub.com. (Remember, GitHub is just a host of regular Git repositories, which also puts a nice GUI around it all.) We want to add the ability to keep certain parts of the codebase private. What we do now is add another remote repository to the same local repository. We have two repos on GitHub (*site.com* and *private.site.com*), but only one repository (and, therefore, one directory) on our machine. Two GitHub repos, and one local one.

In your local repo, check out a new branch. For the sake of this article we shall call the branch *dev*. This branch might contain work in progress, or draft blog posts, or anything you don't want to be made publicly viewable on GitHub.com. The contents of this branch will, in a moment, live in our private repository.

```
$ git checkout -b dev
```

We have now made a new branch called *dev* off the branch we were on last (*master*, unless you renamed it).

Now we need to add our private remote (*private.site.com*) so that, in a second, we can send *this* branch to *that* remote:

Keeping Parts of Your Codebase Private on GitHub

```
$ git remote add private  
git@github.com:[user]/private.site.com.git
```

Like before, we are just telling Git to add a new remote to this repo, only this time we've called it `private` and it lives at `git@github.com:[user]/private.site.com.git`. We now have one local repo on our machine which has two remote repositories associated with it.

Now we need to tell our `dev` branch to push to our private remote:

```
$ git push -u private dev
```

Here, as before, we are pushing some code to a repo. We are saying that we want to push the `dev` branch to the private remote, and, once again, we've set up upstream tracking. This means that, by default, the `dev` branch will only push and pull to and from the private remote (unless you ever explicitly state otherwise).

Now you have two branches (`master` and `dev` respectively) that push to two remotes (`origin` and `private` respectively) which are public and private respectively.

Any work we do on the `master` branch will push and pull to and from our publicly viewable remote, and any code on the `dev` branch will push and pull from our private, hidden remote.

ADDING MORE BRANCHES

So far we've only looked at two branches pushing to two remotes, but this workflow can grow as much or as little as you'd like. Of course, you'd never do all your work in only two branches, so you might want to push any number of them to either your public or private remotes. Let's imagine we want to create a branch to try something out real quickly:

```
$ git checkout -b test
```

Now, when we come to push this branch, we can choose which remote we send it to:

```
$ git push -u private test
```

This pushes the new test branch to our private remote (again, setting the persistent tracking with `-u`).

You can have as many or as few remotes or branches as you like.

COMBINING THE TWO

Let's say you've been working on a new feature in private for a few days, and you've kept that on the private remote. You've now finalised the addition and want to move it into your public repo. This is just a simple merge. Check out your master branch:

```
$ git checkout master
```

Then merge in the branch that contained the feature:

```
$ git merge dev
```

Now `master` contains the commits that were made on `dev` and, once you've pushed `master` to its remote, those commits will be viewable publicly on GitHub:

```
$ git push
```

Note that we can just run `$ git push` on the `master` branch as we'd previously set up our upstream tracking (`-u`).

MULTIPLE MACHINES

So far this has covered working on just one machine; we had two GitHub remotes and one local repository. Let's say you've got yourself a new Mac (yay!) and you want to clone an existing project:

```
$ git clone git@github.com:[user]/site.com.git
```

This **will not** clone any information about the remotes you had set up on the previous machine. Here you have a fresh clone of the public project and you will need to add the private remote to it again, as above.

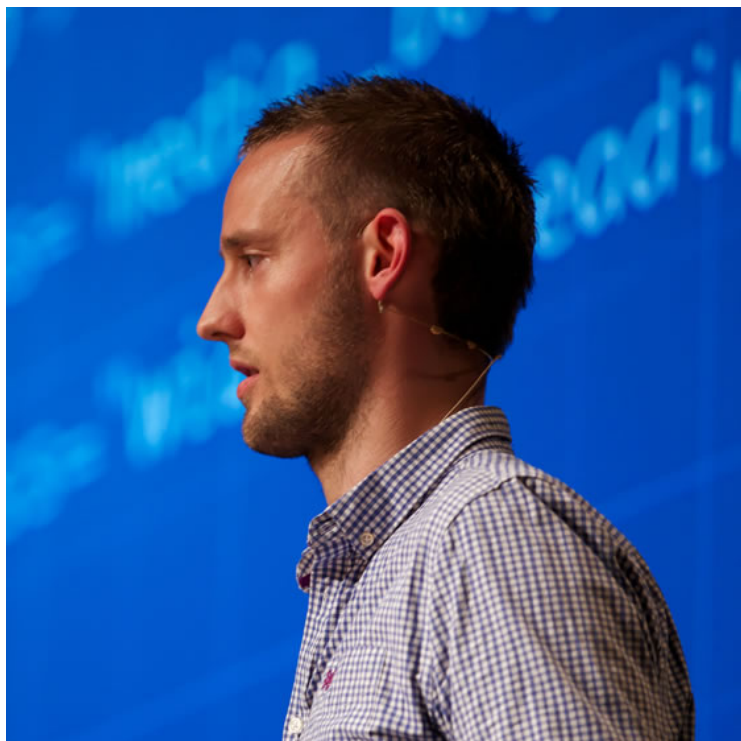
DONE!

If you'd like to see me blitz through all that in one go, check the [showterm](#) recording.

The beauty of this is that we can still share our code, but we don't have to develop quite so openly all of the time. Building a framework with a killer new feature? Keep it in a private branch until it's ready for merge. Have a blog post in a Jekyll site that you're not ready to make live? Keep it in a private drafts branch. Working on a new feature for your personal site? Tuck it away until it's finished. Need a staging area for a Pages-powered site? Make a staging remote with its own **custom domain**.

All this boils down to, really, is the fact that you can bring multiple remotes together into one local codebase on your machine. What you do with them is entirely up to you!

ABOUT THE AUTHOR



Harry is a Consultant Front-end Architect, designer, developer, writer and speaker from the UK—previously a Senior Developer at BSkyB, he now helps tech teams all over the world in building better front-ends. He Tweets at [@csswizardry](#).

He specialises in authoring and scaling large front-ends. He writes on the subjects of maintainability, architecture, performance, OOCSS and more at [csswizardry.com](#). He is the lead and sole developer of [inuit.css](#), a powerful, scalable, Sass-based, BEM, OOCSS framework.

10. Why Bother with Accessibility?

Laura Kalbag

24ways.org/201310

Web accessibility (known in other fields as inclusive design or universal design) is the degree to which a website is available to as many people as possible. Accessibility is most often used to describe how people with disabilities can access the web.

HOW WE APPROACH ACCESSIBILITY

In the web community, there's a surprisingly inconsistent approach to accessibility. There are some who are endlessly dedicated to accessible web design, and there are some who believe it so intrinsic to the web that it shouldn't be considered a separate topic. Still, of those who are familiar with accessibility, there's an overwhelming number of designers, developers, clients and bosses who just aren't that bothered.

Over the last few months I've spoken to a lot of people about accessibility, and I've heard the same reasons to ignore it over and over again. Let's take a look at the most common excuses.

EXCUSE 1: "PEOPLE WITH DISABILITIES DON'T REALLY USE THE WEB"

Accessibility will make your site available to more people — the inclusion case

In the same way that the accessibility of a building isn't just about access for wheelchair users, web accessibility isn't just about blind users and screen readers. We can affect positively the lives of many people by making their access to the web easier.

There are four main types of disability that affect use of the web:

Visual

Blindness, low vision and colour-blindness

Auditory

Profoundly deaf and hard of hearing

Motor

The inability to use a mouse, slow response time, limited fine motor control

Cognitive

Learning difficulties, distractibility, the inability to focus on large amounts of information

None of these disabilities are completely black and white

Examining deafness, it's clear from the medical scale that there are many grey areas between full hearing and total deafness:

- mild
- moderate
- moderately severe
- severe
- profound
- totally deaf

For eyesight, and brain conditions that affect what users see, there is a huge range of conditions and challenges:

- astigmatism
- colour blindness
- akinetopsia (motion blindness)
- scotopic visual sensitivity (visual stress related to light)
- visual agnosia (impaired recognition or identification of objects)

While we might have medical and government-recognised definitions that tell us what makes a disability, day-to-day life is not so straightforward. People experience varying degrees of different conditions, and often one or more conditions at a time, creating a false divide when you view disability in terms of us and them.

Impairments aren't always permanent

As we age, we're more likely to experience different levels of visual, auditory, motor and cognitive impairments. We might have an accident or illness that affects us temporarily. We might struggle more earlier or later in the day. There are so many little physiological factors that affect the way people interact with the web that we can't afford to make any assumptions based on our own limited experiences.

Impairments might be somewhere between the user and the website

There are also impairments that aren't directly related to the user. Environmental factors have a huge effect on the way people interact with the web. These could be:

- Low bandwidth, or intermittent internet connection
- Bright light, rain, or other weather-based conditions
- Noisy environments, or a location where the user doesn't want to disturb their neighbours with sound

- Browsing with mobile devices, games consoles and other non-desktop devices
- Browsing with legacy browsers or operating systems

Such environmental factors show that it's not just those with physical impairments who benefit from more accessible websites. We started designing responsive websites so we could be more **future-friendly**, and with a shared goal of better optimised experiences, accessibility should be at the core of responsive web design.

EXCUSE 2: “WE DON’T WANT TO AFFECT THE EXPERIENCE FOR THE MAJORITY OF OUR USERS”

Accessibility will improve your site for all your users — the usability case

On a basic level, the different disability groups, as shown in the inclusion case, equate to simple usability goals:

- **Visual** – make it easy to read
- **Auditory** – make it easy to hear
- **Motor** – make it easy to interact
- **Cognitive** – make it easy to understand and focus

Taking care to ensure good usability in these areas will also have an impact on accessibility. Unless your site is catering specifically to a particular disability, where

extreme optimisation is most beneficial, taking care to design with accessibility in mind will rarely negatively affect the experience of your wider audience.

EXCUSE 3: “WE DON’T HAVE THE BUDGET FOR ACCESSIBILITY”

Accessibility will make you money – the business case

By reducing your audience through ignoring accessibility, you’re potentially excluding the income from those users. Designing with accessibility in mind from the beginning of a project makes it easier to make small inexpensive optimisations as part of the design and development process, rather than bolting on costly updates to increase your potential audience later on.

The following are excerpts from a white paper about companies that increased the accessibility of their websites to comply with government regulation.

Improvements in accessibility **doubled** Legal and General’s life insurance sales online.

Improvements in accessibility increased Tesco's grocery home delivery sales by £13 million in 2005... To their surprise they found that many normal visitors preferred the ease of navigation and improved simplicity of the [parallel] accessible site and switched to use it. Tesco have replaced their 'normal' site with their accessible version and expect a further increase in revenues.

Improvements in accessibility increased Virgin.net sales by 68%.

Statistics all from WSI white paper: Improve your website's usability and accessibility to increase sales (PDF).

EXCUSE 4: "ACCESSIBLE WEBSITES ARE UGLY"

Accessibility won't stop your site from being beautiful — the beauty case

Many people use ugly accessible websites as proof that all accessible websites are ugly. This just isn't the case. I've compiled some examples of beautiful and accessible websites with screenshots of how they look through the Color Oracle simulator and how they perform when run through Webaim's Wave accessibility checker tool.

Why Bother with Accessibility?


While automated tools are no substitute for real users, they can help you learn more about good practices, and give you guidance on where your site needs improvements to make it more accessible.

AMAZON.CO.UK

It may not be a decorated beauty, but Amazon is often first in functional design. It's a huge website with a lot of interactive content, but it generates just five errors on the Wave test, and is easy to read under a Color Oracle filter.

The screenshot shows the Amazon.co.uk homepage during Black Friday. The top navigation bar includes links for 'Your Amazon.co.uk', 'Today's Deals', 'Gift Cards', 'Sell', and 'Help'. A search bar is located below the navigation. The main content area features several promotional banners: a Sony banner for a chance to win up to \$10,000, a Kindle Fire HD deal, a Black Friday banner for hundreds of great deals, and a 'Scarves Unlimited' banner. The 'Digital Cameras Bestsellers' section is also visible, featuring Nikon and Canon cameras. The page is cluttered with various promotional elements and navigation links.

10-1. Screenshot of Amazon website







 **WAVE**
web accessibility evaluation tool

www.amazon.co.uk →




Styles No Styles Contrast

Summary

WAVE has detected the following:

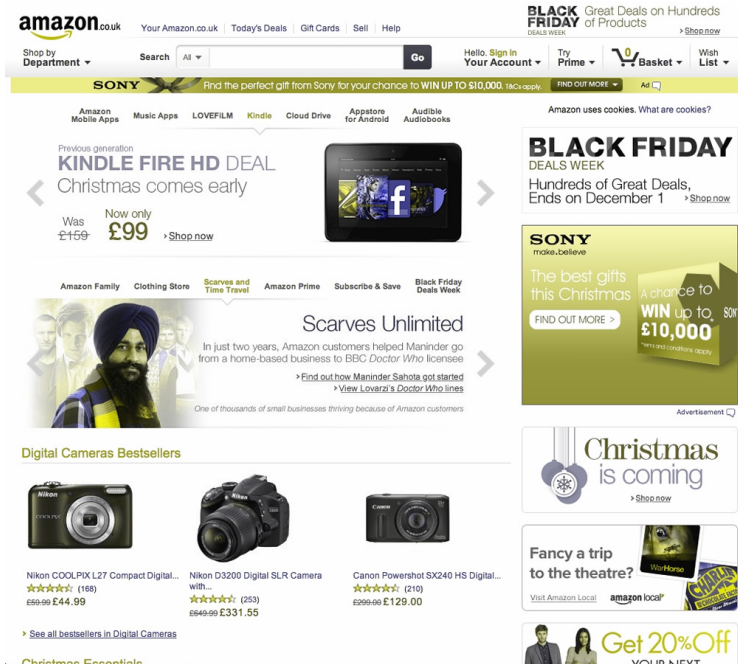
-  **5 Errors**
-  **16 Alerts**
-  **40 Features**
-  **48 Structural Elements**
-  **4 HTML5 and ARIA**
-  **27 Contrast Errors**

Panel Options

-  **DETAILS:** A listing of all the WAVE icons in your page.
-  **DOCUMENTATION:** Explanation of the WAVE icons and how you can make your page more accessible.
-  **OUTLINE:** The heading structure of the web page.

10-2. Screenshot of Amazon's Wave results – five errors

Why Bother with Accessibility?



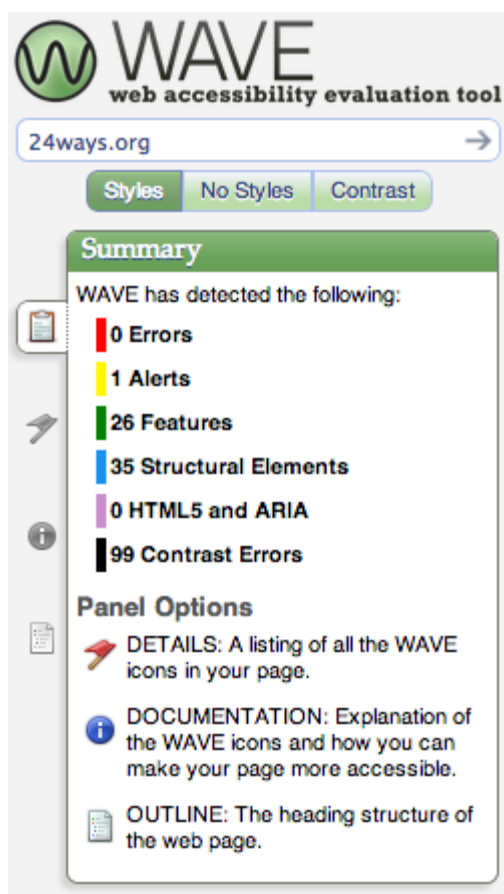
10-3. Screenshot of Amazon through a Color Oracle filter

24 WAYS

When Tim Van Damme redesigned 24 ways back in 2007, it was a striking and unusual design that showed what could be achieved with CSS and some imagination. Despite the complexity of the design, it gets an outstanding zero errors on the Wave test, and is still readable under a Color Oracle filter.



10-4. Screenshot of pre-2013 24 ways website design



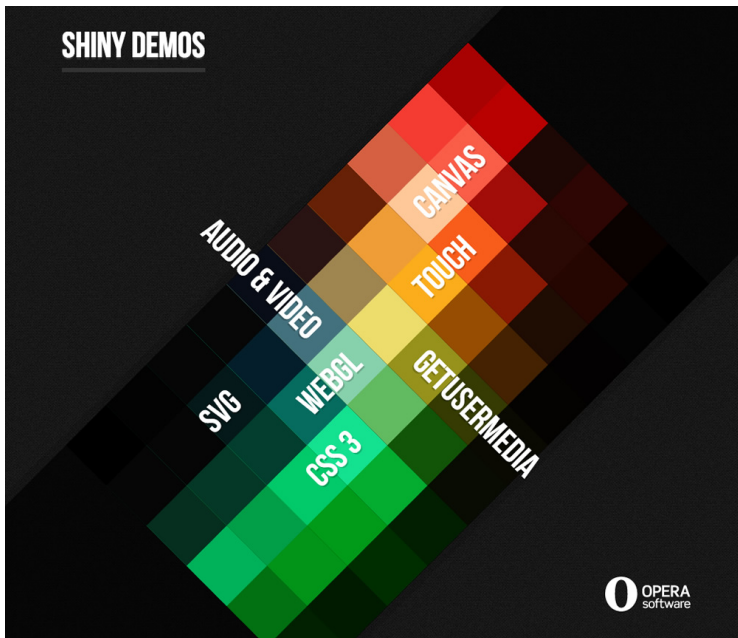
10-5. Screenshot of 24 ways Wave results – zero errors



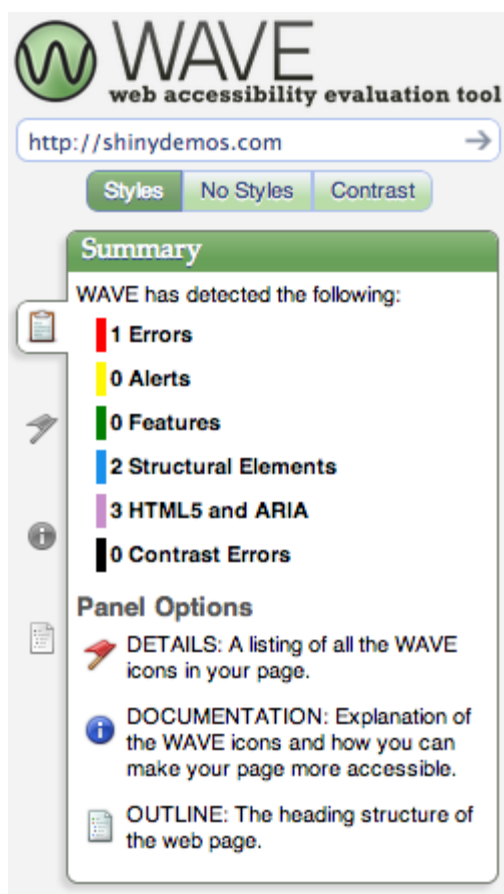
10-6. Screenshot of 24ways through a Color Oracle filter

OPERA'S SHINY DEMOS

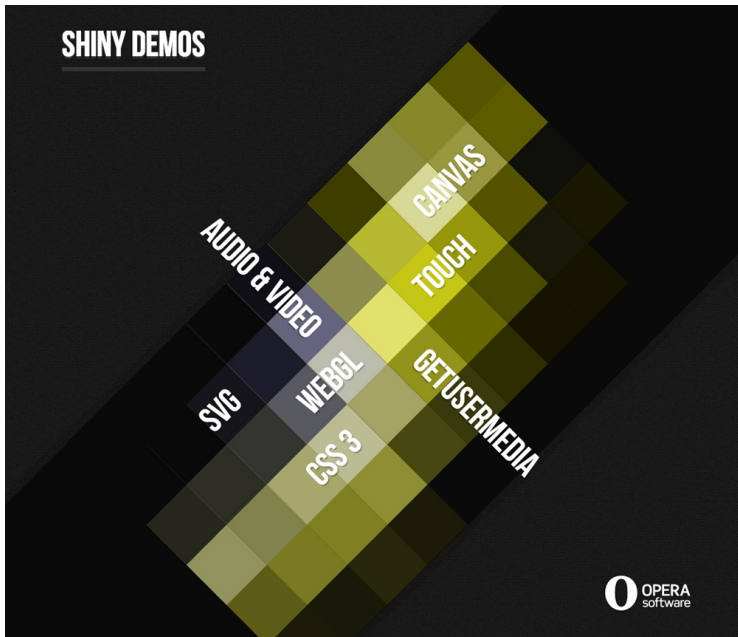
Demos and prototypes are notorious for ignoring accessibility, but Opera's Shiny Demos site shows how exploring new technologies doesn't have to exclude anyone. It only gets one error on the Wave test, and looks fine under a Color Oracle filter.



10-7. Screenshot of Opera's Shiny Demos website



10-8. Screenshot of Opera's Shiny Demos Wave results – 1 error



10-9. Screenshot of Opera's Shiny Demos through a Color Oracle filter

SOUNDCLOUD

When a site is more app-like, relying on more interaction from the user, accessibility can be more challenging. However, SoundCloud only gets one error on the Wave test, and the colour contrast holds up well under a Color Oracle filter.

[Home](#)
[Explore](#)

[Sign in or](#)
[Sign up](#)

Trending Music

Trending Audio

[Classical](#)
[Country](#)
[Electronic](#)
[Hip Hop](#)
[Jazz](#)
[Metal](#)
[Pop](#)
[R&B](#)
[Reggae](#)
[Rock](#)
[World](#)
[Audiobooks](#)
[Business](#)
[Comedy](#)
[Entertainment](#)
[Learning](#)
[News & Politics](#)
[Religion & Spirituality](#)
[Science](#)
[Sports](#)
[Storytelling](#)
[Technology](#)

Hear what's up and coming on SoundCloud

Century Media Records

DARK FUNERAL - Feed On The Mortals

3 days

metal

Like

Repost

Add to playlist

Share

Buy

17,048

410

89

74

Ab Jo

SomeYouGet (SuperLowKeyBustaRhymesRemix)

2 days

Remix

Like

Repost

Add to playlist

Share

30,617

1,557

443

107

khadisma

hi,howyadoin

3 days

Like

Repost

Add to playlist

Share

29,347

1,591

419

94

G-Eazy

Far Alone (feat. Jay Ant)

6 days

Hip Hop

Like

Repost

Add to playlist

Share

Download

Buy on iTunes

2,296

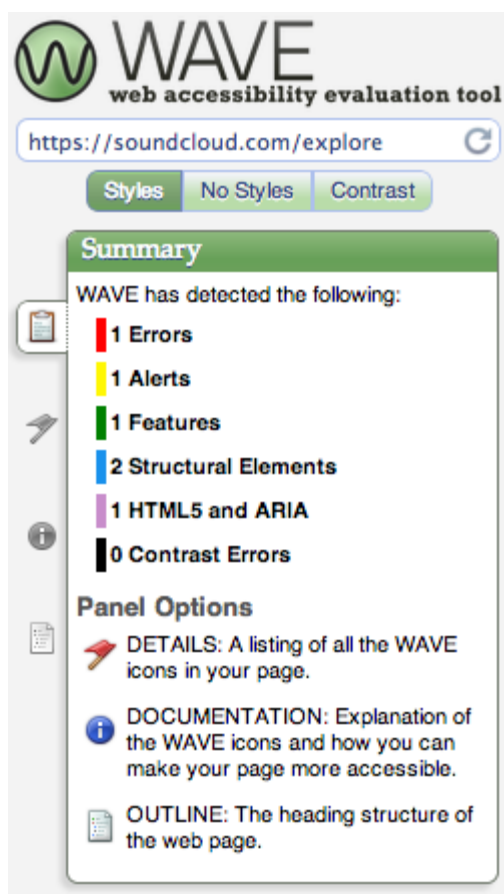
374

[Legal](#)
[Privacy](#)
[Cookies](#)
[Imprint](#)

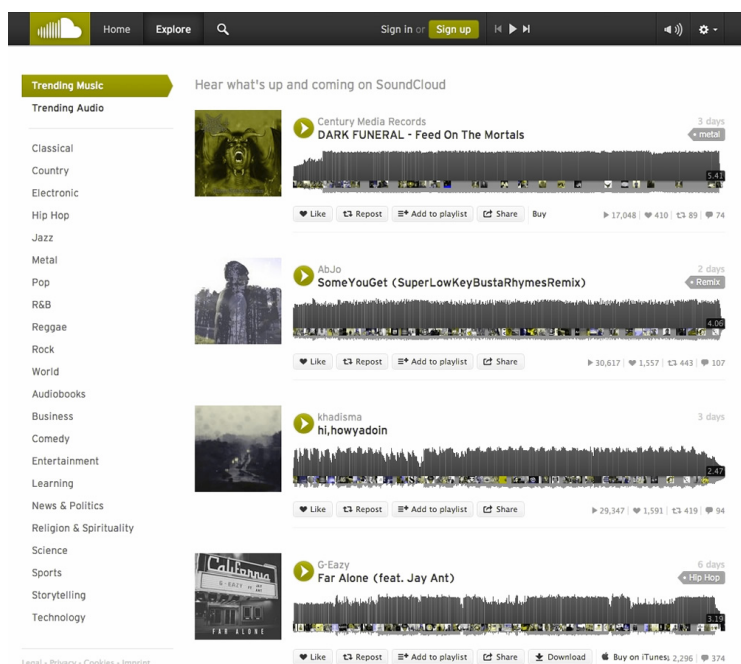
10-10. Screenshot of SoundCloud website

120

24 ways 2013 edition



10-11. Screenshot of SoundCloud's Wave results – one error



10-12. Screenshot of SoundCloud through a Color Oracle filter

EDUCATION AND BALANCE

As with most web design, doing accessibility well is about combining your knowledge of accessibility with your project's context to create a balance that serves your users' needs. Your types of content and interactions will dictate one set of constraints. Your users' needs and goals will dictate another. In broad terms, web design as a practice is finding the equilibrium between these constraints.

And then there's just caring. The web as a platform is open, affordable and available to many. Accessibility is our way to ensure that nobody gets shut out.

ABOUT THE AUTHOR



Laura Kalbag is a designer easily excited by web design and development. Among her list of ever-changing pet subjects are responsive web, semantic web, and web fonts, but she's really fascinated by anything in the areas of web, mobile and design.

Laura has been a **freelancer** for the whole of her professional life. She revels in working with small and meaningful clients, creating websites, apps, icons, illustrations and the odd logo.

11. Grunt for People Who Think Things Like Grunt are Weird and Hard

Chris Coyier

24ways.org/201311

Front-end developers are often told to do certain things:

- **Work in as small chunks of CSS and JavaScript** as makes sense to you, then concatenate them together for the production website.
- **Compress your CSS and minify your JavaScript** to make their file sizes as small as possible for your production website.
- **Optimize your images** to reduce their file size without affecting quality.
- **Use Sass** for CSS authoring because of all the useful abstraction it allows.

That's not a comprehensive list of course, but those are the kind of things we need to do. You might call them *tasks*.

I bet you've heard of **Grunt**. Well, Grunt is a *task runner*. Grunt can do all of those things for you. Once you've got it set up, which isn't particularly difficult, those things can happen automatically without you having to think about them again.

But let's face it: Grunt is one of those fancy newfangled things that all the cool kids seem to be using but at first glance feels strange and intimidating. I hear you. This article is for you.

LET'S NIP SOME MISCONCEPTIONS IN THE BUD RIGHT AWAY

Perhaps you've *heard* of Grunt, but haven't done anything with it. I'm sure that applies to many of you. Maybe one of the following hang-ups applies to you.

I don't need the things Grunt does

You probably do, actually. Check out that list up top. Those things aren't nice-to-haves. They are pretty vital parts of website development these days. If you already do all of them, that's awesome. Perhaps you use a variety of different tools to accomplish them. Grunt can help bring them under one roof, so to speak. If you don't already do all of them, you probably should and Grunt can

help. Then, once you are doing those, you can keep using Grunt to do more for you, which will basically make you better at doing your job.

Grunt runs on Node.js — I don't know Node

You don't have to know Node. Just like you don't have to know Ruby to use Sass. Or PHP to use WordPress. Or C++ to use Microsoft Word.

I have other ways to do the things Grunt could do for me

Are they all organized in one place, configured to run automatically when needed, and shared among every single person working on that project? Unlikely, I'd venture.

Grunt is a command line tool — I'm just a designer

I'm a designer too. I prefer native apps with graphical interfaces when I can get them. But I don't think that's going to happen with Grunt¹.

The extent to which you need to use the command line is:

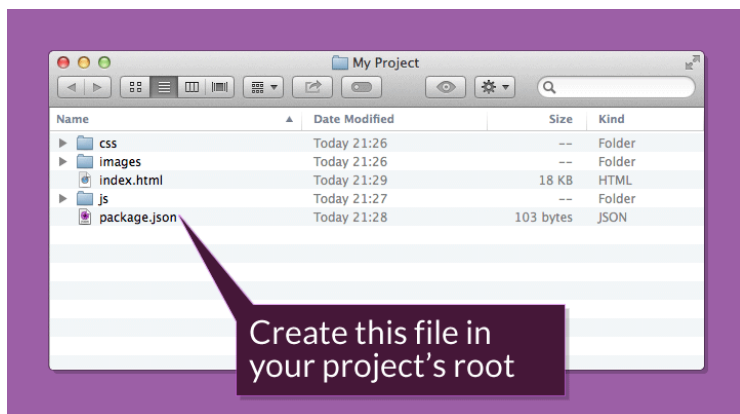
1. Navigate to your project's directory.
2. Type `grunt` and press **Return**.

After set-up, that is, which again isn't particularly difficult.

OK. LET'S GET GRUNT INSTALLED

Node is indeed a prerequisite for Grunt. If you don't have Node installed, don't worry, it's very easy. You literally download an installer and run it. Click the big **Install** button on the Node website.

You install Grunt on a per-project basis. Go to your project's folder. It needs a file there named *package.json* at the root level. You can just create one and put it there.



11-1. *package.json* at root

The contents of that file should be this:

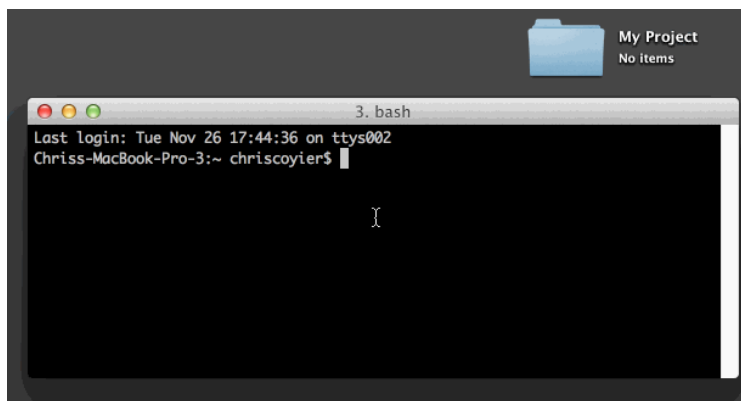
```
{  
  "name": "example-project",  
  "version": "0.1.0",  
  "devDependencies": {
```

```
    "grunt": "~0.4.1"
  }
}
```

Feel free to change the name of the project and the version, but the `devDependencies` thing needs to be in there just like that.

This is how Node does dependencies. Node has a package manager called **NPM** (Node packaged modules) for managing Node dependencies (like a `gem` for Ruby if you're familiar with that). You could even think of it a bit like a plug-in for WordPress.

Once that `package.json` file is in place, go to the terminal and navigate to your folder. Terminal rube like me do it like this:



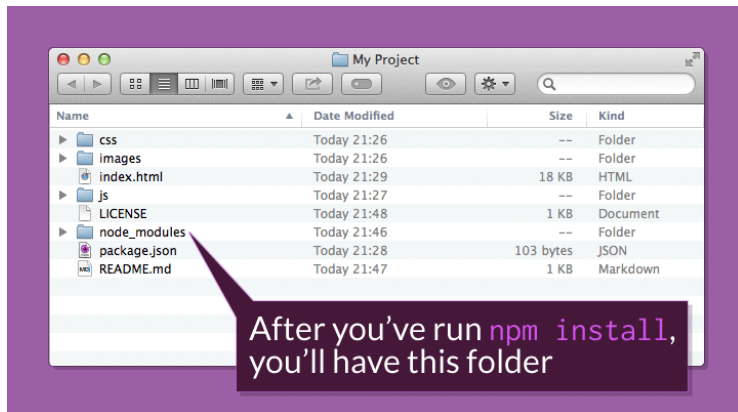
11-2. Terminal rube changing directories

Grunt for People Who Think Things Like Grunt are Weird and Hard

Then run the command:

```
npm install
```

After you've run that command, a new folder called *node_modules* will show up in your project.



11-3. Example of *node_modules* folder

The other files you see there, *README.md* and *LICENSE* are there because I'm going to put this project on GitHub and that's just standard fare there.

The last installation step is to install the Grunt CLI (command line interface). That's what makes the grunt command in the terminal work. Without it, typing grunt will net you a "Command Not Found"-style error. It is a separate installation for efficiency reasons. Otherwise, if you had ten projects you'd have ten copies of Grunt CLI.

This is a one-liner again. Just run this command in the terminal:

```
npm install -g grunt-cli
```

You should close and reopen the terminal as well. That's a generic good practice to make sure things are working right. Kinda like restarting your computer after you install a new application was in the olden days.

LET'S MAKE GRUNT CONCATENATE SOME FILES

Perhaps in our project there are three separate JavaScript files:

1. ***jquery.js*** – The library we are using.
2. ***carousel.js*** – A jQuery plug-in we are using.
3. ***global.js*** – Our authored JavaScript file where we configure and call the plug-in.

In production, we would concatenate all those files together for performance reasons (one request is better than three). We need to tell Grunt to do this for us.

But wait. Grunt actually doesn't do anything all by itself. Remember Grunt is a task *runner*. The tasks themselves we will need to add. We actually haven't set up Grunt to do anything yet, so let's do that.

The official Grunt plug-in for concatenating files is **grunt-contrib-concat**. You can read about it on GitHub if you want, but all you have to do to use it on your project is to run this command from the terminal (it will henceforth go without saying that you need to run the given commands from your project's root folder):

```
npm install grunt-contrib-concat --save-dev
```

A neat thing about doing it this way: your *package.json* file will automatically be updated to include this new dependency. Open it up and check it out. You'll see a new line:

```
"grunt-contrib-concat": "~0.3.0"
```

Now we're ready to use it. To use it we need to start configuring Grunt and telling it what to do.

You tell Grunt what to do via a configuration file named *Gruntfile.js*²

Just like our *package.json* file, our *Gruntfile.js* has a very special format that must be just right. I wouldn't worry about what every word of this means. Just check out the format:

```
module.exports = function(grunt) {  
  
    // 1. All configuration goes here  
    grunt.initConfig({  
        pkg: grunt.file.readJSON('package.json'),
```

```

        concat: {
            // 2. Configuration for concatenating files
            goes here.
        }

    });

    // 3. Where we tell Grunt we plan to use this
    plug-in.
    grunt.loadNpmTasks('grunt-contrib-concat');

    // 4. Where we tell Grunt what to do when we type
    "grunt" into the terminal.
    grunt.registerTask('default', ['concat']);

};

```

Now we need to create that configuration. The documentation can be overwhelming. Let's focus just on the very simple **usage example**.

Remember, we have three JavaScript files we're trying to concatenate. We'll list file paths to them under `src` in an array of file paths (as quoted strings) and then we'll list a destination file as `dest`. The destination file doesn't have to exist yet. It will be created when this task runs and squishes all the files together.

Both our *jquery.js* and *carousel.js* files are libraries. We most likely won't be touching them. So, for organization, we'll keep them in a */js/libs/* folder. Our *global.js* file is

where we write our own code, so that will be right in the `/js/` folder. Now let's tell Grunt to find all those files and squish them together into a single file named *production.js*, named that way to indicate it is for use on our real live website.

```
concat: {
  dist: {
    src: [
      'js/libs/*.js', // All JS in the libs folder
      'js/global.js'  // This specific file
    ],
    dest: 'js/build/production.js',
  }
}
```

Note: throughout this article there will be little chunks of configuration code like above. The intention is to focus in on the important bits, but it can be confusing at first to see how a particular chunk fits into the larger file. If you ever get confused and need more context, refer to **the complete file**.

With that concat configuration in place, head over to the terminal, run the command:

```
grunt
```

and watch it happen! *production.js* will be created and will be a perfect concatenation of our three files. This was a big aha! moment for me. Feel the power course through your veins. Let's do more things!

LET'S MAKE GRUNT MINIFY THAT JAVASCRIPT

We have so much prep work done now, adding new tasks for Grunt to run is relatively easy. We just need to:

1. Find a Grunt plug-in to do what we want
2. Learn the configuration style of that plug-in
3. Write that configuration to work with our project

The official plug-in for minifying code is **grunt-contrib-uglify**. Just like we did last time, we just run an NPM command to install it:

```
npm install grunt-contrib-uglify --save-dev
```

Then we alter our *Gruntfile.js* to load the plug-in:

```
grunt.loadNpmTasks('grunt-contrib-uglify');
```

Then we configure it:

```
uglify: {  
  build: {  
    src: 'js/build/production.js',  
    dest: 'js/build/production.min.js'  
  }  
}
```

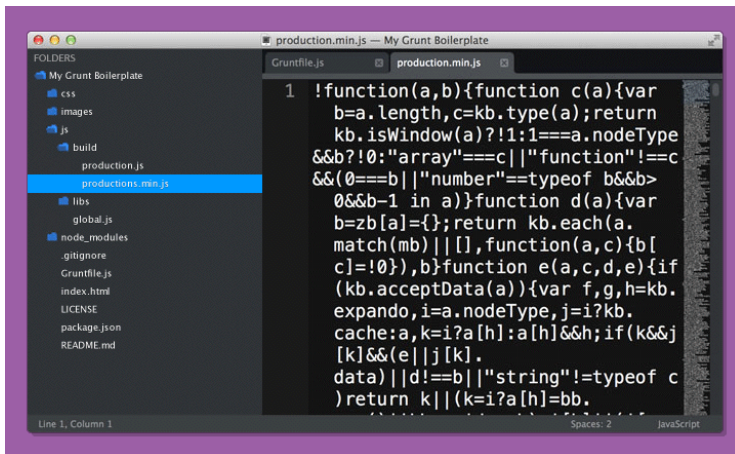
Let's update that default task to also run minification:

```
grunt.registerTask('default', ['concat', 'uglify']);
```

Super-similar to the concatenation set-up, right?

Grunt for People Who Think Things Like Grunt are Weird and Hard

Run grunt at the terminal and you'll get some deliciously minified JavaScript:



11-4. Minified JavaScript

That *production.min.js* file is what we would load up for use in our *index.html* file.

LET'S MAKE GRUNT OPTIMIZE OUR IMAGES

We've got this down pat now. Let's just go through the motions. The official image minification plug-in for Grunt is *grunt-contrib-imagemin*. Install it:

```
npm install grunt-contrib-imagemin --save-dev
```

Register it in the *Gruntfile.js*:

```
grunt.loadNpmTasks('grunt-contrib-imagemin');
```

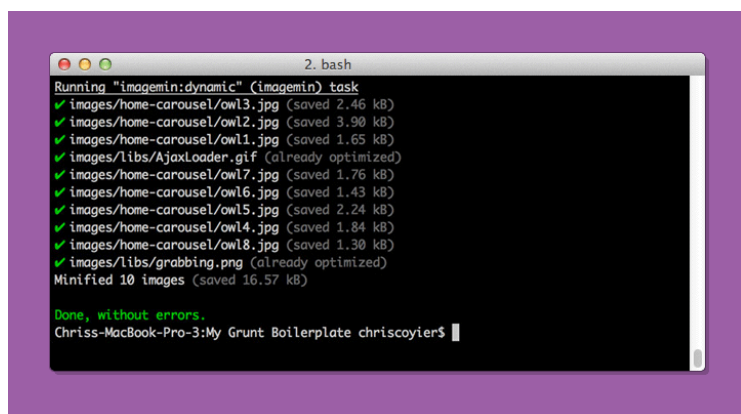
Configure it:

```
imagemin: {
  dynamic: {
    files: [{
      expand: true,
      cwd: 'images/',
      src: ['**/*.{png,jpg,gif}'],
      dest: 'images/build/'
    }]
  }
}
```

Make sure it runs:

```
grunt.registerTask('default', ['concat', 'uglify',
  'imagemin']);
```

Run grunt and watch that gorgeous squishification happen:



11-5. Squished images

Gotta love performance increases for nearly zero effort.

LET'S GET A LITTLE BIT SMARTER AND AUTOMATE

What we've done so far is awesome and incredibly useful. But there are a couple of things we can get smarter on and make things easier on ourselves, as well as Grunt:

1. Run these tasks automatically when they should
2. Run only the tasks needed at the time

For instance:

1. Concatenate and minify JavaScript when JavaScript changes
2. Optimize images when a new image is added or an existing one changes

We can do this by watching files. We can tell Grunt to keep an eye out for changes to specific places and, when changes happen in those places, run specific tasks. Watching happens through the official **grunt-contrib-watch** plugin.

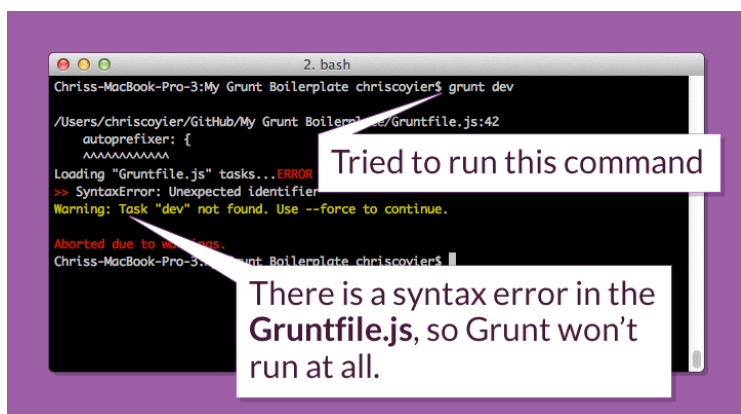
I'll let you install it. It is exactly the same process as the last few plug-ins we installed. We configure it by giving watch specific files (or folders, or both) to watch. By watch, I mean monitor for file changes, file deletions or file additions. Then we tell it what tasks we want to run when it detects a change.

We want to run our concatenation and minification when anything in the `/js/` folder changes. When it does, we should run the JavaScript-related tasks. And when things happen elsewhere, we should *not* run the JavaScript-related tasks, because that would be irrelevant. So:

```
watch: {
  scripts: {
    files: ['js/*.js'],
    tasks: ['concat', 'uglify'],
    options: {
      spawn: false,
    },
  },
}
```

Feels pretty comfortable at this point, hey? The only weird bit there is the `spawn` thing. And you know what? I don't even really know what that does. From what I understand from the documentation it is the smart default. That's real-world development. Just leave it alone if it's working and if it's not, learn more.

Note: Isn't it frustrating when something that looks so easy in a tutorial doesn't seem to work for you? If you can't get Grunt to run after making a change, it's very likely to be a syntax error in your *Gruntfile.js*. That might look like this in the terminal:



11-6. Errors running Grunt

Usually Grunt is pretty good about letting you know what happened, so be sure to read the error message. In this case, a syntax error in the form of a missing comma foiled me. Adding the comma allowed it to run.

LET'S MAKE GRUNT DO OUR PREPROCESSING

The last thing on our list from the top of the article is using Sass — yet another task Grunt is well-suited to run for us. But wait? Isn't Sass technically in Ruby? Indeed it is. There is a version of Sass that will run in Node and thus not add an additional dependency to our project, but it's not quite up-to-snuff with the main Ruby project. So, we'll use the official `grunt-contrib-sass` plug-in which just assumes you have Sass installed on your machine. If you don't, follow the command line instructions.

What's neat about Sass is that it can do concatenation and minification all by itself. So for our little project we can just have it compile our main *global.scss* file:

```
sass: {
  dist: {
    options: {
      style: 'compressed'
    },
    files: {
      'css/build/global.css': 'css/global.scss'
    }
  }
}
```

We wouldn't want to manually run this task. We already have the watch plug-in installed, so let's use it! Within the watch configuration, we'll add another subtask:

```
css: {
  files: ['css/*.scss'],
  tasks: ['sass'],
  options: {
    spawn: false,
  }
}
```

That'll do it. Now, every time we change any of our Sass files, the CSS will automatically be updated.

Let's take this one step further (it's absolutely worth it) and add LiveReload. With LiveReload, you won't have to go back to your browser and refresh the page. Page

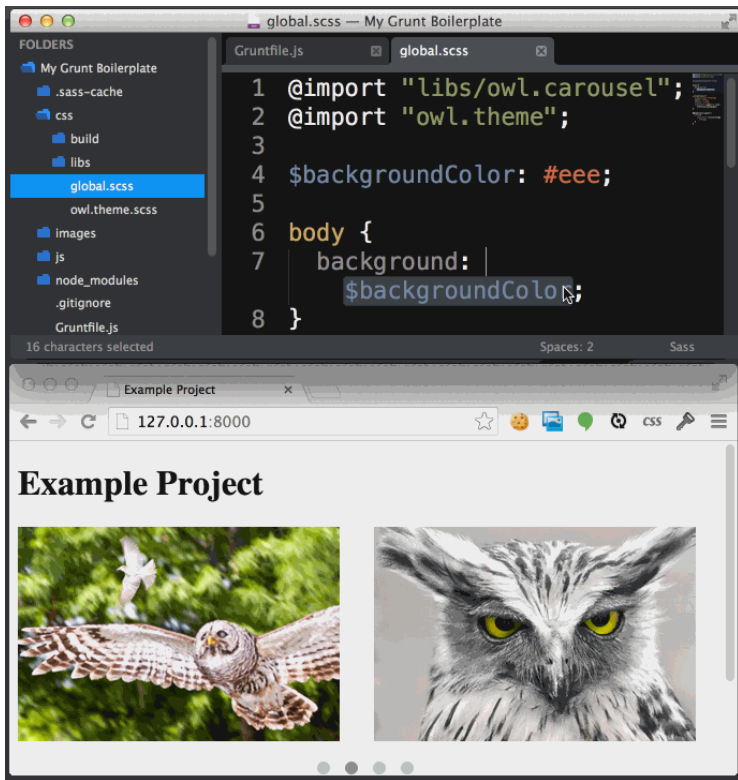
refreshes happen automatically and in the case of CSS, new styles are injected without a page refresh (handy for heavily state-based websites).

It's very easy to set up, since the LiveReload ability is built into the watch plug-in. We just need to:

1. Install the **browser plug-in**
2. Add to the top of the watch configuration:

```
. watch: {  
  options: {  
    livereload: true,  
  },  
  scripts: {  
    /* etc */  
  }  
}
```

3. Restart the browser and click the LiveReload icon to activate it.
4. Update some Sass and watch it change the page automatically.



11-7. Live reloading browser

Yum.

Prefer a video?

If you're the type that likes to learn by watching, I've made a screencast to accompany this article that I've published over on [CSS-Tricks: First Moments with Grunt](#)

LEVELING UP

As you might imagine, there is a *lot* of leveling up you can do with your build process. It surely could be a **full time job** in some organizations.

Some hardcore devops nerds might scoff at the simplistic setup we have going here. But I'd advise them to slow their roll. Even what we have done so far is tremendously valuable. And don't forget this is all free and open source, which is amazing.

You might level up by adding more useful tasks:

- Running your CSS through **Autoprefixer** (A+ Would recommend) instead of a preprocessor add-ons.
- Writing and running JavaScript unit tests (example: **Jasmine**).
- Build your image sprites and SVG icons automatically (example: **Grunticon**).
- Start a server, so you can link to assets with proper file paths and use services that require a real URL like **TypeKit** and such, as well as remove the need for other tools that do this, like **MAMP**.
- Check for code problems with **HTML-Inspector**, **CSS Lint**, or **JS Hint**.
- Have new CSS be **automatically injected** into the browser when it ever changes.
- Help you commit or push to a version control repository like **GitHub**.

- Add version numbers to your assets (cache busting).
- Help you deploy to a staging or production environment (example: DPLOY).

You might level up by simply understanding more about Grunt itself:

- Read **Grunt Boilerplate** by Mark McDonnell.
- Read **Grunt Tips and Tricks** by Nicolas Bevacqua.
- Organize your *Gruntfile.js* by splitting it up into smaller files.
- Check out other people's and projects' *Gruntfile.js*.
- Learn more about Grunt by digging into its source and learning about its API.

LET'S SHARE

I think some group sharing would be a nice way to wrap this up. If you are installing Grunt for the first time (or remember doing that), be especially mindful of little frustrating things you experience(d) but work(ed) through. Those are the things we should share in the comments here. That way we have this safe place and useful resource for working through those confusing moments without the embarrassment. We're all in this thing together!



¹ Maybe someday someone will make a beautiful Grunt app for your operating system of choice. But I'm not sure that day will come. The configuration of the plug-ins is the important part of using Grunt. Each plug-in is a bit different, depending on what it does. That means a uniquely considered UI for every single plug-in, which is a long shot.

Perhaps a decent middleground is this **Grunt DevTools** Chrome add-on.

² *Gruntfile.js* is often referred to as *Gruntfile* in documentation and examples. Don't literally name it *Gruntfile* — it won't work.

ABOUT THE AUTHOR



Chris Coyier is a web designer and developer living in Milwaukee. He writes about all thing web at [CSS-Tricks](#), talks about all things web at conferences around the world and on his podcast [ShopTalk](#) with Dave Rupert, and co-founded the web coding playground [CodePen](#).

12. The Responsive Hover Paradigm

Jenn Lukas

24ways.org/201312

CSS transitions and animations provide web designers with a whole slew of tools to spruce up our designs. Move over ActionScript tweens! The techniques we can now implement with CSS are reminiscent of Flash-based adventures from the pages of web history.

Pairing CSS enhancements with our `:hover` pseudo-class allows us to add interesting events to our websites. We have a ton of power at our fingertips. However, with this power, we each have to ask ourselves: just because I can do something, *should* I?

WHY BOTHER?

We hear a lot of mantras in the web community. Some proclaim the importance of content; some encourage methods like mobile first to support content; and others

warn of the overhead and speed impact of decorative flourishes and visual images. I agree, one hundred percent. At the same time, I believe that content can reign king and still provide a beautiful design with compelling interactions and acceptable performance impacts. Maybe, just maybe, we can even have a little bit of fun when crafting these systems!

Yes, a site with pure HTML content and no CSS will load very fast on your mobile phone, but it leaves a lot to be desired. If you went to your local library and every book looked the same, how would you know which one to borrow? Imagine if every book was printed on the same paper stock with the same cover page in the same type size set at a legible point value... how would you know if you were going to purchase a **cookbook about wild game** or a **young adult story about teens fighting to the death**?

For certain audiences, seeing a site with hip, lively hovers sure beats a stale website concept. I've worked on many higher education sites, and setting the interactive options is often a very important factor in engaging potential students, alumni, and donors. The same can go for e-commerce sites: enticing your audience with surprise and delight factors can be the difference between a successful and a lost sale.

Knowing your content and audience can help you decide if an intriguing experience is appropriate for your site; if it is, then hover responses can be a real asset.

WHY HOVER?

We have all these capabilities with CSS properties to create the aforementioned fun interactions, and it would be quite easy to fall back into some old patterns and animation abuse. The world of Flash intros and skip links could be recreated with CSS keyframes. However, I don't think any of us want to go the route of forcing users into unwanted exchanges and road blocking content.

What's great about utilizing hover to pair with CSS powered actions is that it's user initiated. It's a well-established expectation that when a user mouses over an object, something changes. If we can identify that something as a link, then we will expect something to change as we move our mouse over it. By waiting to trigger a CSS-based response until a user chooses to engage with a target makes for a more polished experience (as opposed to barraging our screens with animations all willy-nilly). This makes it the perfect opportunity to add some unique spunk.

WHAT ABOUT MOBILE, TOUCH, AND RESPONSIVE?

So, you're on board with this so far, but what about mobile and touch devices? Sure, some devices like the Samsung Galaxy S4 have some hovering capabilities, but certainly most do not. Beyond mobile devices, we also have to worry about desktops with touch capabilities. It's super difficult to detect if a user is currently using touch or hover. One option we have is to design strictly for touch only and send hover enhancements to the graveyard. However, being that I'm all "fuck yeah hovers!," I like to explore all options. So, let's examine four different types of hover patterns and see how they can translate to our touch devices.

1. THE ESSENTIAL TEXT HOVER

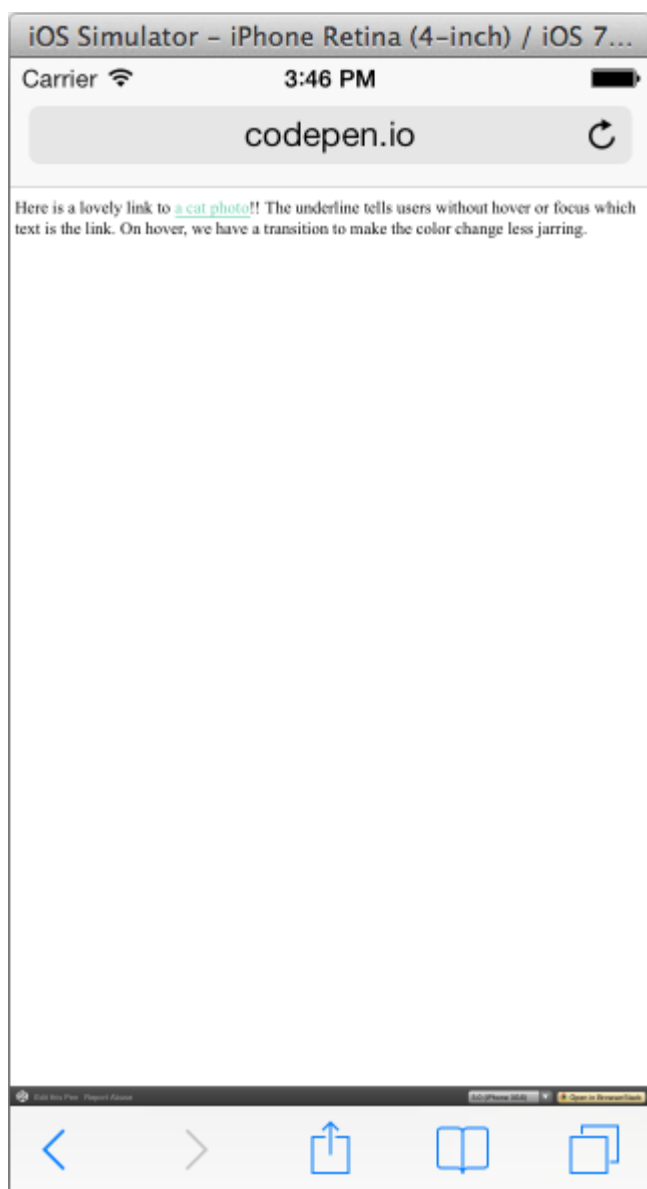
Changing text color on hover is something we've done for a while and it has helped aid in identifying links. To maintain the **best accessibility** we can achieve, it helps to have a different visual indicator on the default :link state, such as an underline. By making sure all text links have an underline, we won't have to rely on visual changes during hover to make sure touch device users know that it is a link. For hover-enabled devices, we can add a basic color transition. Doing so creates a nice fade, which makes the change on hover less jarring. Kinda like smooth jazz. The code* to achieve this is quite simple:

```
a {  
  color: #6dd4b1;  
  transition: color 0.25s linear;  
}  
  
a:hover, a:focus {  
  color: #357099;  
}
```

- Browser prefixes are omitted

You can see in the final result that, for both touch and hover, everyone wins:

See the [Pen Most Basic Link Transition](#) by Jenn Lukas (@Jenn) on CodePen



2. VISUAL BACKGROUND WIZARDRY AND ANIMATED HOVERS

We can take this a step further by again making changes to our aesthetic on hover, but not making any content changes. Altering image hovers for fun and personality can separate your site from others; that personality is important and can enhance our content.

Let's look at a few sites that do this really well. Scroll down to the judges section of **CSS Off** and check out the illustrations of the judges. On hover, the illustration fades into a photo of the judge. This provides a realistic alternative to the drawing. Users without the hover can click into the detail page, where they can see the full color picture and learn more about the judges; the information is still available through a different pathway.

Going back to the higher education field, let's visit **Delaware Valley College**. The school had recently gone through a rebranding that included loop icons as a symbol to connect ideas. These icons are brought into the website on hover of the slideshow arrows (WebKit browsers). The hover reveals a loop animation, tying in overall themes and adding some extra pizzazz that makes me think, "This is a hip place that feels current." For visitors who can't access the hover effect, the default arrow state clearly represents a clickable link, and there is swipe functionality on mobile devices to boot.

DIY.org's **Frontend Dev** page has a bunch of enjoyable hover actions happening, featuring scaling transforms and looping animations. Nothing new is revealed on hover, so touch devices won't miss anything, but it intrigues the user who is visiting a site about front-end dev doing cool front-end things. It backs up its claim of front-end knowledge by adding this enhancement.

The old Cowork Chicago (now redirecting) had a great example, captured here:

See: [//player.vimeo.com/video/60981511](https://player.vimeo.com/video/60981511)

Coop: Chicago Coworking from Jenn Lukas on Vimeo.

The code for the Join areas is quite simple:

```
.join-buttons .daily, .join-buttons .monthly {
    height: 260px; z-index: 0; margin-top: 30px;
    transition: height .2s linear, margin .2s linear;
}

.join-buttons .daily:hover, .join-buttons .monthly:hover
{
    height: 280px; margin-top: 20px;
}

li.button:hover {
    z-index: 20;
}
```

The slight rotation on the photos, and the change of color and size of the rate options on hover, add to the fun factor. The site attempts to advertise the co-working space by letting bits of their charisma show through with these transitions. They don't hit the user over the head with animations, but provide a nice addition to make sure visitors know it's a welcoming place to work. Some text is added on the hover, but the text isn't essential to determine where the link goes.

3. IMAGE BLOCK HOVERS

There have been more designs popping up with large image blocks acting as extensive hit area links to subsequent pages. On hover of these links, text is revealed, letting the user know where the link destination goes.

See the [Pen Transitioning Max Height](#) by Jenn Lukas (@Jenn) on [CodePen](#)

This type of link is tough for users on touch as the image might not provide enough context to reveal its target. If you weren't aware of what my illustrated avatar from 2007 looked like (or even if you did), then how would you know that this is a link to my Twitter page? Instead, if we provide enough context — such as the @jennlukas handle — you could assume the destination. Users who receive the hover can also see the Twitter bio. It won't break the

experience for users that can't hover, but it will provide a nice interaction and some more information for those that can.

See the [Pen Transitioning Max Height](#) by Jenn Lukas (@Jenn) on CodePen

The [Esquire](#) site follows this same pattern, in which the title of the story is shown and the subheading is revealed on hover. [Dining at Altitude](#) took the opposite approach, where all text is shown by default and, on hover, you can see more of the image that the text sits atop. This is a nice technique to follow. For touch users, following the link will allow them to see more of the image detail that was revealed on hover.

4. DROP-DOWN NAVIGATION MENU HOVERS

Main navigation options that rely on hover have come up as a problem for touch. One way to address this is to be sure your top level items are all functional links to somewhere, and not blank anchors to trigger a submenu drop-down. This ensures that, even without the hover-triggered menu, users can still navigate to those top-level pages. From there, they should be able to access the tertiary pages shown in the drop-down. Following this arrangement, drop-down menus act as a quick shortcut

and aren't necessary to the navigational structure. If the top navigation items are your most visited pages, this execution won't hinder your visitors.

If the information within the menu is vital, such as a lone account menu, another option is to show drop-down menus on click instead of hover. This pattern will allow both mouse and touch users to access the drop-downs.

WHY CAN'T WE JUST DETECT HOVER?

This is a really tricky thing to do. Internet Explorer 10 on Windows 8 uses the `aria-haspopup` attribute to simulate hover on touch devices, but usually our audience stretches beyond that group. There's been discussion around using Modernizr, but **false positives** have come with that. A **W3C draft** for Media Queries Level 4 includes a hover feature, but it's **not supported yet**. Since some devices can hover and touch, should you rely on hover effects for those? Arguments have come up that users can be browsing your site with a mouse and then decide to switch to touch, or vice versa. That might be a large concern for you, or it might be an edge case that isn't vital to your site's success.

For one site, I used `mousemove` and `touchstart` JavaScript events in order to detect if a visitor starts to browse the site with a mouse. The design initiates for touch users,

showing all text on load, but as soon as a mouse movement occurs, the text becomes hidden and is then revealed on hover.

See the Pen **Detect Touch devices with mousemove and touchstart** by Jenn Lukas (@Jenn) on CodePen

One downside to this approach is that the text is viewable until a mouse enters the document, but if the elements are further down the page it might not be noticed. A second downside is if a user on a touch- and hover-enabled device starts browsing with the mouse and then switches back to touch, the hover-centric styles will remain until a new page load. These were acceptable scenarios in the project I worked on, but might not be for every project.

CAN WE GIVE OUR VISITORS A CHOICE?

I've been thinking about how we can combat the concern of not knowing if our customers are using touch or a mouse, not to mention keyboard or Wacom tablets or **Minority Report** screens. We can cover keyboards with **our friend** : focus, but that still doesn't solve our other dilemmas.

Remember when we couldn't rely on browsers to zoom text and we had to use those small A, medium A, big A [AAA] buttons? On selection of one of those options, a

different style sheet would load with small, medium, or large text sizes to satisfy our user's request. We could even set cookies to remember their font choices. What if we offered a similar solution, a hover/touch switcher, for our new predicament?

See the Pen [cwuJf](#) by Jenn Lukas (@Jenn) on CodePen

We could add this switcher to our design. Maybe add it to the header on smaller screens and the footer on larger screens to play the odds. Then be sure to deliver the appropriate touch- or hover-optimized adventure for our guests.

How about adding **View** options in the areas where we're hiding content until hover? Looking at **Delta Cycle**, there's logic in place to switch layouts on some mobile devices. On desktops we can see the layout shows the product and price by default, and the name of the item and an **Add to cart** button on hover. If you want to keep this hover, but also worry that touch users can't access it — or even if you are concerned that people might want to view it with more details up front — we could add another view switcher.

See the Pen [List/Grid Views for Hover or Touch](#) by Jenn Lukas (@Jenn) on CodePen

Similar to the list versus grid view we often see in operating systems, a choice here could cover all of our bases.

CONCLUSION

There is no one-size-fits-all solution when it comes to hover patterns. Design for your content. If you are providing important information about driving directions or healthcare, you might want to err on the side of designing for touch only. If you are behind an educational site and trying to entice more traffic and sign-ups, or a more immersive e-commerce site selling **pies**, then hover activity can help support your content and engage your visitors without being a detriment. While content can be our top priority, let's not forget that our designs and interactions, hovers included, can have a great positive impact on how visitors experience our site. Hover wisely, friends.

ABOUT THE AUTHOR



Jenn Lukas is a multi-talented front-end consultant and freelance developer in Philadelphia and is the founder of **Ladies in Tech**. She speaks at a variety of conferences, writes for **The Nerdary**, and has contributed to **The Pastry Box Project**.

Jenn's past experiences range from creating Navy training simulations to leading the front-end team at Happy Cog as Interactive Development Director. She was named one of Mashable's 15 Developer/Hacker Women to Follow on Twitter,

and you can find her on **Twitter** posting development and cat-related news. When she's not crafting sites with the finest of web standards, Jenn teaches HTML and CSS for **GirlDevelopIt**.

13. Data-driven Design with an Annual Survey

Aarron Walter

24ways.org/201313

Too often, we base designs on assumptions that don't match customer perspectives. Why? Because the data we need to make informed decisions isn't available.

Imagine starting off the year with a treasure trove of user data that can be filtered, sliced, and diced to inform new UI designs, help you discover where users struggle the most, and expose emerging trends in your customers' needs that could lead to new features. Why, that would be useful indeed. And it's easy to obtain by conducting an annual survey.

Annual surveys may seem as exciting as receiving socks and undies for Christmas, but they're the gift that keeps on giving all year long (just like fresh socks and undies). I'm not ashamed to admit it: I love surveys! Each time my design research team runs a survey, we learn so much about customer motivations, interests, and behaviors.

Surveys provide an aggregate snapshot of your users that can't easily be obtained by other research methods, and they can be conducted quickly too. You can build a survey in a few hours, run a pilot test in a day, and have real results streaming in the following day. Speed is essential if design research is going to keep pace with a busy product release schedule.

Surveys are also an invaluable springboard for customer interviews, which provide deep perspectives on user behavior. If you play your cards right as you construct your survey, you can capture a user ID and an email address for each respondent, making it easy to get in touch with customers whose feedback is particularly intriguing. No more recruiting customers for your research via Twitter or through a recruiting company charging a small fortune. You can filter survey responses and isolate the *exact* customers to talk with in moments, not months.

I love this connected process of sending targeted surveys, filtering the results, and then — with surgical precision — selecting just the right customers to interview. Not only is it fast and cheap, but it lets design researchers do quantitative and qualitative research in a coordinated way. Aggregate survey responses help you quantify the perspectives of different user segments, and interviews help you get into the heads of your customers.

An annual survey can give your team the data needed to make more informed designs in the new year. It all starts with a plan.

PLANNING YOUR SURVEY

Before you start jotting down questions to ask users, spend some time thinking about the work your team will be doing in the coming year. Are you planning new mobile apps or a responsive redesign? Then questions about devices used and behaviors around mobile devices might be in order. Rethinking your content strategy? Then you might want to ask a few questions about how your customers consume content.

You can't predict all of the projects you'll be working on in the coming year, but tuck a couple of sections in your survey about the projects you're certain about. This will give you the research you need to start new projects with solid foundational data.

Google Drive is a great place to start collaboratively building survey questions with colleagues. Questions that seem crystal clear in your head get challenged, refined, or even expanded quickly when the entire team can chime in.

As you craft your survey, try to consider how you'll filter it once all of the data is compiled. Do you need to see responses by industry, by age of an account, by devices used, or by size of company? Adding the right filter

questions can help you discover fascinating patterns in user segments. Filtering on responses to a few questions can surface insights like: customers in non-profit companies with more than 100 employees are 17% more likely to use an Android phone and are most attracted to features A, D, and F. A designer working on the landing page for a non-profit would **love** to have concrete information like this. Filter questions are key, so consider them carefully. But don't go overboard — too many of them and you'll start to hurt your survey response rate.

Multiple choice questions are the heart of most surveys because respondents can complete them quickly, which increases response rate, and researchers can analyze them without a lot of manual categorization. Open text field questions are valuable too, but be careful not to add too many to your survey. You'll hate yourself after the survey's done and you have to sort through and tag thousands of open responses so patterns become visible. Oy vey!

An open-ended question works well towards the end of the survey. At this point respondents have a lot of topics swirling around in their head and tend to say weird things that will pique your interest. This is where you'll find the outliers who are using your product. They'll be fascinating to interview, and on occasion will help you see your work in a brand new way.

Conclude your survey with a question asking permission to get in touch for a followup interview so you don't pester people who want to be left alone.

With your questions nailed down, it's time to build out that survey and get it ready for sending!

BUILDING YOUR SURVEY

There are dozens of apps you could use to build your survey, but **SurveyMonkey** is the one that I prefer. It lets you **pass in variables** for each respondent such as user ID and email address. Metadata about respondents is essential if you're going to do any follow-up interviews with your customers in the coming year. SurveyMonkey also makes it easy to set up **question logic**, showing questions to customers only if they responded in a certain way to a prior question. This helps you avoid asking irrelevant questions to some respondents.

DETERMINING SURVEY RECIPIENTS

Once you've chosen a survey tool and entered all of your questions, you need to gather a list of recipients. Your first instinct will be to send it to everyone. You might say, "I need maximum response and metric shit tons of data!" But this is rarely the best approach — broad distribution almost always leads to lower response rates, increased noise, and decreased signal in your data. Are there

subsets of customers you could send to, like only those who are active, those who are paying, or have been with you for a certain length of time? Talk to the keepers of your customer database and see how they can segment it so you can be certain you're talking to just the people who will have the most relevant responses for your needs.

If you want to get super nerdy when finding the right customer sample to survey, use a [sample size calculator]. Sampling is a deep subject best explored in **other articles**.

CRAFTING YOUR SURVEY EMAIL

After focusing your energies on writing and building your survey, the email asking your customers to respond seems almost trivial, but it will greatly influence your response rate. Take great care when writing your subject line and the body of the email. If you can pull it off, **A/B testing** subject lines can greatly improve the open rate of your email and click-through to your survey. My design research team has seen a ~10% increase in open and click rates when we A/B tested. We've found that personalizing subject lines and greetings with the recipients name (ie. "Hey, Aarron. How can we make our app work better for you?") gave us the best response rates. Your mileage may vary.

The tone of your email is important — be friendly, honest, and to the point. Those that are passionate about your product will be happy to share their perspective. Writing a survey email that people will actually respond to ain't easy — in fact, they're almost always annoying. But **Ben Chestnut** found a non-annoying way to send a survey email and improve response rates.

Hey, Freddie.

2012 was big for us here at MailChimp (our marketing team put together [a spiffy annual report that reveals the details](#)).

We released a lot of new stuff, and improved old mainstays:

- Launched [Gather, a text messaging app for events](#)
- Released a [transactional email app called Mandrill](#)
- Totally revamped [TinyLetter](#)
- Enhanced [list searching](#) and added [campaign search](#)
- Improved [Autoresponders and RSS-to-email](#)
- Rebuilt our [email editor with drag and drop design](#)

Holy cow, we've been busy!

But we've got more new stuff in the works for MailChimp in 2013. We want to make sure we're addressing the things that're most important to you. Could you take a few minutes to answer some questions that'll help shape MailChimp's direction in 2013?

Take the Survey

(takes about 8 minutes and we'll love you for it)

Thanks in advance for your time. We think you're going to like what we're up to in 2013.

Aaron Walter

User Experience Head Honcho at MailChimp

13-1. The email sent for the 2013 MailChimp survey let customers know what we'd been up to in the previous year, and invited feedback on what we should work on in the coming year.

The link to your survey should be a clear call to action. A big button with a label like "Answer a few questions" generally does the trick. The URL linking to the survey will

need to include some variables like user ID and email. It might look something like this if you're using SurveyMonkey:

```
http://surveymonkey.com/s/  
somesurveyid/?uid=*<UID>*&email=*<email>*
```

As each email is sent, the proper data will be populated in the variables, passing it on to the survey app for inclusion in each response. This is the magic that will help you pinpoint customers to interview down the road, so take special care to test that all is working before sending to all recipients. How you construct the survey link will vary depending on what survey tool and email service provider you use, so don't take my example as gospel. You'll need to read the documentation for your survey and email apps to set things up properly.

PILOT BEFORE SENDING

By now, you've whipped yourself into a fever pitch over your brilliant survey and the data you hope to collect. Your finger is on the send button, poised for action, but there's one **very** important thing to do before you send to the entire list of customers: send a pilot email. How do you know if your questions are clear, your form logic is sound, and you're passing variables from the email to the survey properly? You won't, unless you send to a small segment of your recipients first.

The data collected in your pilot will make plain where your survey needs refinement. This data won't be used in your final analysis, as you're probably going to make a few changes to your questions.

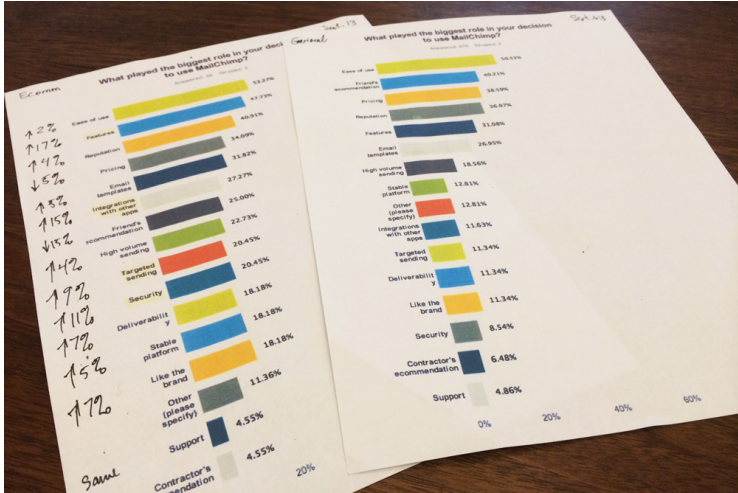
Send the pilot survey to enough people that you can really stress test the clarity of the questions and data you're gathering, while considering how much data can you comfortably throw out. If you're sending your final survey to a few thousand people, you might find a couple of hundred recipients for your pilot will give you enough insight into what to improve while leaving the vast majority of the recipients for your final survey.

After you've sent your pilot, made your survey adjustments, and ensured the variables are being passed from your email into the survey app, you're ready to send to the remainder of your customers. This is your moment of glory!

ANALYZING YOUR RESULTS

After a couple of weeks you can probably safely close the survey so no other responses come in as you transition from data gathering to data analysis. Any survey app worth its salt will chart responses to your multiple choice questions. Reviewing these charts is a great place to start your analysis. Is there anything particularly interesting that stands out? Jot down some of your observations. I

like to print screenshots of the charts for each question, highlighting areas of interest. These prints become a particularly handy reference point for the next step in your analysis.



13-2. Printing results from a survey makes comparing different customers easy.

Viewing aggregate data about all responses is interesting, but the deltas between different types of customers are where the real revelations happen. Remember those filter questions you added to your survey? They're the tool that'll help you compare customer segments.

Most survey apps will let you filter the data based on response to a question. If the one you're using doesn't, you can always export your data and create pivot tables in

Excel. Try filtering your data based on one of your filter questions, such as industry, company size, or devices used. Now compare those printed screenshots of baseline responses to the filtered data. Chances are you'll see some significant differences in how each group responded to your questions, giving you clues about the variance in interests and motivations in customer segments and a leg up as you work on future design projects.

Open-ended responses are equally interesting, but much more time-consuming to analyze. Yes, you need to read through thousands of responses, some of which are constructive and some of which are not. Taking the time to tag each open response will help you see trends and filter out the responses that are unhelpful.

Unlike questions with predefined answers, open-ended responses let users express unique ideas and use cases you may not be looking for. The tedium of reading thousands of response is always cut by eureka moments when users tell you something fascinating that changes your perspective on your app. These are the folks you want to pull out for follow-up **interviews**. Because you've already captured their email addresses when you set up your survey and your email, getting in touch will be a piece of cake.

Filter, compare, interview, and summarize; then share your findings with your colleagues. Reports are great for head honchos, but if you want to really inform and inspire, create a video, a poster series, or even a comic to communicate what you've learned. Want to get really fancy? Store your survey results in a **centrally accessible location** so anyone in your company can research and discover the insights they need to make more informed designs.

Good design researchers discover valuable insights. *Great* design researchers turn those insights into stories.

CONCLUSION

As we enter the new year, it's a great time to reflect on the work we've done in the past and how we can do better in the future. Without a doubt, designers working with a foundation of insights about customers can make more effective UIs. But designers aren't the only ones who stand to gain from the data collected in an annual survey—anyone who makes things for or communicates with customers will find themselves empowered to do better work when they know more about the people they serve. The data you collect with your survey is a fantastic holiday gift to your colleagues, one that they'll appreciate throughout the year.

ABOUT THE AUTHOR



Aarron Walter is the Director of User Experience at MailChimp, where he strives to make software more human. Aarron is the author of *Designing for Emotion from A Book Apart*. Aarron taught design at colleges in the US and Europe for nearly a decade, and speaks at conferences around the world. His design guidance has helped the White House, the US Department of State, and dozens of startups and venture capitalists. He tweets about design under the moniker [@aaron](#) on Twitter.

14. Home Kanban for Domestic Bliss

Meri Williams

24ways.org/201314

My wife is an architect. I'm a leader of big technical teams these days, but for many years after I was a dev I was a project/program manager. Our friends and family used to watch *Grand Designs* and think that we would make the ideal team — she could design, I could manage the project of building or converting whatever dream home we wanted.

Then we bought a house.

A Victorian terrace in the north-east of England that needed, well, a fair bit of work. The big decisions were actually pretty easy: yes, we should knock through a double doorway from the dining room to the lounge; yes, we should strip out everything from the utility room and

redo it; yes, we should roll back the hideous carpet in the bedrooms upstairs and see if we could restore the original wood flooring.

Those could be managed like a project.

What couldn't be was all the other stuff. Incremental improvements are harder to schedule, and in a house that's over a hundred years old you never know what you're going to find when you clear away some tiles, or pull up the carpets, or even just spring-clean the kitchen ("Erm, hon? The paint seems to be coming off. Actually, so does the plaster..."). A bit like going in to fix bugs in code or upgrade a machine — sometimes you end up **quite far down the rabbit hole**.

And so, as we tried to fit in those improvements in our evenings and weekends, we found ourselves disagreeing. Arguing, even. We were both trying to do the right thing (make the house better) but since we were fitting it in where we could, we often didn't get to talk and agree in detail what was needed (exactly *how* to make the house better). And it's really frustrating when you stay up late doing something, just to find that your other half didn't mean *that* they meant *this* instead, and so your effort was wasted.

Then I saw **this tweet** from my friend and colleague Jamie Arnold, who was using the same kanban board approach at home as we had instituted at the UK Government Digital Service to manage our portfolio.

Mrs Arnold embraces Kanban wall at home.
Disagreements about work in progress and
priority significantly reduced.. ;)
pic.twitter.com/407brMCH

— Jamie Arnold (@itsallgonewrong) October 27, 2012

And despite Jamie's questionable taste in fancy dress outfits (look closely at that board), he is a proper genius when it comes to processes and particularly agile ones. So I followed his example and instituted a home kanban board.

WHAT IS THIS KANBAN OF WHICH YOU SPEAK?

Kanban boards are an artefact from lean manufacturing — basically a visualisation of a production process. They are used to show you where your bottlenecks are, or where one part of the process is producing components faster than another part of the process can cope. Identifying the bottlenecks leads you to set work in progress (WIP) limits, so that you get an overall more efficient system.

Increasingly kanban is used as an agile software development approach, too, especially where support work (like fixing bugs) needs to be balanced with incremental enhancement (like adding new features).

I'm a big advocate of kanban when you have a system that needs to be maintained and improved by the same team at the same time. Rather than the sprint-based approach of scrum (where the next sprint's stories or features to be delivered are agreed up front), kanban lets individuals deal with incidents or problems that need investigation and bug fixing when urgent and important. Then, when someone has capacity, they can just go to the board and pull down the next feature to develop or test.

SO, HOW DID WE USE IT?

One of the key tenets of kanban is that you visualise your workflow, so we put together a whiteboard with columns: Icebox; To Do Next; In Process; Done; and also a section called Blocked. Then, for each thing that needed to happen in the house, we put it on a Post-it note and initially chucked them all in the Icebox — a collection with no priority assigned yet.

Each week we looked at the Icebox and pulled out a set of things that we felt should be done next. This was pulled into the To Do Next column, and then each time either of us had some time, we could just pull a new thing over into

the In Process column. We agreed to review at the end of each week and move things to Done together, and to talk about whether this kanban approach was working for us or not.

We quickly learned for ourselves why kanban has WIP limits as a key tenet — it's tempting to pull everything into the To Do Next column, but that's unrealistic. And trying to do more than one or two things each at a given time isn't terribly productive owing to the cost of task switching. So we tend to limit our To Do Next to about seven items, and our In Process to about four (a max of two each, basically).

We use the Blocked column when something can't be completed — perhaps we can't fix something because we discovered we don't have the required tools or supplies, or if we're waiting for a call back from a plumber. But it's nice to put it to one side, knowing that it won't be forgotten.

WHAT HELPED THE MOST?

It wasn't so much the visualisation that helped us to see what we needed to do, but the conversation that happened when we were agreeing priorities, moving them to In Process and then on to Done made the biggest

difference. Getting clear on the order of importance really is invaluable — as is getting clear on what Done really means!

The Blocked column is also great, as it helps us keep track of things we need to do outside the house to make sure we can make progress. We also found it really helpful to examine the process itself and figure out whether it was working for us. For instance, one thing we realised is it's worth tracking some regular tasks that need time invested in them (like taking recycling that isn't picked up to the recycling centre) and these used to cycle around and around. So they were moved to Done as part of our weekly review, but then immediately put back in the Icebox to float back to the top again at a relevant time.

But the best thing of all? That moment where we get to mark something as **done**! It's immensely satisfying to review at the end of the week and have a physical marker of the progress you've made.

All in all, a home kanban board turned out to be a very effective way to *pull* tasks through stages rather than always trying to plan them out in advance, and definitely made collaboration on our home tasks significantly smoother. Give it a try!

ABOUT THE AUTHOR



Meri Williams is a geek, a manager and a manager of geeks. She's led teams ranging in size from 30 to 300, mostly with folks spread across the world. After starting her career as a developer, she moved on to project and then product management before moving into engineering and operations management.

A published author and speaker, she sponsors scholarships to encourage more young women into STEM careers in her hometown of Stellenbosch, South Africa. You can follow her on Twitter at [@Geek_Manager](#) or read her blog at [blog.geekmanager.co.uk](#)

15. In Their Own Write: Web Books and their Authors

Owen Gregory

24ways.org/201315

The currency of written communication — words on the page, words on the screen — comprises many denominations. To further our ends in web design and development, we freely spend and receive several: tweets aphoristic and trenchant, banal and perfunctory; blog posts and articles that call us to action or reflection; anecdotes, asides, comments, essays, guides, how-tos, manuals, musings, notes, opinions, stories, thoughts, tips pro and not-so-pro. So many, many words.

Our industry (so much more than this, but what on earth are we, collectively?), our community thrives on writing and sharing knowledge and experience. 24 ways is a case in point. Everyone can learn and contribute through reading and writing — it's what we've always done.

To web authors and readers seeking greater returns, though, broader culture has vouchsafed an enduring and singular artefact: the book.

Last month I asked a small sample of web book authors if they would be prepared to answer a few questions; most of them kindly agreed. In spirit, the survey was informal: I had neither hypothesis nor unground axe. I work closely with writers — and yes, I've edited or copy-edited books by several of the authors I surveyed — and wanted to share their thoughts about what it was like to write a book ("...it was challenging to find a coherent narrative"), why they did it ("Who wouldn't want to?") and what they learned from the experience ("That I could!").

REASONS FOR WRITING A BOOK

In web development the connection between authors and readers is unusually close and immediate. Working in our medium precipitates a unity that's rare elsewhere. Yet writing and publishing a book, even during the current books revolution, is something only a few of us attempt

and it remains daunting and a little remote. What spurs an author to try it? For some, it's a deeply held resistance to prevailing trends:

I felt that designers and developers needed to be shaken out of what seemed to me had been years of stagnation.

—Andrew Clarke

Or even a desire to protect us from ourselves:

I felt that without a book that clearly defined progressive enhancement in a very approachable and succinct fashion, the web was at risk. I was seeing Tim Berners-Lee's vision of universal availability slip away...

—Aaron Gustafson

Sometimes, there's a knowledge gap to be filled by an author with the requisite excitement and need to communicate. Jon Hicks took his "pet subject" and was "enthused enough to want to spend all that time writing", particularly because:

...there was a gap in the market for it. No one had done it before, and it's still on its own out there, with no competition. It felt like I was able to contribute something.

Cennydd Bowles felt a professional itch at a particular point in his career, understanding that

[a]s a designer becomes more senior, they start looking for ways to scale the effects of their work. For some, that leads into management. For others, into writing.

Often, though, it's also simply a personal challenge and ambition to explore a subject at length and create something substantial. Anna Debenham describes a motivation shared by several authors:

To be able to point to something more tangible than an article and be able to say “I did that.”

That sense of a book's significance, its heft and gravity even, stems partly from the cultural esteem which honours books and their authors. Books have a long history as sources of wisdom, truth and power. Even with more books being published each year than ever before, writing one is still commonly considered a laudable achievement, including in our field.

CHALLENGES OF WRITING A BOOK

Received wisdom has it that writing online should be brief and chunky and approachable: get to the point; divide it all up; subheadings and lists are our friends; write like you're talking; no one has time to read. Much of such advice is true. Followed well, it lends our writing punch and pith, vigour and vim. The web is nimble, the web *keeps up*, and it suits what we write about developing for it. It's

perfect for delivering our observations, queries and investigations into all the various aspects of the work, professional and personal.

Yet even for digital natives like web authors, books printed and electronic retain an attractive glister.

Ideas can be developed more fully, their consequences explored to greater depth and extended with more varied examples, and the whole conveyed with more eloquence, more style. Why shouldn't authors delay their conclusions if the intervening text is apposite, rich with value and helps to flesh out the skeleton of an argument?

Conclusions might or might not be reached, of course, but a writer is at greater liberty in a book to digress in tangential and interesting ways.

Writing a book involves committing time, energy, thought and money. As Brian Suda found, it can be tough "getting the ideas out of my head into a cohesive blob of text."

Some authors end up talking to themselves...

It helps me to keep a real person in mind, someone who I'm talking to as I write. Sometimes I have the same conversations over and over in my head.

—Andrew Clarke

...while others are thinking ahead, concerned with how their book will be received:

Would anyone want to read it? Would they care? Would it be respected by my peers?

—Joe Leech

Challenges that arose time and again included “starting” and “getting words on the page” as well as “knowing when to stop” or “letting go”. Personal organization problems and those caused by publishers were also widely mentioned. Time loomed large. Making time, finding time. Giving up “sleep and some sanity” and realizing “it will take you far, far, far longer than you naively assumed”. Importantly, writing time is time away from gainful employment: Aaron Gustafson found the hardest thing about writing a book to be “the loss of income while I was writing.”

PERILS AND PLEASURES OF EDITING

Editing, be it structural, technical or copy editing, is founded on reciprocity. Without openness and a shared belief that the book is worthwhile, work can founder in acrimony and mistrust. Editors are a book’s first and most critical (in every sense) readers. Effective and perceptive editing makes a book as good as it can be, finding the book within the draft like sculpture reveals the statue in the stone.

A good editor calls you out on poor assumptions and challenges you to really clarify your thinking. Whilst it can be difficult during the process to have your thinking challenged, it's always been worth it — for me personally — in the long run. A good editor also reins you in when you've perhaps wandered off track or taken a little too long to make a point.

—Christopher Murphy

Andy Croll found editing “all positive” and Aaron Gustafson loves “working with a strong editor [...] I want someone to tell it to me straight.” But it can be a rollercoaster, “both terrifying and the real moment of elation”. Mixed emotions during the editing process are common:

It was very uncomfortable! I knew it was making the work stronger, but it was awkward having my inconsistencies and waffle picked apart.

—Jon Hicks

It can be distressing to have written work looked over by a professional, particularly for first-time book authors whose expertise lies elsewhere:

I was a little nervous because I don't consider myself a skilled writer — I never dreamed of becoming an author. I'm a designer, after all.

—Geri Coady

Communication is key, particularly when it comes to checking or changing the author's words.

I like a good banter between me and the tech editor — if we can have a proper argument in Word comments, that's great.

—Rachel Andrew

But if handled poorly, small battles can break out. Rachel Andrew again:

However, having had plenty of times where the technical editor has done nothing more than give a cursory glance, I started to leave little issues in for them to spot. If they picked them up I knew they were actually testing the code and I could be sure the work was being properly tech edited. If they didn't spot them, I'd find someone myself to read through and check it!

A major concern for writers is that their voices will be altered, filtered, mangled or otherwise obscured by the editing process. Good copy editing must remain unnoticed while enhancing the author's voice in print. Donna Spencer appreciated the way her editor "tidied up

my work and made it a million times better, but left it sounding exactly like me.” Similarly, Andrew Travers “was incredibly impressed at how well my editor tightened up my own writing without it feeling like another’s voice” and Val Head sums up the consensus that:

the editor was able to help me express what I was trying to say in a better way [...] I want to have editors for everything now.

At the keyboard, keep your friends close, but your editors closer.

PUBLISHING AND PUBLISHERS

Conditions ought to militate against the allure of writing a book about web design and development. More books are published each year than ever before, so readerships elude new authors and readers can struggle to find authors to trust in their fields of interest. New spaces for more expansive online writing about working on and with the web are opening up (sites like **Contents Magazine** and **STET**), and seminal online web development texts are emerging. Publishing online is simple, far-reaching and immediate.

Much more so than articles and blog posts, books take time to research, write and read; add the complexity of commissioning, editing, designing, proofreading, printing, marketing and distribution processes, and it can take

many months, even years to publish. The ceaseless headlong momentum of the web can leave articles more than a few weeks old whimpering in its wake, but updating them at least is straightforward; printed books about web development can depreciate as rapidly as the technology and techniques they describe, while retaining the “terrifying permanence that print bestows: your opinions will follow you forever”.

So much moves on, and becomes out of date. Companies featured get bought by larger companies and die, techniques improve and solutions featured become terribly out of date. Unlike a website, which could be updated continuously, a book represents the thinking ‘at that time’.

—Jon Hicks

Publishers work hard to mitigate these issues, promoting new books and new authors, bringing authors and readers together under a trusted banner. When a publisher packages up and releases a writer’s words, it confers a seal of approval and “badge of quality”, very important to new authors.

Publishers have other benefits to offer, from expert knowledge:

My publisher was extraordinarily supportive (and patient). Her expertise in my chosen subject was both a pressure (I didn't want to let her down) and a reassurance (if she liked it, I knew it was going to be fine).

—Andrew Travers

...to systems and support mechanisms set up specifically to encourage writers and publish books:

Working as a team means you're bringing in everyone's expertise.

—Chui Chui Tan

As a writer, the best part about writing for a publisher was the writing infrastructure offered.

—Christopher Murphy

There can be drawbacks, however, and the occasional horror story:

We were just one small package on a huge conveyor belt. The publisher's process ruled all.

—Cennydd Bowles

It's only looking back I realise how poorly some publishers treat writers — especially when the work is so poorly remunerated.

My worst experience was when a publisher decided, after I had completed the book, that they wanted to push a different take on the subject than the brief I had been given. Instead of talking to me, they rewrote chunks of my words, turning my advice into something that I would never have encouraged. Ultimately, I refused to let the book go out under my name alone, and I also didn't really promote the book as I would have had to point out the things I did not agree with that had been inserted!

—Rachel Andrew

Self-publishing is now a realistic option for web authors, and can offers “complete control over the end product” as well as the possibility of earning more than a “pathetic author revenue percentage”. There can be substantial barriers, of course, as self-publishing authors must face for themselves the risks and challenges conventional publishers usually bear. Ideally, creating a book is a collaboration between author and publisher. Geri Coady found that “working with my publisher felt more like working with a partner or co-worker, rather than working for a boss.”

WISE WORDS

So, after meeting the personal costs of writing and publishing a web book — fear, uncertainty, doubt, typing (so much typing) — and then smelling the roses of success, what's left for an author to say? Some words, perhaps, to people thinking of writing a book.

Donna Spencer identifies a stumbling block common to many writers with an insight into the writing process:

Having talked to a lot of potential authors, I think most have the problem that they haven't actually figured out the 'answer' to their premise yet. They feel like they are stuck in the writing, but they are actually stuck in the thinking.

For some no-nonsense, straightforward advice to cut through any anxiety or inadequacy, Rachel Andrew encourages authors to "treat it like any other work. There is no mystery to writing, you just have to write. Schedule the time, sit down, write words." Tim Brown notes the importance of the editing process to refine a book and help authors reach their readers:

Hire good editors. Editors are amazing thinkers who can vastly improve the quality and clarity of a piece of writing.

We are too much beholden to the practical demands and challenges of technology, so Aaron Gustafson suggests a writer should “favor philosophies over techniques and your book will have a longer shelf life.”

Most intimations of renown and recognition are nipped in the bud by Joe Leech’s warning: “Don’t expect fame and fortune.” Although Cennydd Bowles’ bitter experience can be discouraging:

The sacrifices required are immense. You probably won’t make it.

...he would do things differently for a future book:

I would approach the book with [...] far more concern about conveying the damn joy of what I do for a living.

The pleasure of writing, not just having written is captured by James Chudley when he recalls:

How much I enjoy writing and also how much I enjoy the discipline or having a side project like this. It’s a really good supplement to working life.

And Jon Hicks has words that any author will find comforting:

It will be fine. Everything will be fine. Just get on with it!



As the web expands effortlessly and ceaselessly to make room for all our words, yet it can also discourage the accumulation of any particular theme in one space, dividing rich seams and scattering knowledge across the web's surface and into its deepest reaches. How many words become weightless and insubstantial, signals lost in the constant white noise of indistinguishable voices, unloved, unlinked? The web forgets constantly, despite the (somewhat empty) promise of digital preservation: articles and data are sacrificed to expediency, profit and apathy; online attention, acknowledgement and interest wax and wane in days, hours even.

Books can encourage deeper engagement in readers, and foster faith in an author, particularly if released under the imprint of a recognized publisher within the field. And books are changing. Although still not widely adopted, EPUB3 is the new standard in ebooks, bringing with it new possibilities for interaction and connection: readers with the text; readers with readers; and readers with authors. EPUB3 is built on HTML, CSS and JavaScript – sound familiar? In the past, we took what we could from the printed page to make the web; now books are rubbing up against what we've made.

So: a book.

Ever thought you could write one? Should write one?
Would?



I'd like to thank all the authors who wrote their books and answered my questions.

- Rachel Andrew · *CSS3 Layout Modules, The CSS3 Anthology* and more
- Cennydd Bowles · *Undercover User Experience Design*, with James Box
- Tim Brown · *Combining Typefaces*
- James Chudley · *Usability of Web Photos*
- Andrew Clarke · *Hardboiled Web Design*
- Geri Coady · *Colour Accessibility*
- Andy Croll · *HTML Email*
- Anna Debenham · *Front-end Style Guides*
- Aaron Gustafson · *Adaptive Web Design*
- Val Head · *CSS Animations*
- Jon Hicks · *The Icon Handbook*
- Joe Leech · *Psychology for Designers*
- Christopher Murphy · *The Craft of Words*, with Niklas Persson
- Donna Spencer · *Information Architecture, Card Sorting and How to Write Great Copy for the Web*
- Brian Suda · *Designing with Data*
- Chui Chui Tan · *International User Research*

- Andrew Travers · *Interviewing for Research*

ABOUT THE AUTHOR



Owen Gregory is an editor, website designer and musician living in Birmingham, UK. He started designing for the web in 1998 and established his small business **Full Cream Milk** in 2006. Prior to that, Owen studied English and writing to master's level, and he now brings these two interests together for your friendly neighbourhood web book publishers, like Five Simple Steps. He tweets as **@FullCreamMilk** because FullCreamMilkMan is too long for Twitter.

Owen is what is sometimes called a classically trained musician, and he plays oboe and cor anglais (neither English nor a horn) in a number of non-professional orchestras.

Oh, and Andy Clarke once thanked Owen for “being Lewis to my Morse”. Which is better than being Robin to his Batman.

16. Credits and Recognition

Geri Coady

24ways.org/201316

A few weeks ago, I saw a friendly little tweet from a business congratulating a web agency on being nominated for an award. The business was quite happy for them and proud to boot — they commented on how the same agency designed their website, too.

What seemed like a nice little shout-out actually made me feel a little disappointed. Why? In reality, I knew that the web agency didn't actually design the site — *I did*, when I worked at a different agency responsible for the overall branding and identity.

I certainly wasn't disappointed at the business — after all, saying that someone designed your site when they were responsible for development is an easy mistake to make.

Chances are, the person behind the tweets and status updates might not even know the difference between words like design and development.

What really disappointed me was the reminder of how many web workers out there never explain their roles in a project when displaying work in a portfolio. If you're strictly a developer and market yourself as such, there might be less room for confusion, but things can feel a little deceptive if you offer a wide range of services yet never credit the other players when collaboration is part of the game. Unfortunately, this was the case in this situation. Whatever happened to credit where credit's due?

ADVERTISING ATTRIBUTION

Have you ever thumbed through an advertising annual or browsed through the winners of an advertising awards website, like the campaign below from Copenhagen Chocolate on **Advertising Age**? If so, it's likely that you've noticed some big differences in how the work is credited.



Kopenhagen Chocolate

Featured in:
2013 Advertising Annual

Pedro Konishi/Samyr Souen, art directors
Caio Borges, writer
[Luciano Lincoln/Wilson](#)
[Mateos/Sergio Valente/Marco](#)
[Versolato](#), creative directors
[Sergio Prado](#), photographer
Lucas Louzada/Fernanda
Oliveira/Samyr Souen, illustrators
[Clariana Costa/Daniela Strang](#),
art buyers
[DOB Brasil](#), ad agency
Kopenhagen Chocolates, client

16-1. Everyone involved in a creative advertising project is mentioned.

Art directors, writers, creative directors, photographers, illustrators and, of course, the agency all get a fair shot at fifteen minutes of fame. Why can't we take this same idea and introduce it to our own showcases?

CREDITING ON CLIENT SITES

Ah, the good old days of web rings, guestbooks, and under construction GIFs, when slipping in a cheeky “designed by” link in the footer of your masterpiece was just another common practice. These days most clients, especially larger companies and corporations, aren't willing to have any names on their site except their own.

If you'd still like to leave a little proof of authorship on a website, consider adding a *humans.txt* file to the root of the site and, if possible, add an author tag in the <head> of the site:

```
<link type="text/plain" rel="author" href="http://domain/  
humans.txt">
```

It's a great way to add more detailed information than just a meta name without being intrusive. The example on the humanstxt.org website serves to act as a guideline, but how much detail you add is completely up to you and your team.

```
/* TEAM */
Chef:Juanjo Bernabeu
Contact: hello [at] humanstxt.org
Twitter: @juanjobernabeu
From:Barcelona, Catalonia, Spain

UI developer: Maria Macias
Twitter: @maria_ux
From:Barcelona, Catalonia, Spain

One eyed illustrator: Carlos Mañas
Twitter: @oneeyedman
From:Madrid, Spain

Standard Man: Abel Cabans
Twitter: @abelcabans
From:Barcelona, Catalonia, Spain

Web designer: Abel Sutilo
Twitter: @abelsutilo
From:Sevilla, Andalucia, Spain

/* THANKS */

(First) EN Translator: Jos Flores
Twitter: @prosciuttos
From: Barcelona, Catalonia, Spain

CA Translator: Eva AC
Twitter: @evaac
From:Barcelona, Catalonia, Spain

EN Translator: Marta Armada
Twitter: @martuishere
From: Barcelona, Catalonia, Spain
```

16-2. Part of the *humans.txt* file on humanstxt.org

Alternatively, you can use the **HTML5** `rel="author"` attribute to link to information about the author of the page in the form of a `mailto:` address, a link to a contact form, or a separate authors page.

CREDITING IN PORTFOLIOS

While *humans.txt* is a great approach when you're authoring a site, it's even more important to clearly define your role in your own portfolio.

While I believe it's proper etiquette to include the names of folks you collaborated with, sometimes it might not be necessary (or even possible) to list every single person, especially if you've worked with a large agency.

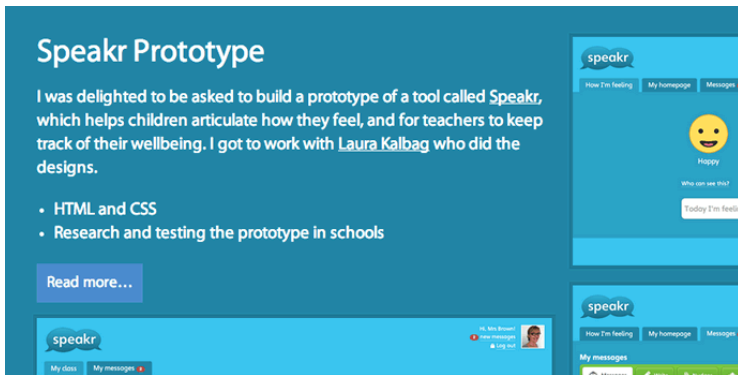
"Fake it till you make it" is not a term that should apply to your portfolio. Clearly stating your own responsibilities means that nobody else browsing your work samples will assume that you did more than your actual share, and being ambiguous about your role isn't fair to yourself, or others.

Before adding any work to your portfolio, ensure that you have permission from your client. Even if you included a clause in your contract about being allowed to post your work online, it's always best to double-check. Sometimes you might not know if your work has been officially launched, and leaking something before it's ready is bound to make a client frown.

Examples

There are plenty of portfolios out there that we can use for inspiration. Here are some examples that I like from other folks in the web industry:

ANNA DEBENHAM



16-3. In her portfolio, Anna outlines her responsibilities and those of others.

In the description, Anna clearly explains her duties of doing the HTML and CSS, along with performing research and testing the prototype in schools. She also credits Laura Kalbag for the design work.

NAOMI ATKINSON DESIGN

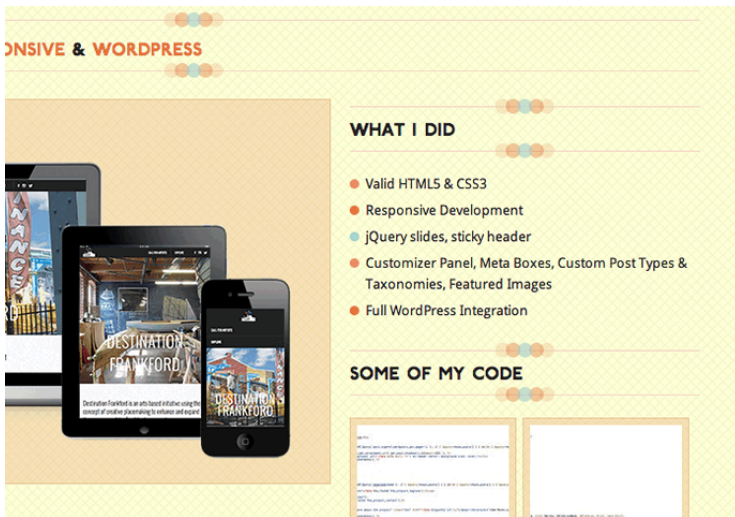
The work portfolio of Naomi Atkinson Design is short and to the point — they were responsible for the iPhone app design and IA for Artspotter.



16-4. The portfolio of Naomi Atkinson Design states clearly what they did.

AMBER WEINBERG

Amber Weinberg is strictly a developer, but a potential client could see her portfolio and assume she might be a designer as well. To avoid any misunderstandings, she states her roles up front in a section called “What I Did,” supported by examples of her code.



16-5. Amber Weinberg sets out all her roles in each of her portfolio's case studies.

WHAT IF SOMEONE DOESN'T WANT TO BE CREDITED?

Let's face it — we've all been there. A project, for whatever reason, turns out to be an absolute disaster and we don't feel like it's an accurate representation of the quality of our work.

If you're crediting someone else but suspect they might rather pretend it never happened, be sure to drop them a line and ask if they'd like to be included. And, if someone contacts you and asks to remove their name, don't feel offended — just politely remove it.

GET UPDATING!

Now that the holiday season is almost here, many of you might be planning to set aside some time for personal projects. Grab yourself a gingerbread latte and get those portfolios up to date. Remember, It doesn't have to be long-winded, just honest. Happy holidays!

ABOUT THE AUTHOR



Geri Coady is a colour-obsessed illustrator and designer from Newfoundland, Canada. She is a former Art Director at a Canadian advertising agency and is now pursuing her own clients through her website at hellogeri.com. Geri loves chatting about nerdy things on **Twitter** and has shared her thoughts in publications such as net magazine, The Pastry Box Project, and Digital Arts. She's the author of the **Pocket Guide to Colour Accessibility** from Five Simple Steps, a sometimes-illustrator for A List Apart, and was voted Net Magazine's Designer of the Year in 2014.

17. Project Hubs: A Home Base for Design Projects

Brad Frost

24ways.org/201317

SCENE: A design review meeting. Laptop screens. Coffee cups.

Project manager: Hey, did you get my email with the assets we'll be discussing?

Client: I got an email from you, but it looks like there's no attachment.

PM: Whoops! OK. I'm resending the files with the attachments. Check again?

Client: OK, I see them. It's *homepage_v3_brian-edits_FINAL_for-review.pdf*, right?

PM: Yeah, that's the one.

Client: OK, hang on, Bill's going to print them out. (3-minute pause. Small talk ensues.)

Client: Alright, Bill's back. We're good to start.

Brian: Oh, actually those homepage edits we talked about last time are in the *homepage_v4_brian_FINAL_v2.pdf* document that I posted to Basecamp earlier today.

Client: Oh, OK. What message thread was that in?

Brian: Uh, I'm pretty sure it's in "Homepage Edits and Holiday Schedule."

Client: Alright, I see them. Bill's going back to the printer. Hang on a sec...



This is only a slightly exaggerated version of my experience in design review meetings.

The design project dance is a sloppy one. It involves a slew of email attachments, PDFs, PSDs, revisions, GitHub repos, staging environments, and more. And while tools like **Basecamp** can help manage all these moving parts, it can still be incredibly challenging to extract only the important bits, juggle deliverables, and see how your project is progressing.

Enter project hubs.

PROJECT HUBS

A project hub consolidates all the key design and development materials onto a single webpage presented in reverse chronological order. The timeline lives online (either **publicly available** or password protected), so that everyone involved in the team has easy access to it.

Greater Pittsburgh Community Food Bank Redesign Timeline

November 22nd, 2013

Sitemap, HTML Wireframes, Style Tiles

[View sitemap](#)

[View HTML Wireframes](#)

[View style tiles](#)

[Type explorations](#)

November 18th, 2013

Blog post: Scope Components, Not Pages

[View post](#)

November 17th, 2013

URL Design and Mapping

[View URL Design](#)

[View URL Mapping](#)

October 3rd, 2013

Blog post: Establishing Design Direction

[View post](#)

October 2nd, 2013

Stakeholder meeting & design exercises

[View Exercise Recap](#)

September 24th, 2013

Blog post: Pittsburgh Food Bank Open Redesign

[View post](#)

September 23th, 2013

Blog post: Designing In The Open

[View post](#)

September 19th, 2013

Blog post: Greater Pittsburgh Community Food Bank Open Redesign

[View post](#)

September 18th, 2013

Contract signed

September 13th, 2013

Meeting reviewing contract, designing in the open, strategy and atomic design

[View Meeting Notes](#)

September 13th, 2013

Set up Pattern Lab

[View Pattern Lab](#)

August 29th, 2013

Deliver Contract

[View Contract](#)

17-1. A project hub.

I was introduced to project hubs after seeing Dan Mall's open redesign of Reading Is Fundamental. Thankfully, I had a chance to work with Dan on two projects where I got to see firsthand how beneficial a project hub can be. Here's what makes a project hub great:

- Serves as a centralized home base for the project
- Trains clients and teams to decide in the browser
- Easily and visually view project's progress
- Provides an archive for project artifacts

A HOME BASE

Your clients and colleagues can expect to get the latest and greatest updates to your project when visiting the project hub, the same way you'd expect to get the latest information on a requested topic when you visit a Wikipedia page. That's the beauty of URIs that don't change.

Creating a project hub reduces a ton of email volley nonsense, and eliminates the need to produce files and directories with staggeringly ridiculous names like *design/12.13.13/team/brian/for_review/_FINAL/styletile_121313_brian-edits-final_v2_FINAL.pdf*. The team can simply visit the project hub's URL and click the link to

whatever artifact they need. Need to make an update? Simply update the link on the project hub. No more email tango and silly file names.

DECIDING IN THE BROWSER

Let's change the phrase "designing in the browser" to "deciding in the browser."

Dan Mall

We make websites, but all too often we find ourselves looking at web design artifacts in abstractions. We email PDFs to each other, glance at mockup JPGs on our desktops, and of course kill trees in order to print out designs so that we can scribble in the margins. All of these practices subtly take everyone further and further away from the design's eventual final resting place: the browser.

Because a project hub is just a simple webpage, reviewing designs is as easy as clicking some links, which keep your clients and teams in the browser.

You can keep people in the browser with yet another clever trick from the wily Dan Mall: instead of sending clients PDFs or JPGs, he created a simple webpage and tossed his static visuals into the template ([you can view an example here](#)). This forces clients to review web design work in the browser rather than launching a PDF viewer or Preview.

Now this all might sound trivial to you (“Of course my client knows that we’re designing a website!”), but keeping the design artifacts in the browser subconsciously helps remind everyone of the medium for which you’re designing, which helps everyone focus on the right aspects of the design and have the right conversations.

PROGRESS OVER TIME

When you’re in the trenches, it’s often hard to visualize how a project is progressing. Tools like Basecamp include discussions, files, to-dos, and more, which are all great tools but also make things a bit noisy. Project hubs provide you and your clients a quick and easy way to see at a glance how things are coming along. Teams can rest assured they’re viewing the most current versions of designs, and managers can share progress with stakeholders simply by providing a link to the project hub.

Over time, a project hub becomes an easily accessible archive of all the design decisions, which makes it easy to compare and contrast different versions of designs and prototypes.

SETTING UP A PROJECT HUB

Setting up your own project hub is pretty simple. Simply create a webpage with some basic styles and branding. I've created a project hub template that's **available on GitHub** if you want a jump-start.

Publish the webpage to a URL somewhere that makes sense (we've found that a subdomain of your site works quite well) and share it with everyone involved in the project. Bookmark it. Let everyone know that this is where design updates will be shared, and that they can always come back to the project hub to track the project's progress.

When it comes time to share new updates, simply add a new node to the timeline and republish the webpage. Simple FTPing works just fine, but it might make sense to keep track of changes using version control. Our **project hub** for our open redesign of the Pittsburgh Food Bank is managed on GitHub, which means that I can make edits to the hub right from GitHub. Thanks to the magical wizardry of **webhooks**, I can **automatically deploy** the project hub so that everything stays in sync. That's the fancy-pants way to do it, and is certainly not a requirement. As long as you're able to easily make edits and keep your project hub up to date, you're good to go.

SO THAT'S THE HUBBUB

Project hubs can help tame the chaos of the design process by providing a home base for all key design and development materials. Keep the design artifacts in the browser and give clients and colleagues quick insight into your project's progress.

Happy hubbing!

ABOUT THE AUTHOR



Brad Frost is a web designer, speaker, writer, and consultant located in beautiful Pittsburgh, PA. He's passionate about creating Web experiences that look and function beautifully on the never-ending stream of connected devices, and is constantly **tweeting**, **writing** and **speaking** about it. He's also created some tools and resources for web designers, including **This Is Responsive**, **Pattern Lab**, **Mobile Web Best Practices**, and **WTF Mobile Web**.

18. Get Started With GitHub Pages (Plus Bonus Jekyll)

Anna Debenham

24ways.org/201318

After several failed attempts at getting set up with GitHub Pages, I vowed that if I ever figured out how to do it, I'd write it up. Fortunately, I did eventually figure it out, so here is my write-up.

WHY I THINK GITHUB PAGES IS COOL

Normally when you host stuff on GitHub, you're just storing your files there. If you push site files, what you're storing is the code, and when you view a file, you're viewing the code rather than the output. What GitHub Pages lets you do is store those files, and if they're HTML files, you can view them like any other website, so there's no need to host them separately yourself.

GitHub Pages accepts static HTML but can't execute languages like PHP, or use a database in the way you're probably used to, so you'll need to output static HTML files. This is where templating tools such as **Jekyll** come in, which I'll talk about later.

The main benefit of GitHub Pages is ease of collaboration. Changes you make in the repository are automatically synced, so if your site's hosted on GitHub, it's as up-to-date as your GitHub repository. This really appeals to me because when I just want to quickly get something set up, I don't want to mess around with hosting; and when people submit a pull request, I want that change to be visible as soon as I merge it without having to set up web hooks.

BEFORE YOU GET STARTED

If you've used GitHub before, already have an account and know the basics like how to set up a repository and clone it to your computer, you're good to go. If not, I recommend getting familiar with that first. The GitHub site has **extensive documentation on getting started**, and if you're not a fan of using the command line, the official GitHub apps for **Mac** and **Windows** are great.

I also found **this tutorial about GitHub Pages by Thinkful** really useful, and it contains details on how to turn an existing repository into a GitHub Pages site.

Although this involves a bit of using the command line, it's minimal, and I'll guide you through the basics.

SETTING UP GITHUB PAGES

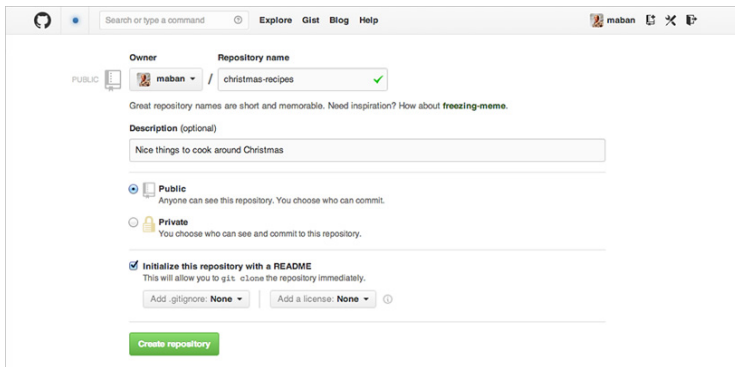
For this demo we're going to build a Christmas recipe site — nothing complex, just a place to store recipes so we can share them with people, and they can fork or suggest changes to ones they like. My GitHub username is **maban**, and the project I've set up is called **christmas-recipes**, so once I've set up GitHub Pages, the site can be found here: <http://maban.github.io/christmas-recipes/>

You can set up a custom domain, but by default, the URL for your GitHub Pages site is *your-username.github.io/your-project-name*.

Set up the repository

The first thing we're going to do is create a new GitHub repository, in exactly the same way as normal, and clone it to the computer. Let's give it the name *christmas-recipes*. There's nothing in it at the moment, but that's OK.

Get Started With GitHub Pages (Plus Bonus Jekyll)

A screenshot of the GitHub 'Create new repository' page. The page has a header with the GitHub logo, a search bar, and navigation links for 'Explore', 'Gist', 'Blog', and 'Help'. The user 'maban' is logged in. The form includes fields for 'Owner' (maban) and 'Repository name' (christmas-recipes), which is marked as valid with a green checkmark. There is a 'Description (optional)' text area containing 'Nice things to cook around Christmas'. Below this are radio buttons for 'Public' (selected) and 'Private'. A checkbox for 'Initialize this repository with a README' is checked. At the bottom, there are dropdown menus for 'Add .gitignore: None' and 'Add a license: None', followed by a green 'Create repository' button.

After setting up the repository on the GitHub website, I cloned it to my computer in my *Sites* folder using the GitHub app (you can clone it somewhere else, if you want), and now I have a local repository synced with the remote one on GitHub.

Navigate to the repository

Now let's open up the command line and navigate to the local repository. The easiest way to do this in Terminal is by typing `cd` and dragging and dropping the folder into the terminal window and pressing **Return**. You can refer to Chris Coyier's GIF illustrating this very same thing, from last week's 24 ways article "Grunt for People Who Think Things Like Grunt are Weird and Hard" (which is excellent).

So, for me, that's...

```
cd /Users/Anna/Sites/christmas-recipes
```

Create a special GitHub Pages branch

So far we haven't done anything different from setting up a regular repository, but here's where things change.

Now we're in the right place, let's create a *gh-pages* branch. This tells GitHub that this is a special branch, and to treat the contents of it differently.

Make sure you're still in the *christmas-recipes* directory, and type this command to create the *gh-pages* branch:

```
git checkout --orphan gh-pages
```

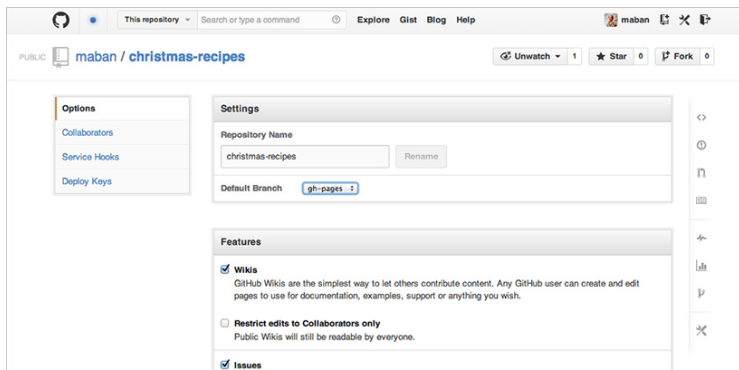
That `--orphan` option might be new to you. An orphaned branch is an empty branch that's disconnected from the branch it was created off, and it starts with no commits, making it a special standalone branch. `checkout` switches us from the branch we were on to that branch.

If all's gone well, we'll get a message saying Switched to a new branch 'gh-pages'.

You may get an error message saying you don't have admin privileges, in which case you'll need to type `sudo` at the start of that command.

Make *gh-pages* your default branch (optional)

The *gh-pages* branch is separate to the *master* branch, but by default, the *master* branch is what will show up if we go to our repository's URL on GitHub. To change this, go to the repository settings and select *gh-pages* as the default branch.



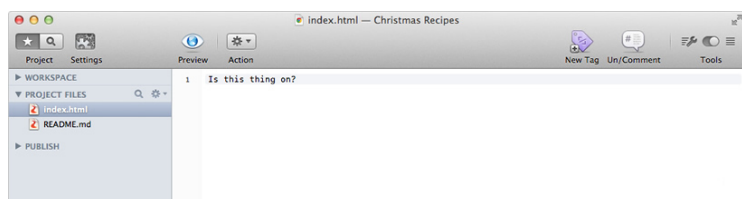
If, like me, you only want the one branch, you can delete the *master* branch by following Oli Studholme's tutorial. It's actually really easy to do, and means you only have to worry about keeping one branch up to date.

If you prefer to work from *master* but push updates to the *gh-pages* branch, Lea Verou has written up a quick tutorial on how to do this, and it basically involves working from the *master* branch, and using `git rebase` to bring one branch up to date with another.

At the moment, we've only got that branch on the local machine, and it's empty, so to be able to see something at our special GitHub Pages URL, we'll need to create a page and push it to the remote repository.

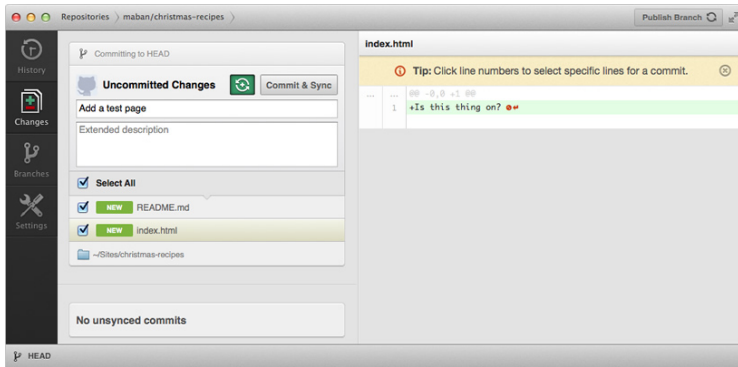
Make a page

Open up your favourite text editor, create a file called *index.html* in your *christmas-recipes* folder, and put some exciting text in it. Don't worry about the markup: all we need is text because right now we're just checking it works.



Now, let's commit and push our changes. You can do that in the command line if you're comfortable with that, or you can do it via the GitHub app. Don't forget to add a useful commit message.

Get Started With GitHub Pages (Plus Bonus Jekyll)



Now we're ready to see our gorgeous new site! Go to your-username.github.io/your-project-name and, hopefully, you'll see your first GitHub Pages site. If not, don't panic, it can take up to ten minutes to publish, so you could make a quick cake in a cup while you wait.

After a short wait, our page should be live! Hopefully that wasn't too traumatic. Now we know it works, we can add some proper markup and CSS and even some more pages.

If you're feeling brave, how about we take it to the next level...

SETTING UP JEKYL

Since GitHub Pages can't execute languages like PHP, we need to give it static HTML files. This is fine if there are only a few pages, but soon we'll start to miss things like PHP includes for content that's the same on every page, like headers and footers.

Jekyll helps set up templates and turn them into static HTML. It separates markup from content, and makes it a lot easier for people to edit pages collaboratively. With our recipe site, we want to make it really easy for people to fix typos or add notes, without having to understand PHP. Also, there's the added benefit that static HTML pages load really fast.

Jekyll isn't officially supported on Windows, but it is still possible to run it if you're prepared to get your hands dirty.

Install Jekyll

Back in Terminal, we're going to install Jekyll...

```
gem install jekyll
```

...and wait for the script to run. This might take a few moments. It might take so long that you get worried it's broken, but resist the urge to start mashing your keyboard like I did.

Get Jekyll to run on the repository

Fingers crossed nothing has gone wrong so far. If something did go wrong, don't give up! **Check this helpful post by Andy Taylor** – you probably just need to install something else first.

Now we're going to tell Jekyll to set up a new project in the repository, which is in my *Sites* folder (yours may be in a different place). Remember, we can drag the directory into the terminal window after the command.

```
jekyll new /Users/Anna/Sites/christmas-recipes
```

If everything went as expected, we should get this error message: Conflict: /Users/Anna/Sites/christmas-recipes exists and is not empty.

But that's OK. It's just upset because we've got that *index.html* file and possibly also a *README.md* in there that we made earlier. So move those onto your desktop for the moment and run the command again.

```
jekyll new /Users/Anna/Sites/christmas-recipes
```

It should say that the site has installed.

Check you're in the repository, and if you're not, navigate to it by typing `cd` , drag the *christmas-recipes* directory into terminal...

```
jekyll cd /Users/Anna/Sites/christmas-recipes
```

...and type this command to tell Jekyll to run:

```
jekyll serve --watch
```

By adding `--watch` at the end, we're forcing Jekyll to rebuild the site every time we hit **Save**, so we don't have to keep telling it to update every time we want to view the

changes. We'll need to run this every time we start work on the project, otherwise changes won't be applied. For now, wait while it does its thing.

Update the config file

When it's finished, we'll see the text press `ctrl-c` to stop. **Don't do that, though.** Instead, open up the directory. You'll notice some new files and folders in there. There's one called `_site`, and that's where all the site files are saved when they're turned into static HTML. **Don't touch the files in here** — they're the generated files and will get overwritten every time we make changes to pages and layouts.

There's a file in our directory called `_config.yml`. This has some settings we can change, one of them being what our base URL is. GitHub Pages will assume the base URL is above the project repository, so changing the settings here will help further down the line when setting up navigation links.

Replace the contents of the `_config.yml` file with this:

```
name: Christmas Recipes
markdown: redcarpet
pygments: true
baseurl: /christmas-recipes
```

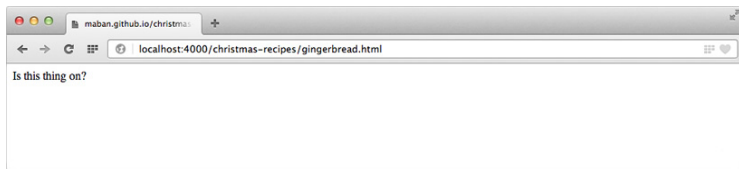
Set up your files

Overwrite the *index.html* file in the root with the one we made earlier (you might want to pop the *README.md* back in there, too).

Delete the files in the *css* folder — we'll add our own later.

View the Jekyll site

Open up your favourite browser and type *http://localhost:4000/christmas-recipes* in the address bar.



Check it out, that's your site! But it could do with a bit more love.

Set up the *_includes* files

It's always useful to be able to pull in snippets of content onto pages, such as the header and footer, so they only need to be updated in one place. That's what an *_includes* folder is for in Jekyll. Create a folder in the root called *_includes*, and within it add two files: *head.html* and *foot.html*.

In *head.html*, paste the following:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>{{ page.title }}</title>
    <link rel="stylesheet" href="{{ site.baseurl
  }}/css/main.css" >
  </head>
  <body>
```

and in *foot.html*:

```
</body>
</html>
```

Whenever we want to pull in something from the *_includes* folder, we can use `{% include filename.html %}` in the layout file — I'll show you how to set that up in next step.

Making layouts

In our directory, there's a folder called *_layouts* and this lets us create a reusable template for pages. Inside that is a *default.html* file.

Delete everything in *default.html* and paste in this instead:

```
{% include head.html %}

<h1>{{ page.title }}</h1>
```

```
{{ content }}

{% include foot.html %}
```

That's a very basic page with a header, footer, page title and some content. To apply this template to a page, go back into the *index.html* page and add this snippet to the very top of the file:

```
---
layout: default
title: Home
---
```

Now save the *index.html* file and hit **Refresh** in the browser. We should see a heading where `{{ page.title }}` was in the layout, which matches what comes after `title:` on the page itself (in this case, *Home*). So, if we wanted a subheading to appear on every page, we could add `{{ page.subheading }}` to where we want it to appear in our layout file, and a line that says `subheading:` This is a subheading in between the dashes at the top of the page itself.

Using Markdown for templates

Anything on a page that sits under the closing dashes is output where `{{ content }}` appears in the template file. At the moment, this is being output as HTML, but we can use Markdown instead, and Jekyll will convert that into

HTML. For this recipe site, we want to make it as easy as possible for people to be able to collaborate, and also have the markup separate from the content, so let's use Markdown instead of HTML for the recipes.

Telling a page to use Markdown instead of HTML is incredibly simple. All we need to do is change the filename from *.html* to *.md*, so let's rename the *index.html* to *index.md*. Now we can use Markdown, and Jekyll will output that as HTML.

Create a new layout

We're going to create a new layout called *recipe* which is going to be the template for any recipe page we create. Let's keep it super simple.

In the *_layouts* folder, create a file called *recipe.html* and paste in this:

```
{% include head.html %}

<main>

  <h1>{{ page.title }}</h1>

  {{ content }}

  <p>Recipe by <a href="{{
page.recipe-attribution-link }}">{{
page.recipe-attribution }}</a>.</p>
```

```
</main>

{% include nav.html %}

{% include foot.html %}
```

That's our template. The content that goes on the recipe layout includes a page title, the recipe content, a recipe attribution and a recipe attribution link.

Adding some recipe pages

Create a new file in the root of the *christmas-recipes* folder and call it *gingerbread.md*. Paste the following into it:

```
---
layout: recipe
title: Gingerbread
recipe-attribution: HungryJenny
recipe-attribution-link: http://www.opensourcefood.com/people/HungryJenny/recipes/soft-christmas-gingerbread-cookies
---

Makes about 15 small cookies.
```

Ingredients

- * 175g plain flour
- * 90g brown sugar
- * 50g unsalted butter, diced, at room temperature
- * 2 tbsp golden syrup
- * 1 egg, beaten
- * 1 tsp ground ginger

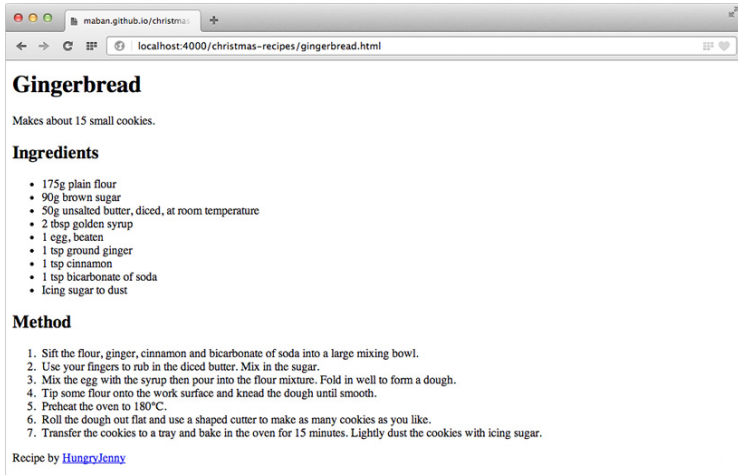
- * 1 tsp cinnamon
- * 1 tsp bicarbonate of soda
- * Icing sugar to dust

Method

1. Sift the flour, ginger, cinnamon and bicarbonate of soda into a large mixing bowl.
2. Use your fingers to rub in the diced butter. Mix in the sugar.
3. Mix the egg with the syrup then pour into the flour mixture. Fold in well to form a dough.
4. Tip some flour onto the work surface and knead the dough until smooth.
5. Preheat the oven to 180°C.
6. Roll the dough out flat and use a shaped cutter to make as many cookies as you like.
7. Transfer the cookies to a tray and bake in the oven for 15 minutes. Lightly dust the cookies with icing sugar.

The content is in Markdown, and when we hit **Save**, it'll be converted into HTML in the `_site` folder. Save the file, and go to <http://localhost:4000/christmas-recipes/gingerbread.html> in your favourite browser.

Get Started With GitHub Pages (Plus Bonus Jekyll)



18-1. As you can see, the Markdown content has been converted into HTML, and the attribution text and link has been inserted in the right place.

Add some navigation

In your `_includes` folder, create a new file called `nav.html`.

Here is some code that will generate your navigation:

```
<nav class="nav-primary" role="navigation" >
  <ul>
    {% for p in site.pages %}
      <li>
        <a {% if p.url == page.url %}class="active"{%
endif %} href="{ { site.baseurl } }{ { p.url } }">{ { p.title
}}</a>
      </li>
    {% endfor %}
  </ul>
</nav>
```

This is going to look for all pages and generate a list of them, and give the navigation item that is currently active a class of `active` so we can style it differently.

Now we need to include that navigation snippet in our layout. Paste `{% include nav.html %}` in *default.html* file, under `{{ content }}`.

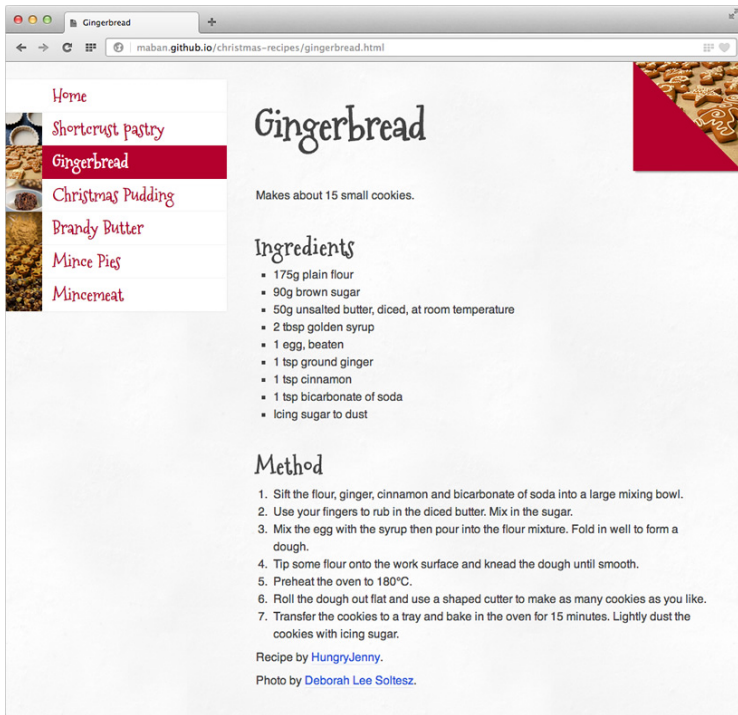
Push the changes to GitHub Pages

Now we've got a couple of pages, it's time to push our changes to GitHub. We can do this in exactly the same way as before. Check your special GitHub URL (*your-username.github.io/your-project-name*) and you should see your site up and running.

If you quit Terminal, don't forget to run `ekyll serve --watch` from within the directory the next time you want to work on the files.

Next steps

Now we know the basics of creating Jekyll templates and publishing them as GitHub Pages, we can have some fun adding more pages and styling them up.



18-2. Here's one I made earlier

I've only been using Jekyll for a matter of weeks, mainly for prototyping. It's really good as a content management system for blogs, and a lot of people host their Jekyll blogs on GitHub, such as Harry Roberts

By hosting the code so openly it will make me take more pride in it *and* allow me to work on it much more easily; no excuses now!

Overall, the documentation for Jekyll feels a little sparse and geared more towards blogs than other sites, but as more people discover the benefits of it, I'm sure this will improve over time.

If you're interested in poking about with some code, all the files from this tutorial are available on [GitHub](#).

ABOUT THE AUTHOR



Anna Debenham is a freelance front-end developer living in Brighton in the UK.

She's the author of *Front-end Style Guides*, and when she's not playing on them, she's testing as many game console browsers as she can get her hands on.

19. How to Write a Book

Jonathan Snook

24ways.org/201319

Were you recently inspired to write a book after reading Owen Gregory's compendium of author insights? Maybe so inspired to strike out on your own and self-publish?

Based on personal experience, writing a book is hard. It requires a great deal of research, experience, and patience. To be able to consolidate your thoughts and what you've learned into a sensible and readable tome is an admirable feat. To decide to self-publish and take on yourself all of the design, printing, distribution, and so much more is tantamount to insanity. Again, based on personal experience.

So, why might you want to self-publish?

If you've spent many a late night doing cross-browser testing just to know that your site works flawlessly in twenty-four different browsers — including Mosaic, of course — then maybe you'll understand the fun that comes from doing it all.

Working with a publisher, you're left to focus on one core thing: writing. That's a good thing. A good publisher has the right resources to help you get your idea polished and the distribution network to get your book on store shelves around the world. It's a very proud moment to be able to walk into a book store and see your book sitting there on the shelf.

Self-publishing can also be a wonderful process as you get to own it from beginning to end. Every decision is yours and if you're a control freak like me, this can be a very rewarding experience.

While there are many aspects to self-publishing, I'm going to speak to just one of them: creating an ebook.

FORMATS

In creating an ebook, you first need to decide what formats you wish to support. There are three main formats, each with their own pros and cons:

1. PDF
2. EPUB
3. MOBI

PDFs are supported on almost every device (Windows, Mac, Kindle, iPad, Android, etc.) and can even be a stepping stone to creating a print version of your book. PDFs allow for full typographic and design control, but at

the cost of needing to fit things into a predefined page layout. Is it US Letter or A4? Or is it a format that isn't easily printed by readers on their home printers?

EPUB is a more fluid format that is supported by the Apple iPad, iPhone, and now on the desktop with OS X Mavericks. It's also supported by Google Play for Android devices. While EPUB is supported on other devices, you're likely to choose EPUB because you're targeting your book at the Apple audience. The EPUB format is HTML-based with support for some CSS and even video and interactive elements. You can create very rich and exciting experiences using the EPUB format that just aren't possible with PDF or MOBI. However, if you decide to support multiple file formats, you'll likely find — as I did — that a consistent experience between all formats is easier to build and maintain, and therefore the extra benefits of interactivity go out the window.

MOBI is a format originally developed for the Mobipocket Reader but more popularly supported by the Amazon Kindle. If you're looking to attract the Kindle audience or publish to Amazon via the Kindle Direct Publishing platform then the HTML-based MOBI format is the format you'll want to go with.

Distribution will probably factor in heavily with what format you decide to go with. Many people I know who self-publish go with PDF only due to its ubiquity.

If you want to garner a wider audience by distributing via Amazon or the iBookstore then you'll need to think about supporting all three formats (as I did).

WHAT TOOLS SHOULD I USE?

I spent a lot of time figuring out the right toolset and finally got something that suits me just right.

In the past, when working with a publisher, I was given a Microsoft Word template that was passed back and forth between myself, the editor, and tech reviewer. This template has been the bane of any book writer that I've spoken to. Not every publisher is like that, though. Some publishers, like O'Reilly, use DocBook, an XML-based format that can be converted into PDF, EPUB, and MOBI.

Publishers already have a style guide and whether it's DocBook or a Word template, they have the tools already in place to easily convert your work into multiple formats.

Self-publishing means that you'll likely have to do a lot of tweaking to get things looking and working the way you want them to. I tried **DocBook** and the open source export tools didn't create HTML to my liking. Fixing even the most mundane things required fiddling with XSL transformations for hours on end. Not the way I like to spend my time. I can only imagine the hoops I would've had to go through to get a PDF to look half-decent.

Tools like Pages or **Scrivener** offer up the ability to publish to multiple formats, too, but none offered me the control over the output that I truly desired. Have a mentioned that I'm a control freak?

I ended up writing my book using a technology that I already knew quite well: *HTML*. By writing in HTML, I already had something that I could post on my website, use for the EPUB and use for the MOBI format. All without having to change a thing. (That's right: the same HTML that is used on **SMACSS.com** is used in the EPUB and is used in the MOBI.)

What about PDF? I could open up the HTML in a web browser, choose **Save as PDF** and be done with it but let's face it: the filename and date attached to every single page doesn't exactly scream professional. Web browsers actually do a surprisingly poor job with supporting the **CSS paged media spec**.

I had resorted to copying and pasting the content into Pages and saving as PDF from there. It wasn't elegant but it worked. However, any changes to my HTML source required redoing those changes in Pages, as well.

Then I met my Prince Charming: **Prince XML**. It's pricey but it works incredibly well. It takes HTML and CSS (that very format I've been using for all of my other file formats)

and will generate a PDF via a command line interface. Prince supports CSS paged media including headers, footers, page counts, and alternating page styles.

From one format, HTML, I can now easily publish to PDF, MOBI, and EPUB, and even my website. I use the PDF version to send to the printer along with cover art to be bound and ready to ship around the world. It's amazing how versatile HTML (and CSS) is.

To learn more about writing books with HTML and CSS, I recommend reading [Building Books with CSS3](#) over at [A List Apart](#).

CREATING AN EPUB

Let's take a step back. Prince gets us from HTML to PDF but how do we make an EPUB out of the HTML?

An EPUB file is essentially a ZIP file with a renamed extension. There are some core files that you need to start with:

```
Root
  META-INF
    container.xml
  mimetype
  content.opf
  toc.ncx
```

After that, you can start adding your content to the project. Be sure to update the *toc.ncx* (Table of Contents) and *content.opf* (the ebook manifest) with any changes you make to your project.

You can learn more about the file formats with the **EPUB Format Construction Guide**.

Once all your files are in place, you'll need to create the EPUB file by running two commands (on OS X, at least):

```
zip -X0 your-ebook.epub mimetype
zip -Xur9D your-ebook.epub *
```

The *mimetype* needs to be the first file inside the ZIP file and therefore gets added first. Then, the rest of the files are added.

I've added a function to my *.bash_profile* to make this even easier:

```
function epub()
{
    zip -q0X $@ mimetype; zip -qXr9D $@ *
}
```

Then, within the folder from which I want to create an ebook, I just run `epub your-ebook.epub` from the Terminal command line and the EPUB file should be ready to go.

CREATING THE MOBI

We have our EPUB and we have our PDF. The last step is the MOBI file. For this, I call upon **Calibre**. Calibre can be used as a reader and as a library but I use it exclusively to export my EPUB files to MOBI.

Calibre includes a command line utility to convert from EPUB to MOBI. (To install the command line tools, go to *Preferences > Advanced > Miscellaneous* and click *Install Command Line Tools*.)

```
ebook-convert your-ebook.epub your-ebook.mobi
```

SPREAD THE JOY

Now that you have all of your different file formats, you need to get them into the hands of people who want to (ho-ho-hopefully) buy your book!

There are a number of marketplaces such as Amazon's Kindle Direct Publishing, iBookstore, Google Play, and NOOK Press.

Some publishers, like **PragProg** and **O'Reilly** will also add self-published books to their roster if they feel it's a good fit for their audience.

With any distribution, you'll have to give up a percentage of your sales—from 30% to 70% of each sale, so consider your options wisely.

Of course, you can always open your own online store and reap as much of the revenue as possible, assuming you can get the traffic to your site. Handling your own distribution allows you to create a deeper one-on-one connection with your customers, something that is impossible with other distribution channels since you don't get customer information through other services—even though you are giving them a huge chunk of your sales!

GO FORTH AND PROSPER

There's a lot of thought and time that goes into writing a book and just as much thought and time can go into creating, publishing, and marketing your book once you're done.

In the end, self-publishing can be a very rewarding process and well worth the time that goes into it.

ABOUT THE AUTHOR



Jonathan Snook writes about tips, tricks, and bookmarks on his blog at Snook.ca. He has also written for A List Apart and .net magazine, and has co-authored two books, *The Art and Science of CSS* and *Accelerated DOM Scripting*. He has also authored and received world-wide acclaim for the self-published book, *Scalable and Modular Architecture for CSS* sharing his experience and best practices on CSS architecture.

Photo: Patrick H. Lauke

20. Untangling Web Typography

Nicole Sullivan

24ways.org/201320

When I was a carpenter, I noticed how homeowners often had this deer-in-the-headlights look when the contractor I worked for would ask them to make tons of decisions, seemingly all at once.

Square or subway tile? Glass or ceramic? Traditional or modern trim details? Flat face or picture frame cabinets? Real wood or laminate flooring? Every day the decisions piled up and were usually made in the context of that room, or that part of that room. Rarely did the homeowner have the benefit of taking that particular decision in full view of the larger context of the project. And architectural plans? Sure, they lay out the broad strokes, but there is still so much to decide.

Typography is similar. Designers try to make sites that are easy to use and understand visually. They labour over the details of line height, font size, line length, and font

weights. They consider the relative merits of different typographical scales for applications versus content-driven sites. Frequently, designers consider all of this in the context of one page, feature, or view of an application. They are asked to make a million tiny decisions.

Sometimes designers just bump up the font size until it looks right.

I don't see anything wrong with that. Instincts are important. Designing in context is easier. It's OK to leave the big picture until later. Design a bunch of things, and *then look for the patterns*. You can't always know everything up front. How does the current feature relate to all the other features on the site? For a large site, just like for a substantial remodel, the number of decisions you would need to internalize to make that knowable would be prohibitively large.

WHEN TYPOGRAPHY GOES AWRY

I should be honest. I know very little about typography. I struggle to understand vertical rhythm and the math in Tim Ahrens's talks about the interaction between type design and rendering technology kind of melted my brain. I have an unusual perspective because I'm not the one making the design decisions, but I am the one implementing them and often cleaning up when a project goes off the rails.

I've seen projects with thousands of font-size declarations and headings. One project even had over ten thousand margin declarations. So while I appreciate creative exploration, I'm also eager to establish patterns in typography and make sure we aren't choosing not to choose. Or, choosing *all the things*.

ANALYZING A SITE'S TYPOGRAPHY

Most of my projects start out with an evaluation of the client's existing CSS. I look for duplication in the CSS by using Grep, though functionality is landing soon in **CSS Lint** to do the same thing automatically. The goal is to find the underlying missing abstractions that, once in place, would allow developers to create new functionality without needing to write additional CSS. In addition to that, my team and I would comb through each site (generally, around ten pages is enough to get the big picture), and take screenshots of each of the components we found.

In this way, we could look for subtle visual differences that were unlikely to add value to the user. By correcting these differences, we could help make the design more consistent, and at the same time the code leaner and more performant. Typography is much like a homeowner who chooses to incorporate too many disparate design elements, pairing a mid-century modern sofa with flowered country cottage curtains. Often the typography

of a site ends up collecting an endless array of new typefaces as the site's overall styles evolve. Designers come and go on a project, and eventually no one can remember how the 16px Verdana got into the codebase.

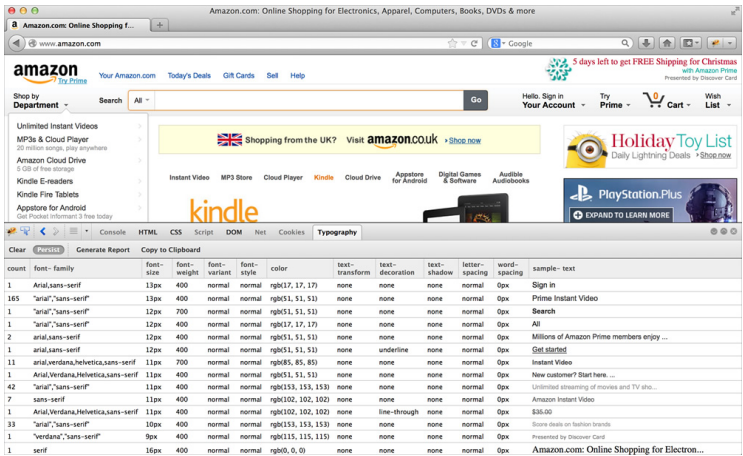
AUTOMATION

We used to do this work by hand. It was incredibly tedious. We'd go through the site, taking screenshots and meticulously documenting the style information we found. We didn't have to do that many times before it became incredibly clear that the task needed to be automated. So we built a little tool called the **Type-o-matic** that could do it for us.

To try it on your site:

1. Download and install the **Firebug** extension to Firefox
2. Download and install the **Type-o-matic** extension to Firebug (I know, I fully intend to port it to Chrome)
3. Now, visit the site you'd like to test
4. Right click and choose *Inspect element with Firebug*
5. Now click on the *Typography* tab
6. Click *Persist*
7. Click *Generate Report*
8. Choose which pages to analyze (we've found that ten is a good number to get the big picture, but you can analyze as many as you'd like — it will even work on just one page!)

9. Now navigate to other pages, and on each subsequent page, click *Generate Report*
10. The table of results can be a bit difficult to interact with, so you can always click *Copy to clipboard*, and copy the results (JSON).



20-1. A screenshot of Type-o-matic in action

WHAT DOES THIS DATA MEAN?

When you've analyzed as many pages or different views as you'd like, you'll start to see some interesting patterns emerge in the data. In the right-hand column, you'll see examples of how each kind of typography we found has been used in a real context on your site. It is organized by color and then by size so you can easily see how you are using typography.

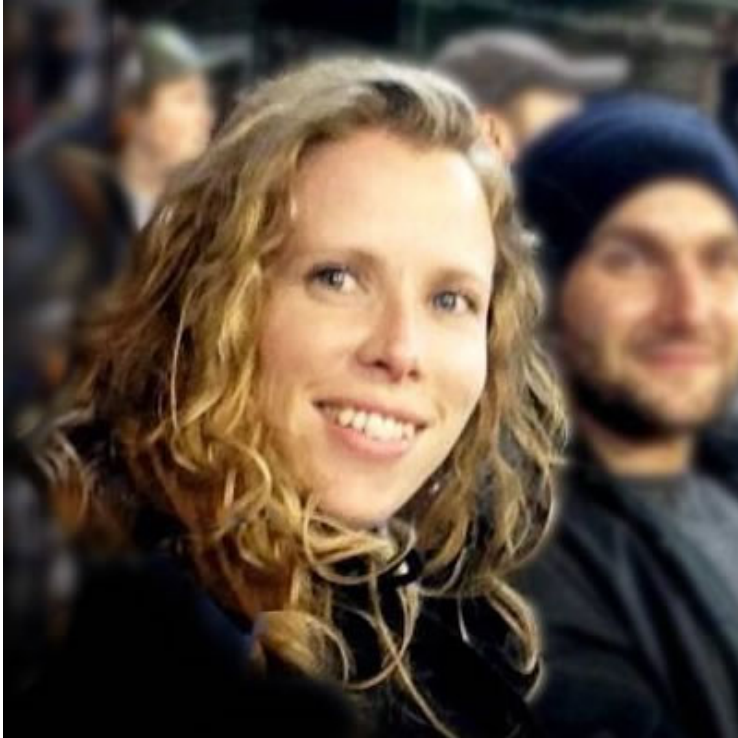
The next thing you'll want to take a look at is in the first column, called "Count". We've counted how many times you've used each combination of typographical styles. This can be incredibly helpful when deciding which styles were intentional, versus one-off color pick errors or experiments that never got removed from the code base. If you've used one color blue 1,400 times, and another just 23, it's pretty obvious which is more in line with broader site-wide styles.

CONSISTENCY BEFORE PERFECTION

It can be really tempting to try to make everything perfect — to try to make every decision final. When you use the data you can collect from this tool, I'd recommend trying to get to consistent before you try to make things perfect. Stop using fifteen different shades of blue type first, and then if you want to change to a new blue, go for it! You'll be able to make design changes much more easily once you've reduced the total number of typographical styles you rely on.

Lower the importance of the decisions you are making. Our sites, like ourselves, are always a work in progress. Or, as a carpenter I used to work with said, "You're not building a fucking piano." We're not building houses. We can choose one typeface today and a different one tomorrow. It is OK to experiment. Be brave.

ABOUT THE AUTHOR



Nicole is a UI performance nerd living and working in San Francisco. She helps companies make their CSS smaller and their UI more manageable. She is also an author, most recently contributing to the [Web Performance Daybook Volume 2](#).

21. Managing a Mind

Christopher Murphy

24ways.org/201321

On 21 May 2013, I woke in a hospital bed feeling exhausted, disorientated and ashamed. The day before, I had tried to kill myself.

It's very hard to write about this and share it. It feels like I'm opening up the deepest recesses of my soul and laying everything bare, but I think it's important we share this as a community. Since starting tentatively to write about **my experience**, I've had many conversations about this: sharing with others; others sharing with me. I've been surprised to discover how many people are suffering similarly, thinking that they're alone. They're not.

Due to an insane schedule of teaching, writing, speaking, designing and just generally trying to keep up, I reached a point where my buffers completely overflowed. I was working so hard on so many things that I was struggling to maintain control. I was living life on fast-forward and my grasp on everything was slowly slipping.

On that day, I reached a low point – the lowest point of my life – and in that moment I could see only one way out. I surrendered. I can't really describe that moment. I'm still grappling with it. All I know is that I couldn't take it any more and I gave up.

I very nearly died.

I'm very fortunate to have survived. I was admitted to hospital, taken there unconscious in an ambulance. On waking, I felt overwhelmed with shame and overcome with remorse, but I was resolved to grasp the situation and address it. The experience has forced me to confront a great deal of issues in my life; it has also encouraged me to seek a deeper understanding of my situation and, in particular, the mechanics of the mind.

THE RELENTLESS PACE OF CHANGE

We work in a fast-paced industry: few others, if any, confront the daily challenges we face. The landscape we work within is characterised by constant flux. It's changing and evolving at a rate we have never experienced before. Few industries reinvent themselves yearly, monthly, weekly... Ours is one of these industries. Technology accelerates at an alarming rate and keeping abreast of this change is challenging, to say the least.

As designers it can be difficult to maintain a knowledge bank that is relevant and fit for purpose. We're on a constant rollercoaster of endless learning, trying to maintain the pace as, daily, new ideas and innovations emerge — in some cases fundamentally changing our medium.

Under the pressure of client work or product design and development, it can be difficult to find the time to focus on learning the new skills we need to remain relevant and functionally competent. The result, all too often, is that the edges of our days have eroded. We no longer work nine to five; instead we work eight to six, and after the working day is over we regroup to spend our evenings learning. It's an unsustainable situation.

FROM THE WORKSHOP TO THE WEB

Added to this pressure to keep up, our work is now undertaken under a global gaze, conducted under an ever-present spotlight. Tools like Dribbble, Twitter and others, while incredibly powerful, have an unfortunate side effect, that of unfolding your ideas in public. This shift, from workshop to web, brings with it additional pressure.

In the past, the early stages of creativity took place within the relative safety of the workshop, an environment where one could take risks and gather feedback from a trusted few. We had space to make and space to break.

No more. Our industry's focus (and society's focus) on sharing, leads us now to play out our decisions in public. This shift has changed us culturally, slowly but surely easing every aspect of our process – and lives – from private to public. This is at once liberating and debilitating.

If you're not careful, an addiction to followers, likes, retweets, page views and other forms of measurement can overwhelm you. When you release your work into the wild and all it's greeted with is silence, it can cripple you.

Reflecting on this, in an insightful article titled **Derailed**, Rogie King asks, "Can social popularity take us off the course of growth and where we were intended to go?" He makes a powerful point, that perhaps we might focus on what really matters, setting aside statistics. He concludes that to grow as practitioners we might be best served by seeking out critique through other avenues, away from the social spotlight.

ON STATUS ANXIETY AND IMPOSTOR SYNDROME

Following my experience I embarked on a period of self-reflection. I wanted to discover what had driven me to take the course of action I had. I wanted to ensure it never happened again. I wanted to understand how the mind works and, in so doing, learn a little more about myself.

I've only begun this journey, but two things I discovered resonated with me: the twin pressures of status anxiety and impostor syndrome.

In his excellent book *Status Anxiety*, the philosopher Alain de Botton explores a growing concern with status anxiety, a worry about how others perceive us and how this shapes our relationship with the world. He states:

We all worry about what others think of us. We all long to succeed and fear failure. We all suffer – to a greater or lesser degree, usually privately and with embarrassment – from status anxiety. [...] This is an almost universal anxiety that rarely gets mentioned directly: an anxiety about what others think of us; about whether we're judged a success or a failure, a winner or a loser.

We see these pressures played out and amplified in the social sphere we all inhabit. We are social animals and we cannot help but react to the landscape we live and work within. Even if your work receives the public praise you so secretly desire, you find yourself questioning this praise.

A psychological phenomenon in which sufferers are unable to internalise their accomplishments, impostor syndrome is far more widespread than you'd imagine. The author Leigh Buchanan describes it as "A fear that one is not as smart or capable as others think." As she puts it,

“People who feel like frauds chalk up their accomplishments to external factors such as luck and timing, or worry they are coasting on charm and personality rather than on talent.”

At the bottom, this was all I could see. I felt overwhelmed by others’ perception of me. Was I a success or a failure? Would I be discovered as the fraud I’d convinced myself that I was? These twin pressures – that I was unconscious of at the time – had lead me to a place of crippling self-doubt, questioning my very existence.

The act of discovery, of investigating how the mind functions, led me to a deeper understanding of myself. Developing an awareness of psychology and learning about conditions like status anxiety and impostor syndrome helped me to understand and recognise how my mind worked, enabling me to manage it more effectively.



MAKE IT COUNT

Reflecting upon my experience, I began to regroup, to focus on what really mattered. I’d taken on too much — as I believe many of us do. I was guilty of wanting to do **all the things**. I started to introduce pauses. Before blindly saying yes to everything, I forced myself to pause and ask: “Is this important?”

Our community offers us huge benefits, but an always-on culture in which we're bombarded daily by opportunity places temptation in our paths. It's easy to get sucked in to a vortex of wanting to be a part of everything. It's important, however, to focus. As **Simon Collison** puts it:

I cull and surrender topics. Then I focus on my strengths, mastering my core skills.

We only have so much time and we can only do so much. It's impossible, indeed futile, to try to do everything. Sometimes we need to step back a little and just enjoy life, enjoy others' achievements, without feeling the need to be actively involved ourselves.

As Mahatma Ghandi put it:

A 'no' uttered from deepest conviction is better and greater than a 'yes' merely uttered to please, or what is worse, to avoid trouble.

Young India, volume 9, 1927

We need to learn to say no a little more often. We need to focus on the work that matters. This, coupled with an understanding of the mind and how it works, can help us achieve a happier balance between work and life.

Don't waste your time. You only have one life. Make it count.

ABOUT THE AUTHOR



Christopher Murphy is a writer, designer and educator based in Belfast. Creative Review described him as, “a William Morris for the digital age,” an epithet he aspires to fulfil, daily. The author of numerous books, collectively covering a multidisciplinary approach towards design, he has written for: **Five Simple Steps**, **8 Faces** and **The Manual**; in addition to publishing the world’s most compact typography journal, **Glyph**.

An internationally respected speaker, he has spoken at conferences worldwide, including: **Build**, **Industry** and, most recently, on the topic of mental health in the technology sector, at **Brooklyn Beta**.

You can follow Christopher's journey via **Twitter**.

22. Bringing Design and Research Closer Together

Emma Boulton

24ways.org/201322

The ‘should designers be able to code’ debate has raged for some time, but I’m interested in another debate: should designers be able to research?

Are you a designer who can do research? Good research and the insights you uncover inspire fresh ways of thinking and get your creative juices flowing. Good research brings clarity to a woolly brief. Audience insight helps sharpen your focus on what’s really important. Experimentation through research and design brings a sense of playfulness and curiosity to your work. Good research helps you do good design.

Being a web designer today is pretty tough, particularly if you’re a freelancer and work on your own. There are so many new ideas, approaches to workflow and trends and tools to keep up with. How do you decide which things to do and which to ignore? A modern web designer needs to

be able to consider the needs of the audience, design appropriate IAs and layouts, choose colour palettes, pick appropriate typefaces and type layouts, wrangle with content, style, code, dabble in SEO, and the list goes on and on. Not only that, but today's web designer also has to keep up with the latest talking points in the industry: responsive design, Agile, accessibility, Sass, Git, lean UX, content first, mobile first, blah blah blah. Any good web designer doesn't need to be persuaded about the merits of including research in their toolkit, but do you really have *time* to include research too?

WHO IS RESPONSIBLE FOR RESEARCH?

Generally, research in the web industry forms part of other disciplines and isn't so much a discipline in its own right. It's very often thought of as part of UX, or activities that make up a process such as IA or content strategy. Research is often undertaken by UX designers, information architects or content strategists and isn't something designers or developers get that involved in. Some people lump all of these activities together and label it **design research** and have **design researchers** to do it. Some companies, such as the one I run with my husband Mark, are lucky enough to have someone with specialist research knowledge (yup, that would be me

folks) who can lead all or most of the research work undertaken by the company. See also Mule Design, GOV.UK, the BBC, Mailchimp, Facebook and Twitter.

What if you're not lucky enough to have your own researcher or team of researchers? Often research is the kind of thing that's nice to have, or it can be cut from scope when doing the budget dance with a client. It often forms part of the discovery phase of a project and sometimes just becomes a tick-box exercise. But research isn't just user testing and it shouldn't just live in a report on Basecamp that no one reads. I would argue that research and experimentation is a way of working or an approach to how you design. Research can be used during the whole design process and *must* be a vital part of a designer's workflow on every project. Even if you work in a small studio, you can still create a culture of audience insight. Even if you work on your own, you can still absorb yourself in as much audience data as you can throughout the project life cycle. Here's how.

RESEARCH IS EVERYONE'S JOB

There is a subtle difference between writing a research report and delivering it to a client, and them actually using it and applying the insights to their thought process. In my experience of working in the audiences team at the BBC,

research was most effective when the role was embedded in the production team and insights were used as part of the editorial process.

In this section I'll talk through some common problems you might encounter in a typical project life cycle and show you ways you can use research to help you. For the sake of this article, let's imagine that we're talking about a particular project here and not ongoing product development. The same principles can of course be applied then, but even if you work in-house rather than on the agency side, you're probably used to working on distinct projects or phases of work.

1. Problem: I want to come up with a new product idea.

Solution: Inspiration through insights.

Before you begin a new project, a good way of quickly absorbing all the existing knowledge that there maybe about a theme, product type or website is to literally surround yourself with it. This is especially relevant for new ideas or product development. Create an *incident room* if you can: fill the walls of your meeting room, the walls near your desk, or even just use a pinboard or online pinboard if space is tight or you're working with a dispersed team. The same process can be used throughout a project's or product life cycle — [read about how MailChimp has applied this idea.](#)

Let's take a new product idea as an example. Say you wanted to develop a responsive tool for web designers but you weren't sure what aspect of responsive design to focus on. First of all, you should pose a hypothesis or problem statement to gather ideas around. For example: "How to speed up a designer's responsive workflow." You would then need to gather insights around this topic. You could run some interviews with freelance designers about how they work responsively. You could shadow a development team for the day to understand their processes. You could observe conversations on Twitter or IRC or wherever your target audience interact to see what people talk about. You could search out industry data and articles currently available.

The next stage is to comb through this data and extract insights from it. You can use good old Post-it notes and a sharpie: capture one insight or thought per Post-it. If one insight leads into another, use two Post-its. The objective is volume. Try to ensure clarity in each Post-it so you don't have to go back and reference material again (maybe you could use a key if you think it'll get confusing).



After this, stick them all up and synthesise the same way you would for any kind of cluster or affinity sort. Organise into broad themes. These themes then become springboards for further exploration and idea generation. You might see a gap or opportunity in one particular area, both from a workflow perspective but also from a business perspective. Bingo. Your insights then become the fuel for ideas generation.

This method doesn't just have to be used for new products — it works particularly well in a discovery phase for new projects or for new features in an existing product. We're doing something similar for our own responsive tool, **Gridset** at the moment.

Resources:

- *Sticky Wisdom* by Dave Allan, Matt Kingdon, Kris Murrin, Daz Rudkin
- *The Science of Serendipity* by Matt Kingdon
- *The Art of Innovation* by Tom Kelley

2. Problem: You're starting a new project and need to know the basics before you get headlong into designing or building.

Solution: Quantitative survey.

Common questions might be:


- Who are the users?
- How many are there?
- What are they like?
- Why do they use the site?
- What do they need from the site?
- What are their goals?

Print out and stick up what you already know and have in your project space or 'incident room': any reports you have found or been given, analytics graphs, personas, pen portraits, as well as screengrabs of the current website, product or branding. Spend time looking through it all and identify the gaps.

If you have very little existing audience data, a quick and easy way to get some baseline information is to run a quick user survey on a current website. You can establish

basic demographic information, appreciation and views of the website as it stands, as well as delve a little deeper into needs and wants. This is also vital if you want some kind of trackable measures to go back to once you have designed and built your shiny new website for your client – read more in my article for 24 ways last year.)

Info.CERN user survey

 <http://info.cern.ch>

Exit this survey

Thanks for agreeing to take part in our short user survey. It's very short and should take around 5 minutes of your time to complete.

This survey is purely for research purposes and your responses won't be used for any other reason. We promise to keep your data anonymous and won't publish any of your comments elsewhere on the web, without your permission.

1. Did you find what you needed on the [info.cern](http://info.cern.ch) website today?

☐ Yes

☐ No

☐ Don't Know

2. Why have you come to the [info.cern](http://info.cern.ch) website today?

☐ To find out about how the world wide web began

☐ To read the original proposal written by Tim Berners Lee

☐ To find out more about the original web browser

☐ To find out more about the first universal line-mode browser

☐ To find out more about what happens at CERN

☐ To find out more about the Higgs Boson

☐ To find out more about the recent announcements

☐ For press information

☐ For contact details for CERN

Other (please specify)

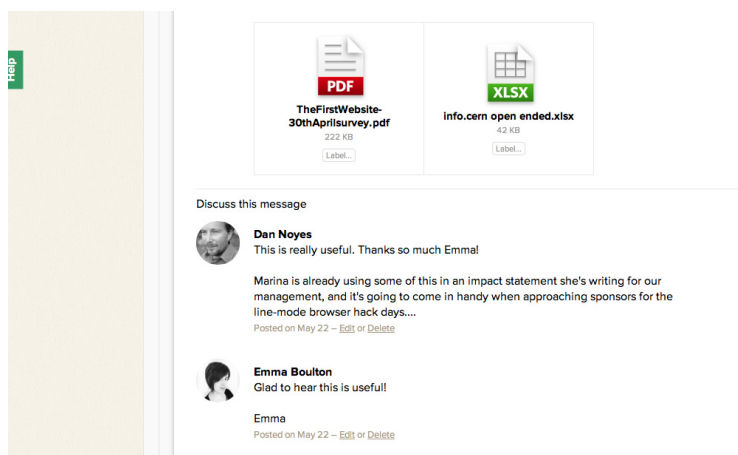
3. Was there a task that you were not able to complete on the [info.cern](http://info.cern.ch) website today?

Next

We use surveys a lot at Mark Boulton Design for our client work. Here's a screen grab of one we ran in March on <http://info.cern.ch> before we redesigned the site and did the work on the First Website Project. We repeated the survey after the new website went live and were able

Bringing Design and Research Closer Together

to compare the results. Both surveys were a great source of insight to the project team as well as for the project stakeholders who needed to pitch the idea of the hack days and fundraise for them.



Once you've run your survey, you should always write up a short summary for yourself and your client to refer to. If you're not a trained researcher, you should try to read up on analysis techniques or data visualisation. It can be easy to misinterpret data and make it bend to the story you are trying to tell. You should be looking for the story in the data and present it without bias.

If you're using the 'incident room' method I mentioned earlier on, you can also extract the insights onto post it notes and add them to your growing body of knowledge.

Resources:

- Using Questionnaires for Design Research by Emma Boulton
- Data-driven Design with an Annual Survey by Aarron Walter
- *Research Methods for Product Design* by Alex Milton and Paul Rodgers
- *A Practical Guide to Designing with Data* by Brian Suda

3. Problem: You have a prototype of a new design and you need some feedback from real users.

Solution: User interviews and task based testing.

Interviewing is a staple research method that every designer should master as it can be used throughout a project life cycle. Erika Hall recently wrote a great article on the basics for A List Apart. From stakeholder interviews in a discovery phase, to initial user research, right through to task based testing and iteration, interviews can be enormously helpful. They are very time-consuming, however, and although speaking to someone is better than speaking to no one, it's always better to plan to do a few interviews at once, rather than one or two. I generally find that patterns only start to emerge after I've spoken to 4 or 5 people. Interviews are another thing we do a lot of at Mark Boulton Design. Most of the interviews we do are remote due to the location of our clients and their users.



Rigour is an important consideration in all research activities and especially if you're a non-researcher. Interviews particularly can be easily skewed by an inexperienced facilitator, which is why pairing can be a good approach. Building rapport, questioning, time keeping, note taking and thinking on your feet can be difficult to do all at once, so having a colleague take notes while you concentrate on leading the conversation can work really well. It's important for the note taker to sit in on more than one interview so that they get a more rounded view of the feedback. The same person should also be involved in the analysis of the data.



Interviews can be analysed and written up in a report or summary as with other types of research. I often use the same kind of collaborative process detailed earlier for deciding on themes, particularly if multiple members of the team have been involved in interviewing.

Interviews are particularly useful for our incident room and can provide much colour and insight to an exploratory process. I often find verbatim quotes to be the most insightful type of data. You might find that an inexperienced researcher (or designer who is used to solving problems) will jump to interpretation too soon and forget to just listen to what the interviewee is saying. Capturing the exact form of words a person uses can help get away from this.

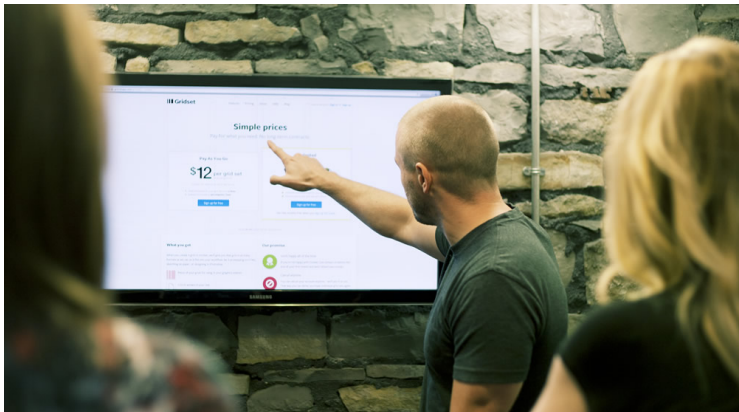
Resources:

- Interviewing Humans by Erika Hall
- *A Pocket Guide to Interviewing for Research* by Andrew Travers
- *Interviewing Users* by Steve Portigal

4. Problem: How successful have I been with this new design?

Solution: Key performance indicators

Once your new design has been realised, it's important to evaluate it. What works, what doesn't work so well? As well as a straightforward design crit, don't forget to introduce audience insights into a review meeting or project wash up.



Work out what your KPIs — your key performance indicators — will be beforehand and then you can start to track them over time. For example, number of visits, appreciation of the site, willingness to recommend the site to a friend, number of sales, and number of conversions are all sensible measures to track. Interviews can again be helpful but cold, hard numbers are often better here. Read [Corey Vilhauer's take on this on A List Apart](#).

Consistency is key here. If you have looked at your analytics and done a survey beforehand, you will have a baseline to start from. Don't keep changing your measures and questions, or your data will not be comparable. Pick a few key questions or a set of measures, create a survey and then run it once a month, once a quarter, every six months or annually. You'll start to see changes over time as the design beds in. You may see seasonal trends and spot patterns in the data related to other activities like marketing, promotion and so on. Keeping a record of all of this will increase your understanding of your audience. We've created a satisfaction survey for Gridset with a number of measures that we track on an ongoing basis. [MailChimp](#) has also created an annual survey with the aim of tracking their audience measures over time

Resources:

- *Search Analytics* by Louis Rosenfeld
- *A Primer on A/B Testing* by Lara Swanson
- *Lean UX* by Jeff Gothelf

ANYONE CAN DO RESEARCH

Research can be brought into the project life cycle at any stage. And of course, anyone can do research — you don't need to be a researcher. Some of the main skills most designers possess are also key research skills: inquisitive nature, problem solving, playfulness, empathy, and so on.

We have a small team at Mark Boulton Design. Most of the team are designers and the rest of us focus on supporting the team and clients both in terms of billable work (research, content strategy, project management) as well as the non-billable things like finance and studio management.

Despite my best intentions, in the past I've undertaken research for clients in isolation — first being briefed by the design lead, carrying out the research and then delivering the findings back, trusting the design team to take the findings on board. This was often due to time and availability of resources.

We've been trying hard to join up our processes and collaborate even more across the team. Undertaking heuristic or design reviews collaboratively; taking part in frequent critiques of our work and the work of others

together; pairing a researcher and a designer to run interviews; workshopping results from interviews to come up with recommendations; working closely together on questionnaire design; shadowing each other on tasks that don't fall within our core skills. A little thing like moving our desks around has also helped us have more conversations that we can all be a part of.



I've come to the conclusion that my role as the research director at Mark Boulton Design is actually a facilitator of research. As well as carrying out research, I am responsible for ensuring that research happens consistently across the team. I am responsible for empowering and training our designers so they feel confident in carrying out their own user, audience or design research for clients. So they know what to look for, when to listen, when to probe and when to take note of

something. So they know how to look for themes, how to synthesise insights from research and how to apply them to their work.

BETTER RESEARCH LEADS TO BETTER DESIGN

So, are you a designer who can do research? Are you a researcher who can design? The best designers are a lucky combination of researcher and designer. If you're not one of those, look at ways of enhancing the skills you lack. Because there's no doubt in my mind, that becoming a better researcher will make you a better designer.

General resources:

- *Seeing the Elephant* by Louis Rosenfeld
- *Connected UX* by Aaron Walter
- *Beyond Usability Testing* by Devan Goldstein
- *Just Enough Research* by Erika Hall
- *The User Experience Team of One* by Leah Buley
- *Undercover User Experience Design* by Cennydd Bowles and James Box
- *A Pocket Guide to Psychology for Designers* by Joe Leech
- *A Pocket Guide to International User Research* by Chui Chui Tan
- *Remote Research* by Nate Bolt and Tony Tulathimutte
- *A Pocket Guide to Experiments for Designers* by Colin McFarland

ABOUT THE AUTHOR



Emma has been helping clients understand their audiences for the better part of the last 14 years. She cut her research teeth in the brave new world of online advertising, before moving to the Audiences team at the BBC. She's now the Research Director at **Mark Boulton Design** and works as part of a small but exceptional team creating great web experiences for clients such as CERN, Al Jazeera and Global Witness. Emma is also the Editor in Chief of indie publisher, **Five Simple Steps**.

23. The Command Position Principle

Andrew Clarke

24ways.org/201323

Living where I do, in a small village in rural North Wales, getting anywhere means driving along narrow country roads. Most of these are just about passable when two cars meet.

If you're driving too close to the centre of the road, when two drivers meet you stop, glare at each other and no one goes anywhere. Drive too close to your nearside and in summer you'll probably scratch your paintwork on the hedgerows, or in winter you'll sink your wheels into mud.

Driving these lanes requires a balance between caring for your own vehicle and consideration for someone else's, but all too often, I've seen drivers pushed towards the hedgerows and mud when someone who's inconsiderate drives too wide because they don't want to risk scratching their own paintwork or getting their wheels dirty.

If you learn to ride a motorcycle, you'll be taught about the command position:

Approximate central position, or any position from which the rider can exert control over invitation space either side.

The command position helps motorcyclists stay safe, because when they ride in the centre of their lane it prevents other people, usually car drivers, from driving alongside, either forcing them into the curb or potentially dangerously close to oncoming traffic.

Taking the command position isn't about motorcyclists being aggressive, it's about them being confident. It's them knowing their rightful place on the road and communicating that through how they ride.

I've recently been trying to take that command position when driving my car on our lanes. When I see someone coming in the opposite direction, instead of instinctively moving closer to my nearside — and in so doing subconsciously invite them into my space on the road — I hold both my nerve and a central position in my lane. Since I done this I've noticed that other drivers more often than not stay in their lane or pull closer to their nearside so we occupy equal space on the road. Although we both still need to watch our wing mirrors, neither of us gets our paint scratched or our wheels muddy.

We can apply this principle to business too, in particular to negotiations and the way we sell. Here's how we might do that.

COMMANDING NEGOTIATIONS

When a customer's been sold to well — more on that in just a moment — and they've made the decision to buy, the thing that usually stands in the way of us doing business is a negotiation over price. Some people treat negotiations as the equivalent of driving wide. They act offensively, because their aim is to force the other person into getting less, usually in return for giving more.

In encounters like this, it's easy for us to act defensively. We might lack confidence in the price we ask for, or the value of the product or service we offer. We might compromise too early because of that. When that happens, there's a pretty good chance that we'll drive away with less than we deserve unless we use the command position principle to help us.

Before we start any negotiation it's important to know that both sides ultimately want to reach an agreement. This isn't always obvious. If one side isn't already committed, at least in principle, then it's not a negotiation at that point, it's something else.

For example, a prospective customer may be looking to learn our lowest price so that they can compare it to our competitors. When that's the case, we've probably failed to qualify that prospect properly as, after all, who wants to be chosen simply because they're the cheapest? In this situation, negotiating is a waste of time since we don't yet know that it will result in us making a deal. We should enter into a negotiation only when we know where we stand. So ask confidently: "Are you looking to [make a decision]?"

When that's been confirmed, it's down to everyone to compromise until a deal's been reached. That's because good negotiations aren't about one side beating the other, they're about achieving a good deal for both. Using the command position principle helps us to maintain control over our negotiating space and affords us the opportunity to give ground only if we need to and only when we're ready. It can also ensure that the person we're negotiating with gives up some of their space.

COMMANDING SALES

It's not always necessary to negotiate when we're doing a business deal, but we should always be prepared to sell. One of the most important parts of our sales process should be controlling when and how we tell someone our price.

Unless it's impossible to avoid, don't work out a price for someone on the spot. When we do that we lose control over the time and place for presenting our price alongside the value factors that will contribute to the prospective customer accepting that price. For the same reason, never give a ballpark or, worse, a guesstimate figure. If the question of price comes up before we're fully prepared, we should say politely that we need more time to work out a meaningful cost.

When we are ready, we shouldn't email a price for our prospective customer to read unaccompanied. Instead, create an opportunity to talk a prospect through our figures, demonstrate how we arrived at them and, most importantly, explain the value of what we're selling to their business. Agree a time and place to do this and, if possible, do it all face-to-face.

We shouldn't hesitate when we give someone a price. When we sound even the slightest bit unsure or apologetic, we give the impression that we'll be flexible in our position before negotiations have even begun.

Think about the command position principle, know the price and present it confidently. That way we send a clear signal that we know our business and how we deal with people. The command position principle isn't about being cocky, it's about showing other people respect, asking for it in return and showing it to ourselves.



Earlier, I mentioned selling well, because we sometimes hear people say that they dislike being sold to. In my experience, it's not that people dislike the sales process, it's that we dislike it done badly.

Taking part in a good sales process, either by selling or being sold to, can be a pleasurable experience. Try to be confident — after all, we understand how our skills will benefit a customer better than anyone else. Our confidence will inspire confidence in others.

Self-confidence isn't the same as arrogance, just as the command position isn't the same as riding without consideration for others. The command position principle preserves others' space as well as our own. By the same token, we should be considerate of others' time and not waste it and our own by attempting to force them into buying something that's inappropriate.

To prevent this from happening, evaluate them well to ensure that they're the right customer for us. If they're not, let them go on their way. They'll thank us for it and may well become customers the next time we meet.

The business of closing a deal can be made an enjoyable experience for everyone if we take control by guiding someone through the sales process by asking the right

questions to uncover their concerns, then allaying them by being knowledgeable and confident. This is riding in the command position.

Just like demonstrating we know our rightful position on the road, knowing our rightful place in a business relationship and communicating that through how we deal with people will help everyone achieve an equitable balance. When that happens in business, as well as on the road, no one gets their paintwork scratched or their wheels muddy.

ABOUT THE AUTHOR



Andrew Clarke runs **Stuff and Nonsense**, a tiny web design company where they make fashionably flexible websites. Andrew's the author of **Transcending CSS** and **Hardboiled Web Design** and hosts the popular weekly podcast **Unfinished Business** where he discusses the business side of web, design and creative industries with his guests. He tweets as **@malarkey**.

24. Run Ragged

Mark Boulton

24ways.org/201324

You care about typography, right? Do you care about words and how they look, read, and are understood? If you pick up a book or magazine, you notice the moment something is out of place: an orphan, rivers within paragraphs of justified prose, or caps masquerading as small caps. So why, I ask you, is your stance any different on the web?

We're told time and time again that as a person who makes websites we have to get comfortable with our lack of control. On the web, this is a feature, not a bug. But that doesn't mean we have to lower our standards, or not strive for the same amount of typographic craft of our print-based cousins. We shouldn't leave good typesetting at the door because we can't control the line length.

When I typeset books, I'd spend hours manipulating the text to create a pleasurable flow from line to line. A key aspect of this is manicuring the right rag — the vertical line of words on ranged-left text. Maximising the space

available, but ensuring there are no line breaks or orphaned words that disrupt the flow of reading. Setting a right rag relies on a bunch of guidelines — or as I was first taught to call them, violations!

VIOLATION 1. NEVER BREAK A LINE IMMEDIATELY FOLLOWING A PREPOSITION

Prepositions are important, frequently used words in English. They link nouns, pronouns and other words together in a sentence. And links should not be broken if you can help it. Ending a line on a preposition breaks the join from one word to another and forces the reader to work harder joining two words over two lines.

For example:

■ The container is for the butter

The preposition here is *for* and shows the relationship between the butter and the container. If this were typeset on a line and the line break was after the word *for*, then the reader would have to carry that through to the next line. The sentence would not flow.

There are lots of prepositions in English – about 150 – but only 70 or so in use.

VIOLATION 2. NEVER BREAK A LINE IMMEDIATELY FOLLOWING A DASH

A dash — either an em-dash or en-dash — can be used as a pause in the reading, or as used here, a point at which you introduce something that is not within the flow of the sentence. Like an aside. Ending with a pause on the end of the line would have the same effect as ending on a preposition. It disrupts the flow of reading.

VIOLATION 3. NO SMALL WORDS AT THE END OF A LINE

Don't end a line with small words. Most of these will actually be covered by violation №1. But there will be exceptions. My general rule of thumb here is not to leave words of two or three letters at the end of a line.

VIOLATION 4. HYPHENATION

In print, hyphens are used at the end of lines to join words broken over a line break. Mostly, this is used in justified body text, and no doubt you will be used to seeing it in newspapers or novels. A good rule of thumb is to not allow more than two consecutive lines to end with a hyphen.

On the web, of course, we can use the CSS `hyphens` property. It's reasonably supported with the exception of Chrome. Of course, it works best when combined with justified text to retain the neat right margin.

VIOLATION 5. DON'T BREAK EMPHASISED PHRASES OF THREE OR FEWER WORDS

If you have a few words emphasised, for example:

■ He calls this *problem definition escalation*

...then try not to break the line among them. It's important the reader reads through all the words as a group.

HOW DO WE DO ALL OF THAT ON THE WEB?

All of those guidelines are relatively easy to implement in print. But what about the web? Where content is poured into a template from a CMS? Well, there are things we can do. Meet your new friend, the non-breaking space, or as you may know them: .

The guidelines above are all based on one decision for the typesetter: when should the line break?

We can simply run through a body of text and add the based on these sets of questions:

1. Are there any prepositions in the text? If so, add a after them.

2. Are there any dashes? If so, add a space after them.
3. Are there any words of fewer than three characters that you haven't already added spaces to? If so, add a space after them.
4. Are there any emphasised groups of words either two or three words long? If so, add a space in between them.

For a short piece of text, this isn't a big problem. But for longer bodies of text, this is a bit arduous. Also, as I said, lots of websites use a CMS and just dump the text into a template. What then? We can't expect our content creators to manually manicure a right rag based on these guidelines. In this instance, we really need things to be automatic.

There isn't any reason why we can't just pass the question of when to break the line straight to the browser by way of a script which compares the text against a set of rules. In plain English, this script could be to scan the text for:

1. Prepositions. If found, add a space after them.
2. Dashes. If found, add a space after them.
3. Words fewer than three characters long that aren't prepositions. If found, add a space after them.
4. Emphasised phrases of up to three words in length. If found, add a space between all of the words.

And there we have it.

A note on fluidity

An important consideration of this script is that it doesn't scan the text to see what is at the end of a line. It just looks for prepositions, dashes, words fewer than three characters long, and emphasised words within paragraphs and applies the accordingly **regardless** of where the thing lives. This is because in a fluid layout a word might appear in the beginning, middle or the end of a line depending on the width of the browser. And we want it to behave in the right way when it does find itself at the end.

SEE IT IN ACTION!

My friend and colleague, **Nathan Ford**, has written a small JavaScript called Ragadjust that does all of this automatically. The script loops through a webpage, compares the text against the conditions, and then inserts in the places that violate the conditions above.

You can get the script from [GitHub](#) and see it in action on my own website.

SOME CAVEATS

As my friend **Jon Tan** says, "There are no rules in typography, just good or bad decisions", and typesetting the right rag is no different.

- The guidelines for the violations above are useful for justified text, too. But we need to be careful here. Too stringent adherence to these violations could lead to ugly gaps in our words — called rivers — as the browser forces justification.
- The violation regarding short words at the end of sentences is useful for longer line lengths, or measures, of text. When the measure gets shorter, maybe five or six words, then we need to be more forgiving as to what wraps to the next line and what doesn't. In fact, **you can see this happening on my site** where I've not included a check on the size of the browser window (purposefully, for this demo, of course. Ahem).
- This article is about applying these guidelines to English. Some of them will, no doubt, cross over to other languages quite well. But for those languages, like German for instance, where longer words tend to be in more frequent use, then some of the rules may result in a poor right rag.

MARGINAL GAINS

In 2007, I spoke with Richard Rutter at SXSW on web typography. In that talk, Richard and I made a point that good typographic design — on the web, in print; anywhere, in fact — relies on small, measurable improvements across an entire body of work. From heading hierarchy to your grid system, every little bit

helps. In and of themselves, these little things don't really mean that much. You may well have read this article, shrugged your shoulders and thought, "Huh. So what?" But these little things, when added up, make a difference. A difference between good typographic design and great typographic design.



APPENDIX

Preposition whitelist

aboard
about
above
across
after
against
along
amid
among
anti
around
as
at
before
behind
below

beneath
beside
besides
between
beyond
but
by
concerning
considering
despite
down
during
except
excepting
excluding
following
for
from
in
inside
into
like
minus
near
of
off
on
onto

opposite
outside
over
past
per
plus
regarding
round
save
since
than
through
to
toward
towards
under
underneath
unlike
until
up
upon
versus
via
with
within
without

ABOUT THE AUTHOR



Mark Boulton is a graphic designer from near Cardiff in the UK. He used to work as a Senior Designer for the BBC, before he took leave of his senses and formed his own design consultancy, **Mark Boulton Design**. He studied typography, enjoys watching a good boxing match, and is partial to a really good cuppa.