Migrating from Blogger to GitHub Pages

Posted by on 14 August 2019 • permalink

So, you might have noticed I've done a bit of decorating... welcome to the 2019 incarnation of dylanbeattie.net. If you're interested in how I migrated ten years' worth of blog posts onto a Jekyll site hosted on GitHub Pages, read on. If you're not, there's funny videos over on /music that you might enjoy.

Still here? Cool! OK, so the very first version of dylanbeattie.net was a site I built in PHP back in 1999 or so, which I hosted on my own physical server – way back in the days when both of those things were still considered a pretty neat idea. About ten years I ago I started using Blogger, and before long I scrapped the old PHP site, pointed the main domain at my Blogger site, and just sort of got on with it.

Blogger's been a pretty good platform, but these days I find I actually need a website to do a lot more than just host blog posts and the occasional 'about me' page, so a few months back I started looking around for alternatives. I'd considered setting up something like Umbraco, or a custom Wordpress site, but even those felt a bit like over-engineering for what I actually needed to do. My requirements basically boiled down to:

- A really easy way to post articles – text and links with some simple formatting and the occasional image
- The option to use rich HTML pages and custom layouts
- Preserving the URIs of all my existing posts and pages – remember, cool URIs don't change. Besides, I've got a decades' worth of Google-fu invested in those pages. It'd suck if all those links and bookmarks stopped working.
- Something that looked good, and a responsive layout that worked well across devices.

Hi there. I'm Dylan.

I do interesting things with computers, code, comedy, music and video, then I travel all over the world and tell people about it. I run Ursatile, a fully online training company for remote workers and software professionals. I'm a keynote speaker, I'm a Microsoft MVP, I created Rockstar, an esoteric programming language which started as a joke and ended up in Classic Rock magazine, and I own the best web address in the history of the internet.

The Optimism Corner

There's also a couple of things I decided I definitely didn't want:

- Comments. It's been years since I saw a decent discussion in the comments thread of a blog post. We have Twitter and Reddit for that now.
- Bootstrap. I'm sure it's lovely. I just don't like it.

Back in January, I was chatting with Todd Gardner about how he created the PubConf website, and he suggested I take a look at GitHub Pages and a thing called Jekyll. I spent an evening playing around with it, and was basically hooked.

Jekyll and GitHub Pages

> ***Note:*** *the code for this site is available on GitHub at github.com/dylanbeattie/dylanbeattie.net*

So here's how it works. You write your site in HTML or Markdown, and create datasets in YAML. Jekyll compiles them into a static HTML site. It supports a templating language called Liquid, which lets you create templates, conditionals, navigation – you can actually do some pretty sophisticated stuff with it, but everything happens at build time. Which is actually very cool.

The really great part is that GitHub Pages has built-in support for Jekyll – so you build your site locally, using `bundle exec jekyll serve` to view it, then you just push the repo to GitHub and it'll build and deploy it for you. It's a beautifully simple workflow, but one that affords a surprising amount of flexibility once you get the hang of it.

Here's some fun little things I learned along the way, and a couple of gotchas to watch out for if this inspires you to try something similar.

Design and layout

The design for my new site is the Arcana template by HTML5 UP, which is not only beautiful, responsive and really well put together, but is also released – like all HTML5 UP's templates – under a Creative Commons Attribution 3.0 License:

> *"...which means you can use them for personal stuff, use them for commercial stuff, change them however you like ... all for free, yo. In exchange, just give HTML5 UP credit for the design and tell your friends about it :)".*

Here's where I'm hoping to be over the next few months:

digit 2022
22 April 2022
Tartu, Estonia

NDC Porto
25-29 April 2022
Porto, Portuga

JFokus
3-4 May 2022
Stockho Sweden

NDC Londor
9-13 May 2022
London, UK

CodeCa Festiva
19 May 2022
Buchare Romani

NDC Copenh
30 May - 2 June 2022
Copenha Denmar

NDC Oslo
26-30 Septemb 2022
Oslo, Norway

All subject to change, cancellation, virtualisation, postponement, rescheduling and other

side-effects of Living In Interesting Times.

Which is lovely. You see, I'm quite happy doing my own design work – and that's the problem. I enjoy it. I get sidetracked. I spend hours – days – tweaking layouts and playing around with things, when I should be writing content and fixing bugs... and I end up with something that's OK, but nothing like as good as some of the amazing templates and layouts that are available online, for free. So credit (and thanks!) to HTML5 UP for the design. I've made a couple of tweaks and added a few extra bits, but the layouts, grid, responsive design, navigation, typography – that's all them. Oh, and it's all built on SASS – which is also supported by GitHub Pages. Did I mention how much I love this platform?

Migrating old blog posts

This turned out to be a lot easier than I expected. There's a Ruby gem designed to do exactly this – check out the Jekyll documentation and this blog post from Kris Rice which goes into a bit more detail about it.

Once I'd migrated all the posts from my old Blogger site, I wanted to make sure all the old page URLs would still work. I didn't want to mess too much with Jekyll's conventions about structuring posts, though – Jekyll uses a `YYYY-MM-DD-title` format for URLS compared to Blogger's `YYYY-MM-title` format, and so I simultaneously wanted to adopt the Jekyll convention for new posts, to make things easier, but also to preserve the addresses of all my old posts so bookmarks and links still work.

Turns out the `jekyll-migrate` gem adds some lines to the top of each migrated HTML file – what Jekyll refers to as 'front matter' – that can help:

```
---
layout: post
title: How to *really* break the internet.
date: '2016-03-23T12:08:00.000Z'
author: Dylan Beattie
tags:
modified_time: '2016-03-23T14:06:37.095Z'
blogger_id: tag:blogger.com,1999:blog-7295454224203070190.post-89961212079801
blogger_orig_url: http://www.dylanbeattie.net/2016/03/how-to-really-break-int
---
```

There's also a Jekyll plugin called `redirect-from`, which is part of the `github-pages` bundle (and, yes, it works a little like the infamous COMEFROM flow control statement beloved of sadistic esolang designers). So all I had to do was trawl through every file in the `_posts` folder, find that line with the `blogger_orig_url` in it, parse the path part out of the URL, and add a line underneath like this:

```
blogger_orig_url: http://www.dylanbeattie.net/2016/03/how-to-really-break-int
redirect_from: "/2016/03/how-to-really-break-internet.html"
```

(Sorry, awk fans – I actually did this with a global search & replace in VS Code…)

That's it – Jekyll now hosts all those pages at their new `/2016/03/23/how-to-really-break-internet.html` addresses, and requesting the original URL returns this:

```
<!DOCTYPE html>
<html lang="en-US">
<meta charset="utf-8">
<title>Redirecting&hellip;</title>
<link rel="canonical" href="/2016/03/23/how-to-really-break-internet.html">
<script>
  location="/2016/03/23/how-to-really-break-internet.html"
</script>
<meta http-equiv="refresh" content="0; url=/2016/03/23/how-to-really-break-in
<meta name="robots" content="noindex">
<h1>Redirecting&hellip;</h1>
<a href="/2016/03/23/how-to-really-break-internet.html">Click here...</a>
</html>
```

It's not *quite* perfect – an HTTP `301 Moved Permanently` would strictly speaking be a better way of handling this – but hey, perfect is the enemy of good enough. It works. Ship it.

Events schedule and flags

You see that little sidebar there, with all the events I'm speaking at and the flags in it of the countries I'm gonna be visiting? The flag images are from GoSquared, and available under an MIT license. To display them in the schedule, I've used a SASS list to generate CSS rules for each flag in the set:

```
/* _flags.scss: generate .flag-xx CSS classes for elements with flag backgrou

$countries: _abkhazia, _basque-country, _british-antarctic-territory, _common
  _england, _gosquared, _kosovo, _mars, _nagorno-karabakh, _nato, _northern-c
  _olympics, _red-cross, _scotland, _somaliland, _south-ossetia, _united-nati
  AD, AE, AF, AG, AI, AL, AM, AN, AO, AQ, AR, AS, AT, AU, AW, AX, AZ, BA, BB,
  BG, BH, BI, BJ, BL, BM, BN, BO, BR, BS, BT, BW, BY, BZ, CA, CC, CD, CF, CG,
  CL, CM, CN, CO, CR, CU, CV, CW, CX, CY, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE,
  ES, ET, EU, FI, FJ, FK, FM, FO, FR, GA, GB, GD, GE, GG, GH, GI, GL, GM, GN,
  GT, GU, GW, GY, HK, HN, HR, HT, HU, IC, ID, IE, IL, IM, IN, IQ, IR, IS, IT,
  JP, KE, KG, KH, KI, KM, KN, KP, KR, KW, KY, KZ, LA, LB, LC, LI, LK, LR, LS,
  LY, MA, MC, MD, ME, MF, MG, MH, MK, ML, MM, MN, MO, MP, MQ, MR, MS, MT, MU,
  MY, MZ, NA, NC, NE, NF, NG, NI, NL, NO, NP, NR, NU, NZ, OM, PA, PE, PF, PG,
  PN, PR, PS, PT, PW, PY, QA, RO, RS, RU, RW, SA, SB, SC, SD, SE, SG, SH, SI,
  SN, SO, SR, SS, ST, SV, SY, SZ, TC, TD, TF, TG, TH, TJ, TK, TL, TM, TN, TO,
  TW, TZ, UA, UG, US, UY, UZ, VA, VC, VE, VG, VI, VN, VU, WF, WS, YE, YT, ZA,
```

```
@each $country in $countries {
  .flag-#{to-lower-case($country)} {
    background-image: url(images/flags/flat/64/#{$country}.png);
  }
}
```

The actual schedule is a YAML file, and Jekyll generates the markup with the correct CSS classes based on the country codes from `schedule.yml`. It's easy to add new dates. The only thing it won't do is automatically archive old ones- 'cos hey, static content, remember? – but I reckon I can live with that for now.

(And yes, you can *absolutely* invite me to speak at your event by editing schedule.yml and sending me a PR. That would be really cool. :))

Speaker bios

I've got a bunch of different speaker bios over at my about me page, and I really wanted a way to pick one and copy it to the clipboard as either HTML, Markdown or plain text – different events use different formats when you're submitting to their CFP or filling out speaker details, and it's a little thing that would make life easier.

The bios themselves are stored as multiline Markdown snippets in /_data/speaker_bios.yml. Instead of allowing Jekyll to just render Markdown > HTML automatically, I've got this little loop in the code for the **about me** page:

```
{% for bio in site.data.speaker_bios %}
<article>
  <hr />
  <h3>
    <span class="clipboard-links">
      <a href="#" data-src-id="{{bio.id}}-html">copy html</a>
      <a href="#" data-src-id="{{bio.id}}-markdown">copy markdown</a>
      <a href="#" data-src-id="{{bio.id}}-text">copy text</a>
    </span>
    {{ bio.word_count }} Word Bio ({{ bio.char_count }} characters)
  </h3>
  {{ bio.content | markdownify }}
        <input type="hidden" id="{{bio.id}}-markdown"
          value="{{ bio.content }}" />
        <input type="hidden" id="{{bio.id}}-html"
          value="{{ bio.content | markdownify | escape_once }}" />
        <input type="hidden" id="{{bio.id}}-text"
          value="{{ bio.content | markdownify | strip_html }}" />
</article>
{% endfor %}
```

so each snippet is rendered to the page as HTML (via the `markdownify` filter), and also captured in three hidden variables – one markdown, one HTML, one plain text. Finally, there's some JavaScript attached to the

'copy xxx' links that'll copy the hidden input value into an invisible `textarea` and copy it.

Syntax Highlighting

One of the great things about writing posts and pages in Markdown is that it's so easy to embed code samples - just wrap them in three backticks either side, something known as a *fenced code block*. Jekyll has a code formatting plugin called Rouge, that's included with the `github-pages` bundle, which allows you to specify a language for your code snippets and it'll highlight them for you:

```
``` html
<p>This HTML snippet will get <a href="http://rouge.jneen.net/">highlighted</a
```
```

To get this to work, I had to enable highlighting in `_config.yml`

```
markdown: kramdown
# enable rouge syntax highlighting
highlighter: rouge
```

I also had to add a stylesheet to the site with the various coloring rules - all Rouge does is wrap all the code keywords, etc. in `<span>` tags with CSS classes on them, so I found this syntax.css file on GitHub, dropped it into my site's CSS, and it worked.

Gotchas

Pretty much everything I tried to do with Jekyll and GitHub Pages worked as documented, but I did hit two weird gotchas that caused a fair bit of head-scratching until I figured out what was going on...

Case sensitive filesystems

I've done most of the dev work for this site locally, on macOS 10.14, using `bundle exec jekyll serve` to preview and test things. And when I finally pushed the whole thing up to GitHub and switched on the GitHub Pages feature, everything worked – except the flag images in the schedule sidebar. And it took me a good couple of hours to figure out what was going on.

My original `flags.scss` file looked like this:

```
$countries: ad, ae, af, ag, ai, al...

@each $country in $countries {
  .flag-#{$country} {
```

```
    background-image: url(images/flags/flat/64/#{$country}.png);
  }
}
```

which generated a bunch of CSS rules that looked like this:

```
.flag-ad { background-image: url(images/flags/flat/64/ad.png); }
.flag-ae { background-image: url(images/flags/flat/64/ae.png); }
```

Now, if you look closely at the files in the GoSquared flagset I'm using, you'll notice the filenames are:

```
AD.png
AE.png
AF.png
```

I'd dropped their entire flag set into my project, and pushed the whole thing to GitHub.

Now, I'm guessing that GitHub Pages is hosted on Linux. And Linux uses case-sensitive filesystems, whereas macOS – where I'd been testing everything – uses a 'case preserving' filesystem. In other words, if you ask macOS for `ae.png`, it'll happily give you `AE.png`, but on Linux, `ae.png` and `AE.png` are different files.

So my CSS rule was telling my browser to ask GitHub Pages for `ae.png`, and GitHub's Linux servers are going "nope. Not found." – 'cos the only file they've got is `AE.png`, which is COMPLETELY DIFFERENT. Obviously.

I could see two ways to fix this. One was to rename all the files in the GoSquared collection… but it turns out it's surprising fiddly to rename a file from `FILENAME` to `filename` on macOS. The other, easier way is just to uppercase all the ISO country codes in the SCSS list, like this. I threw a `to-lower-case($country)` into the class name there because, well, I think uppercase CSS rules are vulgar.

```
$countries: AD, AE, AF, AG, AI, AL...

@each $country in $countries {
  .flag-#{to-lower-case($country)} {
    background-image: url(images/flags/flat/64/#{$country}.png);
  }
}
```

DNS, HTTPS, Cloudflare and GitHub Pages

The last piece of the puzzle was to get the whole site using `dylanbeattie.net` (no `www`) as the canonical URL, and to get everything running over HTTPS. I host all my DNS with Cloudflare, because it's free and works really well – and, like a lot of people, I'd relied on Cloudflare's

HTTP+DNS proxy service for a while to provide HTTPS for my old Blogger site (check out Troy Hunt's Here's Why Your Static Website Needs HTTPS for more on why this is a good idea.)

I wanted to use GitHub Pages to enforce HTTPS, but when I switched the Cloudflare DNS for dylanbeattie.net to point to GitHub's servers, I couldn't switch on the option to enforce HTTPS – instead, I got this error:

> *Enforce HTTPS — Unavailable for your site because your domain is not properly configured to support HTTPS*

and when browsing my new site, I got a warning that:

> *This server could not prove that it is dylanbeattie.net; its security certificate is from www.github.com. This may be caused by a misconfiguration or an attacker intercepting your connection.*

After about a day of head-scratching – change a DNS setting, wait six hours to see if anything happens, change something else, repeat – I contacted GitHub Support.

Turns out that Cloudflare's DNS+HTTP Proxy feature actually interferes with the certificate issuing mechanism used to support HTTPS on GitHub Pages. GitHub asks for DNS servers so it can issue a certificate, but if you've enabled Cloudflare's HTTP proxy feature (which is on by default, even on their free plan), Cloudflare responds to the DNS query with its own server addresses and so GitHub can't see that your domain is pointing at GitHub Pages.

Logged into Cloudflare, switched the records for dylanbeattie.net over to DNS only, and boom – certificate was issued within the hour and everything was up and running.
