A full website setup with Pandoc and GitHub Pages

Matthew Low

03/07/2020

One night, after being inspired by a number of similar projects online, I decided to try reworking my personal website into a fully automated Pandoc and GitHub pages system. I had tried similar projects in the past (a full Markdown-based setup with LaTeX output was one of my attempts) but I never really got to a level with my setup that I was particularly happy with.

My requirements were very simple:

- A very light & minimal setup process, with little need for constant maintenance
- The ability to save Markdown files in folders, and the website to generate itself
- Most importantly, the ability to write math documents and be able to export them to LaTeX if needed.

The final requirement naturally led me to Pandoc, a fantastic piece of kit which allows conversion between a massive number of formats. It is most commonly used by those wanting to write pretty documents in Markdown and then exporting it to LaTeX, but can also be used to generate HTML from Markdown files (with KaTeX/MathJax support!). This is where I began my journey.

The setup is now wonderfully functional. With a simple VS Code workspace setup and a very simple shell script, combined with the power of Pandoc, I now have an incredibly simple static site generator which compiles any `.md` file into `.html` every time I save. Here's how I did it.

1 Part 1: The structure

This setup relies on essentially three things:

- Pandoc;
- VS Code with the Run On Save extension;
- A shell script called `generate.sh`.

Now you just need an empty folder. Setup a directory structure something like this:

```
blog
    blog.html
```

```
        blog.md
        posts
            pandoc-github.html
            pandoc-github.md
    css
        custom.css
    docs
    index.html
    index.md
    private
    script
        generate.sh
    site.code-workspace
    template
        matt.html
```

What are all of these for? Well, this directory structure is just a suggestion; the whole point of this setup is that I can drop Markdown files *anywhere* and the setup will automatically compile it into HTML which you can link to anywhere. This is what I *personally* have setup:

- `blog` is where your blog posts can go;
- `css` is where custom CSS for the websites goes;
- `docs` is where any other Markdown files go; I put notes and other ramblings here;
- `index.html` is the *auto-generated* home page for my website;
- `index.md` is the source file for my home page;
- `private` is for private files that I don't want up on my GitHub repo. This is ignored out by my `.gitignore`;
- `script` is where the `generate.sh` script is placed;
- `site.codeworkspace` is a VS Code workspace setup that handles auto-compilation;
- `template` is where my Pandoc templates go (currently I only have an HTML one setup, but I plan to add a LaTeX template into my workflow);

Really, the only key parts of this setup are `css`, `script`, `site.codeworkspace` and `template`. The rest can be whatever you want.

2 Part 2: Pandoc

So how does Pandoc come into this? To get a peek, have a look at `generate.sh`:

```bash
#!/bin/bash

target="index"
if [ -n "$1" ]; then
  target=$1
fi
```

```
root="/Users/mattchrlw/Google Drive/Site"

cd "${root}"

pandoc "${target}" \
    -o "${target%.*}.html" \
    --template "template/matt" \
    --highlight-style kate \
    --standalone \
    --toc \
    --toc-depth=3 \
    --css "/css/custom.css" \
    --mathjax

pandoc "${target}" \
    -t beamer \
    -s \
    --pdf-engine=xelatex \
    --resource-path="$(dirname "$target")" \
    -o "${target%.*}-slides.pdf" \
```

This script takes in some file name, and automatically generates an HTML file in the same directory, using the provided template and CSS files. The `--standalone` flag is used to ensure every page is generated as its own standalone HTML file. I also use the `--katex` flag for neat mathematics output.

The template is the most important part of this setup, and is what allows a permanent header/footer setup across all pages, creating a sense of consistency instead of just being a bunch of Markdown files thrown together. I created my template by just saving the default template

```
pandoc -D html > template/matt.html
```

and then messing with it. All I added was this:

```
$if(plain)$
$else$
<header>
  <div class="nav">
    <h2>Matthew Low</h2>
      <div class="buttons">
        <a href="/index.html">Home</a>
        <a href="/blog/blog.html">Blog</a>
        <a href="#">About</a>
      </div>
  </div>
</header>
$endif$
```

```
...
$if(plain)$
$else$
<small>(C) 2020 Matthew Low. This site is built with ...</small>
$endif$
```

These modifications just add a header and footer to the page if no “plain” flag is specified in the YAML (see below for an example) of a Markdown file. If “plain” is specified, it will just generate the body of the document, no header, no footer.

The CSS is, of course, what makes this setup polished. CSS can be written the same way you would write CSS for any ordinary website.

Now, whenever you make a Markdown file in your site folder, just put this YAML in the top:

```
---
title: Blah blah blah
---

## Here is some content

Blah blah blah
```

3 Part 3: The “automation”

Now we can generate websites just by running `script/generate.sh` on whatever page we would like to generate. Of course, we would rather do this automatically. This can be done very easily with the aforementioned Run On Save extension for Visual Studio Code (my editor of choice, I’m sure you can find similar tools in other editors). Just set this up in your workspace settings JSON file:

```
"settings": {
  "runOnSave.commands": [
    {
      "match": ".*\\.md$",
      "command": "cd ${workspaceFolder}; script/generate.sh ${file}",
      "runIn": "backend",
      "runningStatusMessage": "Compiling ${fileBasename}",
      "finishStatusMessage": "${fileBasename} compiled"
    }
  ]
},
```

This will automatically run the `script/generate.sh` script on any Markdown file that is saved in the workspace, automating the generation process. The `"runIn": "backend"` combined with the status message files make it so when you compile the website, it will show this in your VSCode status bar:

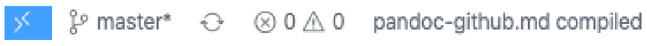

Figure 1: Status bar.

There are plenty of other extensions that can improve your quality-of-life when blogging/writing docs in this setup, including:

- Live Server, essential for previewing your site as you edit. It will even live reload on save!
- Markdown All In One, for making Markdown editing more pleasant, includes LaTeX syntax highlighting.
- Markdown Paste, for pasting images directly into VS Code and auto-linking to them in Markdown files.

4 Part 4: The deployment

Now we're ready to put this up on GitHub pages! Connect your Git repository to a GitHub repo (named `username.github.io`) and follow these simple instructions.

That is all. You now have a fully functional blogging/documenting/portfolio setup, entirely based on Markdown, Pandoc, GitHub Pages and a very simple shell script.

You can find the code for this setup in my GitHub pages repository here.