Markdown to PDF README

A markdown file, and resulting PDF that contains several formatting scenarios

Craig Parker (That's Pahkah, in Maine)
craigp@fossfolks.com
Mar 1, 2020

Contents

A Little History

Back in December of 2018, I wrote **Convert Markdown to PDF with Pandoc and LaTeX**, and showed how I got the Markdown to PDF process working, which saved the technical writers at my company hours and hours of work. Being able to whip through the process in seconds instead of days made life a whole lot easier.

But, there were a few issues. As I said previously, I wasn't happy with the headings. Our finished PDFs had a lot of vertical white space between headings and code blocks. Another weird issue was that LaTeX didn't really have equivalents to what is in HTML. Instead of `<h1>` and `<h2>`, LaTeX has are section and subsection (with some other names for smaller headings). But they really only go down as far as `<h5>`.

While we could live with some of these idiosyncrasies, I was having trouble with a particular project. There was a character that just wouldn't render, regardless of which font I was using. As part of the ensuing ruckus, I began to look for other ways to convert Markdown to PDF.

LegoLand Darth Maul

Convert Markdown to PDF in a New Way

I found was a similar process, and the moving parts were things I knew oodles more about than LaTex. It dawned on me one day that I could use Pandoc to get from Markdown to PDF, then use a CSS template (not LaTeX) and something called WeasyPrint to get from there to a finished PDF. So I charged off in that direction, and a week or so later had some beautiful results.

AT-ATs chasing my kids

Because this is now a two-step process though (Markdown to HTML, then HTML to PDF), it's not *quite* as simple as the previous method. So I've got a script that does it for me, after asking a couple of questions. It creates a PDF based on the HTML and a stylesheet (CSS file).

So how does one go about convert Markdown to PDF? Well, it's a fairly simple operation. The easiest way I've found is to make a Markdown file, then run a script on it. That script prompts for some input, the executes a couple of commands. I make changes as I need to, but this is how the script sits at the moment:

The Markdown to PDF Script

I won't post the whole script here (you've probably already read it on the blog, or if you've got this README then you've probably cloned the GitHub repository and can read it right off of your own hard drive) but I'll give a rundown of what it does.

Blank Line

First, the script prints out a blank line, just to have a bit of a buffer between the command prompt and the script output.

List of Markdown Files

Next, the script lists out all of the files in the current directory that have a .md extension. It does not print out the .md file extension.

Prompt for Filename

Then it prompts for one of those filenames.

Format

Finally, there's one last question. Do we want a Portrait or Landscape layout?

Doing the Work

With all of those prompt responses done, now the script takes the choices you've made, and uses Pandoc to make an HTML document (which includes putting the appropriate stylesheet, `style-landscape.css` or `style-portrait.css`, between

the `<head>` and `</head>` tags). Immediately afterward, it calls WeasyPrint, which uses that HTML and CSS to create the final PDF.

There are comments in the Bash script itself that explain how it works as the script progresses.

The Markdown to PDF CSS Template

I won't post the whole CSS here, but I'll run through some of the things that might make life easier for anyone trying to give this a spin. There's a README.pdf that explains it all too, with examples.

Declaring Fonts

Right up near the top of the file, we declare fonts. I've got the fonts that IBM open sourced a couple years ago, Plex. There are sans (for most everything) and monospace (for code and preformatted text) fonts I've declared. But you can plug in any fonts you want there. Just make sure to grab the regular, bold, and italic version of each, and put them in the same directory that I've stuck the Plex fonts in. Then refer to them the same way I did.

Styling Different Pages

Below the fonts declaration, there's an `@page :first`. This is the cover page. There is a background image declared here, (the FossFolks logo in the example PDF). I've set an image size of 450×300 and gotten it working for me. But if you want a different sized image, you'll have to finagle the margins, playing with them until the image sits where you want it to. Whatever you end up with, when you move forward on other PDFs you should probably stick to the same sized cover images so you don't have to keep dorking with CSS.

The next type of page that's declared is `@page no-chapter`. This is the *Table of Contents* page. Things here are set up pretty much the same as on the regular pages. But change them here (to do something like get rid of the logo and page number in the lower right maybe) if you want, and it won't affect the rest of the document.

Next up is the `@page`. Anything after the Table of Contents page(s) in the PDF is effected. This looks about like the TOC does, but we can tweak here to alter the rest of the document.

At last there's `@page :blank`. Honestly, I don't quite recall what exactly this effects. If I ever figure it out again, I'll update the README.

The remainder of the stylesheet should look familiar to anyone who knows anything about CSS.

Note that there are some settings that I've put in `@page :first` and `@page no-chapter`, but left them with blank values, like this little ditty:

```
@top-center {
 background: none;
 content: ''; }
@top-right {
 background: none;
 content: ''; }
```

If we just deleted them instead of leaving them there with empty values, then they're be overridden with what's in the `@page` section. It's a little hinkey, but you'll see when you play with it.

Table of Contents Problem

There was one wee little issue, with the table of contents. When a list (the H3 headings are the list items) went over a page break, the items on the first page's part got bumped up a bit. Check out the README.pdf in the Git repo to see what I'm talking about.

The fix is to edit `boxes.py`. You'll have to hunt for it, but it's sitting in whichever directory WeasyPrint got installed into. Try this to find it:

```
sudo find / -name boxes.py
```

It should be somewhere like: `/usr/local/lib/python3.x/dist-packages/weasyprint/formatting_structure` (on a Linux machine), or `/usr/local/lib/python3.x/site-packages/weasyprint/formatting_structure/` on a Mac.

We're looking for something in the vicinity of line 320-350 of that file (which may change in future versions) that reads:

```
if (start or end) and old_style == self.style:
```

It essentially means *"if something is equal to something else"*, and we need it to say *"if something is NOT equal to something else"* instead. We do it by replacing the first of those equals signs with an exclamation point, like this:

```
if (start or end) and old_style != self.style:
```

Rendering the TOC should work fine after this change.

Lists over Page Breaks Issue

This is something the WeasyPrint community is still looking into (as of 3/2020)... If there's a list that goes over a page break, the first list item on the next page of the list looks a little weird. It's almost like the number/bullet is bumped down a bit from the contents of that number/bullet point. I kind of work around it for now (like, by making lower-level headings instead of list items -- take the `<h3>` headings under *The Markdown to PDF Script* `<h2>` heading in this document, for example) but it has been labeled a bug by the community. Someone will fix it.

The HTML Template

There really isn’t much else we have to mess with. There’s a `default.html` in the templates directory where we were able to customize some different things that show up. There's a title, subtitle, author, email, and date, and we can see where those are showing up in a finished PDF. If we wanted to edit them though, to have something different showing up on the cover page, this is where we'd do it.

Some Weirder Examples

The headings **The HTML Template** and **Some Weirder Examples** are `<h2>` type headings. In this HTML/CSS setup, each one of those starts a new page.

H3

This is **H3**

H4

This is **H4**

H5

This is **H5**

H6

This is **H6**

Blank Lines Required

That's a joke, really. Blank lines aren't required between things so much, like they are with the LaTeX template I cooked up. Because it's easier to read the Markdown though, I'll probably just keep doing it.

A blank line is ***still*** required before a bulleted list though. And speaking of lists...

Lists

Using `:::dubbah` ahead of a list, and `:::` after it (with blank lines above and below each), we are essentially create a `<div class="dubbah"> </div>` tags in the resulting HTML. This means we can create a different list style for each of those `div` classes. I've made a few, but the sky's the limit.

Normal List

This is the default list style:

- Bullet 1 (top-level)
    - #1 Level 2 Bullet
    - #2 Level 2 Bullet
    - #3 Level 2 Bullet
- Bullet 2 (top-level)
    - #1 Level 2 Bullet
    - #2 Level 2 Bullet
    - #3 Level 2 Bullet
- Bullet 3 (top-level)
    - #1 Level 2 Bullet

- #2 Level 2 Bullet
    Level 4
        Level 5
            Level 6
                Level 7
                    Level 8
                        Level 9
- Bullet 4 (top-level)
    - #1 Level 2
        Level 3
            Level 4
                Level 5
                    Level 6
                        Level 7
                            Level 8
- Bullet 5 (top-level)
    - Another #1 point

        ```
        A code block nested in a list
        ```
- Bullet 6 (top-level)

Bold Top Level, Second Level Bullets

- **Bullet 1 (top-level)**
  - #1 Level 2 Bullet
  - #2 Level 2 Bullet
  - #3 Level 2 Bullet
- **Bullet 2 (top-level)**
  - #1 Level 2 Bullet
  - #2 Level 2 Bullet
  - #3 Level 2 Bullet
- **Bullet 3 (top-level)**
  - #1 Level 2 Bullet
  - #2 Level 2 Bullet

Level 4

Level 5

Level 6

Level 7

Level 8

Level 9

- **Bullet 4 (top-level)**
    - #1 Level 2
        Level 3
            Level 4
                Level 5
                    Level 6
                        Level 7
                            Level 8
- **Bullet 5 (top-level)**
    - Another #1 point

        ```
        A code block nested in a list
        ```
- **Bullet 6 (top-level)**

Bold Top Level, Second Level Numbered

- **Bullet 1 (top-level)**
    1. #1 Level 2 Bullet
    2. #2 Level 2 Bullet
    3. #3 Level 2 Bullet
- **Bullet 2 (top-level)**
    1. #1 Level 2 Bullet
    2. #2 Level 2 Bullet
    3. #3 Level 2 Bullet

- **Bullet 3 (top-level)**
    1. #1 Level 2 Bullet
    2. #2 Level 2 Bullet
        Level 4
            Level 5
                Level 6
                    Level 7
                        Level 8
                            Level 9
- **Bullet 4 (top-level)**
    1. #1 Level 2
        Level 3
            Level 4
                Level 5
                    Level 6
                        Level 7
                            Level 8
- **Bullet 5 (top-level)**
    1. Another #1 point

        ```
        A code block nested in a list
        ```
- **Bullet 6 (top-level)**

Bold Numbers on Top Level, Second Level Bulleted

1. **Bullet 1 (top-level)**
    - #1 Level 2 Bullet
    - #2 Level 2 Bullet
    - #3 Level 2 Bullet
2. **Bullet 2 (top-level)**
    - #1 Level 2 Bullet
    - #2 Level 2 Bullet
    - #3 Level 2 Bullet
3. **Bullet 3 (top-level)**
    - #1 Level 2 Bullet
    - #2 Level 2 Bullet

        Level 4

            Level 5

                Level 6

                    Level 7

                        Level 8

                            Level 9

4. **Bullet 4 (top-level)**
    - #1 Level 2
        Level 3
            Level 4
                Level 5
                    Level 6
                        Level 7
                            Level 8
5. **Bullet 5 (top-level)**
    - Another #1 point

        ```
        A code block nested in a list
        ```
6. **Bullet 6 (top-level)**

You get the idea. Anyone with a knowledge of CSS can make these lists do all sorts of things. You can use that `div` trick to style all manner of things, like tables, preformatted text areas (`<pre` tags), images, etc.

Other Items We Might Run Into

Here is a horizontal line:

> Here is a block quote: Blahdy blah blah MOO I'm a goat.

> ### And here is an H5 header, inside a quote, with a quoted list under it:
>
> 1. ONE list item! Ah ah ah...
> 2. TWO list items! Ah ah ah...
>
> Now for good measure, let's throw in some example code:
>
> ```
> return shell_exec("echo $input | $markdown_script");
> ```

Tables

Eghads... Making tables work in LaTeX is a ruckus, Doing them this way (using an HTML/CSS template with WeasyPrint) lets them just work. There's no LaTeX tinkering required, whatsoever. I've written a PHP app that keeps track of chord charts for songs, and uses Markdown tables. Here's what one looks like:

Intro		2	5	1	5						
		1	4	1	5-/1			4	4	1	3/6
		2	5	1/6	2/5						
Out		1/6	2	5	1						

Forcing Page Breaks

You may not like where something lands, page-wise, and want to kick it down to the next one. There's a div class for that, called `pb`. To do a page break, just use the same kind of div class hack we were doing earlier:

```
:::pb
:::
```

To see how this works, take a looksie at the raw Markdown, just above the **### Forcing Page Breaks** heading. I did it there. You'll be able to see that there's a break right before that heading in the PDF.

Colored Code

Pandoc uses GitHub flavored markdown, which means we can label a code block with a language if we want. We'll get some pretty colors. I haven't dorked much with this, so feel free to let me know if you cook something up that does a little better job of color coordinating scripts and languages.

Python	Bash
`print('Hello, world!')`	`echo "Hello World!";`

Markdown to PDF Summary

When you step back and look at it, this is a pretty cool process, especially when you consider where this whole thing started (ground zero, using an Adobe product that only kind of worked). The Pandoc and WeasyPrint communities offered a plethora of handy information as I embarked on this little side trip.

Now, instead of fighting to squeeze a nice PDF out (and having to *really* learn LaTeX), I can cough up a pretty slick looking PDF with some CSS, which is something I'm *a lot* more familiar with. I'm hoping that sharing it here will save some pour soul hours of screeching and headbanging on their desk trying to get things working. That "helping prevent headaches" is the whole reason I started **FossFolks**, really.

Put it through it's paces, and let me know how you make out.