pdfulator, a Markdown to PDF converter

Tom Gidden <tom@gidden.net>

September 2024
v.1.1.1

https://github.com/tomgidden/pdfulator

Introduction

`pdfulator` is a Markdown-to-PDF converter using:

- *Docker* - a containerization engine
- *Pandoc* - a document format converter
- *PagedJS* - an library for paginating HTML
- *Chromium* - a PDF renderer ;)
- some CSS, fonts, and some minor glue scripts.

It's not rocket science, but it's fiddly and usually not worth spending the time to assemble into a single utility. That's what this is for.

Installation

The utility is packaged as a Docker image, as the dependencies are messy. You can pull down the image from Docker Hub:

```
docker pull tomgidden/pdfulator
docker tag tomgidden/pdfulator pdfulator
```

or you can build it yourself. To build the image, run:

```
make build
```

or manually:

```
docker build -t pdfulator .
```

Due to the inclusion of *Chromium* the image footprint is large.

Usage

Given a Markdown file `foo.md`, if you have the GNUmakefile in the current folder, you can run:

```
make foo.pdf
```

or the hard way:

```
docker run --rm --init -i tomgidden/pdfulator - < foo.md > foo.pdf
```

That's it. For continuous update whenever the Markdown changes, run:

```
make watch
```

or the hard way:

```
docker run --rm --init -v $(pwd):/in tomgidden/pdfulator --watch
```

Customisation and development

Immediate single-file mode

```
DEBUG=1 make foo.pdf
```

That should do three things:

- Use the current `defaults` folder rather than the baked-in copy in the Docker image;
- Use the current folder's `entrypoint.sh` rather than the baked-in copy in the Docker image;
- Preserve the intermediate `work` folder, containing the generated HTML file.

As a result, you can tweak the CSS and other things in `defaults` and quickly see the result without having to rebuild the Docker image.

For example, a dev workflow might look like this:

```
DEBUG=1 make -B foo.pdf && open foo.pdf
```

or even better, just use watch mode:

```
DEBUG=1 make watch
```

Once you're happy with the style, you can build your own version of the image and use it anywhere without having to also transfer any assets.

Styling

The current CSS is a simple Humanist "white-paper" layout typical of my general tastes. I was influenced in my youth by the original 1995 Java™ white papers and other documentation from Sun, and this is somewhat simplified version. It's very rough-and-ready, but it does enough for me right now. I have been wondering if it's worth having multiple themes somehow.

You can override the styling by adding a `theme` folder with custom stylesheets, fonts and other assets. This should override the ones in `defaults`:

```
EXTRA_DOCKER_OPTS="-v $(pwd)/my_css:/theme" make foo.pdf
```

Logo

If the theme folder contains a file called `logo.svg`, it should appear in the top-right. However, to customise the positioning and style, you'll have to use a little custom CSS. As there's a little quirk somewhere, you have to add a little content, even though it's specified in the main stylesheet:

```
@page {
  @top-right {
    content: string("");                  /* seemingly important*/
    background-image: url(/theme/logo.svg);
    background-size: 108pt auto;          /* here's how to size the image */
    /* control `height`, `margin-top` and `margin-right` appropriately, but check
     * it doesn't crash into content on page 2 onwards.
     */
  }
}
```

Document metadata

To support the top front-matter in a Markdown file, you can include a YAML block at the top of the file delineated by `---` and `...`; see this `README.yaml` file for an example.

Unfortunately, other Markdown renderers (notably *GitHub*) may include this as garbled nonsense in their output.

If this bothers you, you can include the YAML as a separate file next to the Markdown instead, eg. `foo.md` and `foo.yaml`.

Metadata entries

- `title` - The title of the document. If not included, the first top-level heading (`#`) in the document will be hoisted to the title. It seems weird to leave the title out of the bare Markdown, but this hoisting behaviour is a little unusual.

- `date` - A block containing `month` and `year` to be displayed in the front-matter.

- `revision` - A revision number, to be displayed in the front-matter and the footer.

- `copyright` - An optional copyright attribution to be included in the page footer. If set, it will be preceded by '©', the year (determined either from `date.year` or `year` in the metadata, or the current year), and then the attribution.

- `footer` - An optional text to be included in the page footer, added to the copyright if there is one.

- `pdfulator_features` - See below

Example metadata

```
title: Set this or just let it use the first `#`

date:
  month: September
  year: 2024
revision: First Draft
authors:
- firstname: Tom
  surname: Gidden
  email: tom@gidden.net
- name: D. C. O'Author
  affiliation: Institute of Documentarian Affairs
- firstname: Harold
  surname: L'astname
  affiliation: Institute of Documentarian Affairs

copyright: Tom Gidden & Institute of Documentarian Affairs
footer: Confidential

pdfulator_features:
- no_wide
- no_wide_pre
- shade_monospace
- strong_monospace
- narrow_monospace
```

`pdfulator_features`

The document metadata see above can include a `pdfulator_features` line or list that contains a few optional choices controlling formatting. These can be left in but ignored (ie. disabled) by prefixing them with `no_`, or just removing them.

These include:

- `wide` - Don’t indent the main body text. This gives extra space, useful especially for pre- and code-blocks, but at the expense of the left margin.

- `wide_pre` - Don’t *further* indent pre-formatted text blocks. Again, extra space, but less easily read.

- `shade_monospace`, `shade_pre`, `shade_code` - Use a light grey background for monospaced font material, or just block (`_pre`) or inline (`_code`) sections. This is to further distinguish from the main text.

- `strong_monospace`, `strong_pre`, `strong_code` - Use a bolder font for monospaced. By default it uses a lighter font to try to distinguish from body text, but this goes the other way.

- `narrow_monospace` - Use a narrower font for monospaced content, to try to get it to fit nicely on a page.

- `justify` - Use full justification for body text. I like this, but some of my designer friends say it’s bad and ragged edge makes for better typography.

- `strong_href` – embolden hyperlinks to make them stand out.

Adding a logo

If there is a file `logo.svg` in the `theme` folder, it will be used in the top-right header box.

TODO

[X] *Themes.*

[ ] *TOCs*

[ ] *Better images*. You can put things in the `theme` folder that can then be referenced for use in `DEBUG=1`, and you can (presumably) use remote URL files. However, there's no easy way to pass them into the container for processing at this time. More thought needed.

[ ] *Improved layout*. This is still a work in progress.

[ ] *Built-in themes*. Instead of having to make a theme, have some premade ones available with settings in metadata.

[ ] *Comprehensive support for the format*

[ ] *HTML*, *EPUB*, etc. Given the use of *Pandoc* these should be very simple to support. I'm just an old fart that likes neat A4 documents even if I never actually print them out.

[ ] Testing of `--watch` and improvement on file globbing and so on.

Any feedback, assistance or code contributions welcome.

History

I've had various DocBook or Markdown to PDF toolchains using XSL-FO, Apache FOP and other tech since the late nineties, usually named "docbot" as I used them in web and email services, Slack bots, etc.

There are a lot of projects called `docbot` and a lot called `md2pdf`. None of them do exactly what I want, though. Decent pagination was served by the DocBook XSL sheets, but HTML-based ones have been lacking. Most still do.

And using DocBook as an intermediary is a bad idea; while DocBook is richer than Markdown (and arguably a far better choice for software documentation) the element semantics aren't suitable for generic documents.

PagedJS now seems to make the HTML route a good option, being a capable polyfill for the print features of CSS3, allowing for running headers and footers and so on.

- v1.0: Released for a short time as “docbot”
- v1.1: Renamed to “pdfulator” and refactored to give a basic “–watch” mode. This is still a work in progress.
- v1.1.1: Themes
- v1.1.2: Preservation of work folder (now in `/work`), and minor styling for logos

Licence

I hereby release the parts of this project I have written freely under Creative Commons CC0 1.0. Attribution and code contributions would be nice though.

This clearly does not apply for the third-party sub-components it uses or the fonts in the `assets` folder which are released under their own licences: OFL and the GUST/LPPL licence as appropriate.

I’ve included the fonts (and their licences) in this package purely for performance and simplicity: otherwise they either need to be downloaded on each invocation, or cached somehow between Docker runs, leaving junk on the host machine. I hope that’s okay within the terms of those licences.