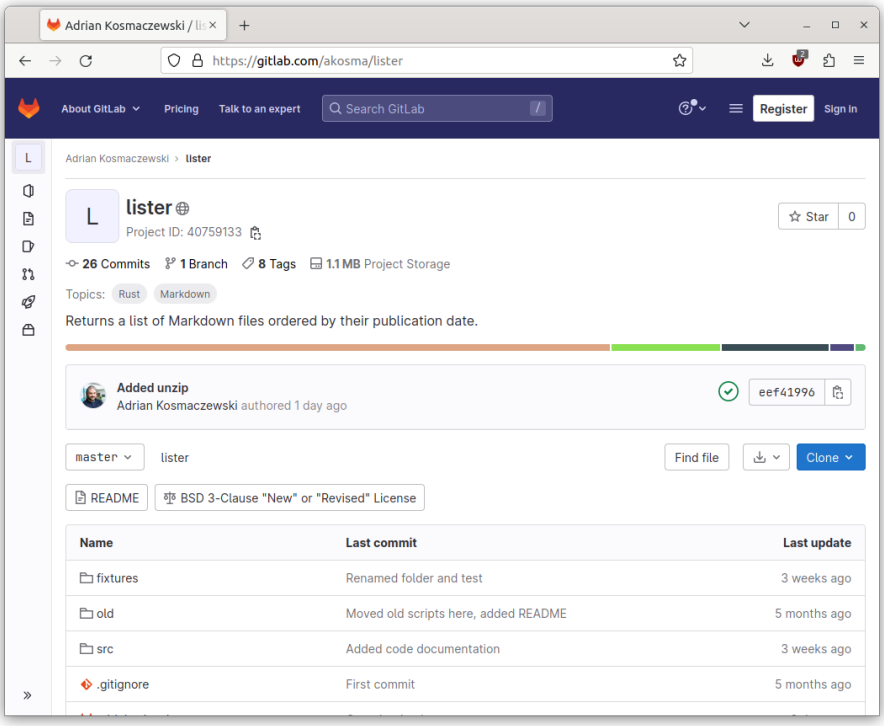
Exporting Hugo to PDF

Adrian Kosmaczewski

2023-04-28

Hugo[1] is fantastic but it misses one key functionality: **the generation of PDF files.** This article provides a possible solution for it using Podman, and a custom tool built in Rust.

The Rust project is called lister[2] and is available in GitLab for you to use and extend. It contains a small utility called `lister` that does only one thing: it lists all Hugo source files in Markdown format and returns a list of files by ascending publication date. This means that the tool opens each file, reads the `date:` parameter on top, and uses that information to order the files.



[1] https://gohugo.io/
[2] https://gitlab.com/akosma/lister/

Container Image

The container image is available at `registry.gitlab.com/akosma/lister:1.7`. The `Dockerfile` below shows how it's built, with a 2-step process, first building the Rust executable, and then creating an image with Pandoc[3], TeX Live[4] with XeTeX[5], Poppler[6], git-restore-mtime[7][8], and some other tools.

```
# Step 1: build Rust tool
FROM docker.io/library/rust:alpine as builder
RUN apk update && apk add --no-cache openssl-dev musl-dev

# Compile dependencies
RUN USER=root cargo new --bin lister
WORKDIR ./lister
COPY ./Cargo.lock ./Cargo.lock
COPY ./Cargo.toml ./Cargo.toml
RUN cargo build --release

# Compile the app itself
RUN rm src/*.rs
RUN rm ./target/release/deps/lister*
ADD src ./src
ADD fixtures ./fixtures
RUN cargo test
RUN cargo build --release

# Step 2: runtime image
FROM docker.io/pandoc/core:3.1.1.0-ubuntu
ENV DEBIAN_FRONTEND noninteractive
RUN apt update && apt install -y texlive texlive-xetex \
     poppler-utils make git-restore-mtime curl unzip
WORKDIR /build
COPY --from=builder /lister/target/release/lister /usr/local/bin/lister
```

The compilation of Rust code can be quite long. This is why the `Dockerfile` is built in such a way that, when used in your own laptop, the dependencies will only be downloaded once and cached. This speeds up the process of debugging and creation until you're ready to ship, wink wink.

[3] https://pandoc.org/

[4] https://tug.org/texlive/

[5] https://en.wikipedia.org/wiki/XeTeX

[6] https://en.wikipedia.org/wiki/Poppler_(software)

[7] https://github.com/MestreLion/git-tools

[8] The `git-restore-mtime` tool is there to reuse the `Makefile` in a CI/CD pipeline, for example GitLab, avoiding the rebuilding of all files by first setting the dates of files (used by `make` to keep track of things) to that of their last modification.

Makefile

Here's the `Makefile` that drives the creation of those PDFs using the `lister` container image:

```
PODMAN := podman run --rm --volume "$${PWD}":/build
IMAGE_LISTER := registry.gitlab.com/akosma/lister:1.7
PANDOC ?= $(PODMAN) --entrypoint "pandoc" $(IMAGE_LISTER)
PDFUNITE ?= $(PODMAN) --entrypoint "pdfunite" $(IMAGE_LISTER)
LISTER ?= $(PODMAN) --entrypoint "lister" $(IMAGE_LISTER)
SOURCES := $(shell $(LISTER) content/posts)
OBJECTS := $(SOURCES:content/posts/%/index.md=build/pdf/%.pdf)

.PHONY: all
all: fullpdf

.PHONY: clean
clean:
    rm -rf build

.PHONY: debug
debug:
    @echo $(SOURCES)

.PHONY: fullpdf
fullpdf: build/pdf build/pdf/_full_website_archive.pdf

build/pdf:
    mkdir -p build/pdf

build/pdf/%.pdf: content/posts/%/index.md
    $(PANDOC) --write=pdf --pdf-engine=xelatex \
        --variable=papersize:a4 --variable=links-as-notes \
        --variable=mainfont:DejaVuSans \
        --variable=monofont:DejaVuSansMono \
        --resource-path=$$(dirname $<) --out=$@ $< 2> /dev/null

build/pdf/_full_website_archive.pdf: $(OBJECTS)
    @echo "Building _full_website_archive.pdf"
    @$(PDFUNITE) scripts/pdf_cover.pdf $^ $@
```

The `make debug` command shows the contents of the `SOURCES` variable, with the list of Markdown files ordered by publication date, excluding those in the future.

If the source file of this article is located in the path `content/posts` (as is the case with Hugo) you can use this `Makefile` as follows:

```
$ make build/pdf/exporting-hugo-to-pdf.pdf
```

The important thing in the `Makefile` above is that the PDF files will be regenerated only if their corresponding Markdown file was updated.

The filename of the PDF file will automatically correspond to the slug of its source article.

Finally, we concatenate all PDF files into a single one called `_full_website_archive.pdf` using the `pdfunite` tool of the Poppler[9] package[10]. I added a `pdf_cover.pdf` file made with LibreOffice just for the sake of completeness.

To show the product of these scripts in action, you can download the PDF version[11] of this article. The file features links as footnotes, images, links, formatting, and respects the source code highlighting of the code snippets.

[9] https://en.wikipedia.org/wiki/Poppler_(software)

[10] At the time of this writing, the final PDF file, including all of the articles in this blog, takes around 30 minutes to build on my 2019 ThinkPad Lenovo X1 Carbon, and has... almost 1800 pages!

[11] exporting-hugo-to-pdf.pdf