Dear Reader,

welcome to my personal website — a place where I publish random stuff that may or may not be of interest to the general public. This particular document is mainly an index that points to the various things I make available on the web. You could start by reading about some of the software I wrote; or you may head over to the blog to see if there are any new posts.

Frank Seifferth
frankseifferth@posteo.net

RSS Feed

updated
24 June 2025

Chapter 1
Blog / Writing

24 June 2025

Combining pdf and zip proof of concept

https://tilde.club/~seifferth/blog/pdf-zip-poc/

1 June 2024

Removing Editing Restrictions from Office Documents (Paged Out!, issue 4)

https://pagedout.institute/download/PagedOut_004_beta1.pdf#page=62

18 December 2023

Creating PDF/Plain Text Polyglots with LuaLaTeX (Paged Out!, issue 3)

https://pagedout.institute/download/PagedOut_003_beta1.pdf#page=26

12 April 2023

On why this website is a pdf file

https://tilde.club/~seifferth/blog/why-pdf/

Chapter 2
Software

Writing Utilities

mkpdf https://github.com/seifferth/mkpdf

Mkpdf is a wrapper around pandoc and latexmk that I use to typeset markdown documents. It supports specifying latex templates in the yaml frontmatter and allows to use biblatex or natbib to process bibliographic references. It also supports the use of custom 'magic lines' in the template which can be used to specify which latex engine and bibliography package to use in combination with that particular template.

jotter https://github.com/seifferth/jotter

Jotter is a set of scripts that bring ctags-like functionality to the world of markdown note taking. It is mainly designed for keeping collections of book and article excerpts organised, but also supports the inclusion of free-form notes that do not have any associated bibtex metadata. Jotter supports the pandoc-markdown citation syntax (strings like '`@reference`') for linking notes together and includes a number of utility scripts to provide additional features, such as listing all note ids that match a given wildcard or looking up all entries that reference a certain citekey.

cite https://github.com/seifferth/cite

Cite brings the most important feature of bloated and expensive reference managers to the command line — with a codebase of less than 30 lines of rather straightforward shell script. Which is to say: It parses bibtex files in the current directory and all parent directories and then opens a fzf-based prompt for searching and selecting citekeys. The most convenient way to use cite is to invoke it from inside a text editor that can directly insert the selected references (printed to stdout using the pandoc-markdown citation syntax) into your document.

sgit https://github.com/seifferth/sgit

Sgit is a small wrapper around git that simplifies its use for versioning prose, where the use of atomic commits and descriptive commit messages often goes against the natural way of doing things. Sgit combines the 'git add' and 'git commit' workflow into a single 'sgit save' command and defaults to storing snapshots with empty commit messages. It still creates an entirely regular git repository, however, so the full power of git is only one letter away.

Miscellaneous

unpyter https://github.com/seifferth/unpyter

While jupyter notebooks come in handy in a number of use cases, the tooling around 'jupyter nbconvert' leaves something to be desired. Most importantly, it does not support reliable back-and-forth conversion between the json-based ipynb format and a more human-friendly plain text representation. Unpyter fills this particular gap in jupyter notebook tooling by providing exactly that: Reliable back-and-forth conversion between ipynb notebooks and (almost) plain python files.

typewrite https://github.com/seifferth/typewrite

Have you ever felt the need to make a pdf viewer render a simple text file? If you do your reading on an old android tablet like myself, you just might have. Not to worry. Converting your txt file into a pdf is as simple as running

```
$ typewrite input.txt output.pdf
```

And the best thing: Typewrite does not only create a pdf; it creates a pdf/txt polyglot. The original text version is right there at the top of the output file. All you need to retrieve it is to open the pdf in a text editor.

dollar_templates
https://github.com/seifferth/dollar_templates

'Dollar templates' provides a slim (and partial, although it might have already reached that famous 80/20 mark) reimplementation of pandoc's templating engine in plain python. It provides two simple library functions for expanding pandoc-style templates from any python project; and it may alleviate the need of adding a massive dependency like pypandoc and the whole pandoc binary to a project. 'Dollar templates' also integrates rather better with python projects in general as it is written in plain python itself and thus allows expanding templates without relying on temporary files or other, even more involved forms of inter-process-communication.

timesheet
https://github.com/seifferth/timesheet

There are probably thousands of tools for tracking working times already. This one is mine.

While many of those existing tools are focussed on having users enter their working times into some sort of database, however, this one is focussed on exporting data. As such, 'timesheet' parses a simple, flexible and rather human-friendly custom file format that specifies the time when a user started or stopped working on a specific task. The time spent is then grouped by an arbitrary set of user-specified fields (such as day, month, year, task name, task description, ...) and is either exported as a csv-formatted table (that could be further processed by external tools, such as csvkit's excellent csvsql); or it is printed in a custom format using arbitrary python format strings. The latter option even lends itself to programmatically generating shell scripts that could, for example, use curl to post the data into an arbitrary html form.

It may also be worth noting that, unlike with a number of different time tracking systems, for 'timesheet', a task can be anything a user wants it to be; from a fine-grained issue or user story in a given ticketing system to a client's name or even

a simple singular item called 'work'. All a task needs to be a task is a unique id that doesn't contain any whitespace. (Tasks can still have an associated description that may even contain whitespace, however, so not to worry.)

ttm https://github.com/seifferth/ttm

Topic Modelling describes the process of using automated algorithms in order to gain a high-level overview of the semantic relationships between different texts in a possibly large text corpus under investigation. While originally proposed in Computational Linguistics, this approach has also gained increasing popularity within Digital Humanities and Digital Literary Studies in recent years.

TTM (short for TSV-based Topic Modelling) is a cli tool written in python that offers a consistent interface to various topic modelling algorithms implemented in third-party libraries. Furthermore, ttm also features an interface to a number of evaluation metrics as well as some functionality that can be used to generate human-friendly descriptions of the semantic relationships encountered through topic modelling.

In contrast to other tools that often try to offer an integrated one-size-fits-all solution, ttm encourages a mix-and-match approach to using the various steps of different topic modelling frameworks. In order to provide maximum flexibility, the data is passed along between the different steps as a tsv-formatted table, where each row represents a document (or part of a document that has been split into multiple pages) and where each step of the topic modelling process adds a new column to the dataset. This allows to easily combine the use of ttm with other tools for investigating and transforming tabular datasets, such as cut, csvkit or the visidata editor for tabular data.

gemdoc https://github.com/seifferth/gemdoc

Gemdoc is a command line script that can be used to create text/gemini+pdf polyglot files. The format of these polyglot files is heavily based on the techniques proposed in the more recent issues of the lab6 zine (hosted at lab6.com/2 and lab6.com/3 via both gemini and https). On an implementation level, gemdoc first converts the text/gemini input into a small subset of html that is then further processed by weasyprint. As a consequence, the layout of the pdf representation can be freely adjusted by supplying user-specified css stylesheets.

As a text/gemini+pdf polyglot creation tool, gemdoc supports two main use cases. On the one hand, it can be used to download — and possibly even to print — content hosted on any gemini capsule. On the other hand, gemdoc can be used to create polyglot files that can themselves be hosted both via gemini and via other network protocols. This, in turn, might be convenient for users who want to mirror their gemini capsules via https but who also wish to serve the same files via both protocols.

ultrafiche https://github.com/seifferth/ultrafiche

Ultrafiche is a tool I wrote to make pdf rendering just work™ in mobile browsers. More specifically, I wanted to make pdf rendering work™ in chrome on android, since mobile versions of both safari and firefox are actually quite good at handling real pdfs already. In order to achieve this feat, ultrafiche converts pdfs into standalone html files with embedded svg images, which is an excellent way to break almost every useful feature of the original file well beyond repair. I wouldn't even blame you for considering this one of the most appaling ways of creating a web page. On the other hand, the result looks reasonably snappy at a casual first glance, so for use cases where superficial first impressions are of overriding concern, serving ultrafiche's html-cum-svg abomination might still be a viable choice to make.

pdfcombine https://github.com/seifferth/pdfcombine

Pdfcombine is a small command line utility that allows you to merge arbitrary pdf pages from one or more input files into an output file. It is basically like pdfunite, but with improved support for specifying page ranges.

Vis Plugins

My text editor of choice is vis (https://martanne.github.io/vis/); a modern implementation of the well known vi text editor enhanced with support for structural regular expressions and multiple cursors. Among other niceties, vis also includes a very powerful lua api that can be used to write custom plugins. Over time, I also wrote a number of plugins for that text editor myself.[1] These plugins are described below.

vis-bytepos https://github.com/seifferth/vis-bytepos

The vis-bytepos plugin adds a very simple function for displaying the current byte offset of the primary cursor when pressing 'gi' in normal mode. This functionality can come in handy when editing files in binary (or partially binary) file formats where the byte offset is important; such as pdf files, for instance.

vis-editorconfig https://github.com/seifferth/vis-editorconfig

This plugin implements most of the functionality described at https://editorconfig.org/. It thus allows to use '`.editorconfig`' files for specifying things like tab width, indentation style or the language settings to use when running a spellchecker. It also implements some of the more resource-intensive functionality as hooks that can optionally be executed every time a file is saved; such as trimming trailing whitespace from all lines or changing all newline characters to either LF or CRLF according to the settings specified in the editorconfig profile for the file being edited.

[1] Furthermore, I also took over maintenance of one more plugin, vis-editorconfig, from its original developer.

vis-eval https://github.com/seifferth/vis-eval

The vis-eval plugin allows the evaluation of arbitrary markdown code blocks in any file being edited with vis. This simple addition effectively allows to turn vis from a simple text editor into a basic notebook computing environment. In essence, this plugin will simply search for the closest code block akin to the following one located anywhere above the primary cursor

````
```python
print("Hello World!")
```
````

Upon pressing 'g<Enter>', this block will be expanded with the output produced by the specified command, which would turn the above codeblock into something like this

````
```python
print("Hello World!")
```
::: {.output exit_code="0"}
    Hello World!
:::
````

When compared to more established notebook computing environments such as jupyter, vis-eval is both very basic and very simple. While jupyter automagically propagates the state of variables between cells, for instance, vis-eval simply invokes standard shell commands and passes them the contents of the code block to be evaluated on stdin. If any state is to be propagated between code blocks with vis-eval, the programmer would usually need to manually add the code for storing this state to or for loading it from disk. While this might seem like a limitation in some cases, it also greatly simplifies mixing code blocks written in different programming languages. And when combined with interpreter commands that implicitly persist state to disk by themselves — such as '`sqlite3 some.db`', for example — the vis-eval plugin even provides an incredibly powerful notebook computing experience out of the box.

vis-super-shellout
https://github.com/seifferth/vis-super-shellout

The vis-super-shellout plugin provides a slightly different version of the built-in ':<' command. Both the built-in ':<' command and the ':R' command provided by vis-super-shellout allow adding the output of arbitrary shell commands to the file opened in vis. The built-in version of that command does not release stdout, however, which makes it impossible to run interactive commands like fzf (https://github.com/junegunn/fzf/) or cite (https://tilde.club/~seifferth/#software:cite). The ':R' command, in contrast, allows to run these programs without issue.

Furthermore, the ':R' command also strips a single final newline from the output returned by the shell command if present. In my experience, this makes it a little more convenient to use common shell commands such as 'date', which often return a single line terminated by a single newline character.

vis-todo
https://github.com/seifferth/vis-todo

The vis-todo plugin adds a very simple function to jump to the next occurrence of the uppercase string 'TODO' when pressing 'gt' in normal mode. This can come in handy if one wants to mark specific lines in a source file in order to quickly return to them later on.

Colophon

The things I link to from this website are usually made available under free copyleft licenses. For software repositories, I usually choose the GNU General Public License version 3 (GPLv3), while for the more literary parts, I commonly use the Creative Commons Attribution-ShareAlike License version 4.0 (CC BY-SA 4.0). Also note that CC BY-SA 4.0 explicitly allows to change the license of derivative works to GPLv3, so if you want to incorporate code snippets from a CC BY-SA 4.0 licensed blog post into a GPLv3 licensed piece of software, for example, you are very much allowed to do so.

If I should have forgotten to add a license to any of the files or repositories linked to from this site, feel free to drop me a note. If you wish to redistribute this index file itself, you are also very welcome to do so under the terms of the CC BY-SA 4.0 license.