TiefDownConverter Documentation

Tiefseetauchner et al

March 12, 2025

Contents

1. LICENSE

```
MIT License

Copyright (c) 2025 Lena Tauchner

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

2. The what?

If you want to skip the funny written explanations, skip to the Usage section.

Well, that's a good question. Ask it later.

Jk, of course you may ask it now. TiefDown is a project format I made up to make it easier to convert my markdown files into something pretty. As a matter of fact, this documentation is managed by a TiefDown project!

The important thing is that this isn't a markdown parser, replacement or anything like that. It's a project format, and it's not even a format, it's pretty much just a manifest file and an executable.

2.1. Why?

I wonder myself every day. But alas, I should know, I wrote this cluster**** so let me explain. The initial concept was born from pain (as many are). I was pretty tired of exporting my markdown files, then converting them, overwriting my old files, then converting them again, overwriting all history in the process. It was just… a mess.

So I did what any sane person would do: I learned Python.

Well, I'm being facetious. I didn't "learn Python," I just expanded my capabilities to calling programs from the command line.

So my script, at first, just called Pandoc, then pdflatex, and then pdflatex again for good measure. It created a PDF, overwriting my old one. It was basically just converting a single markdown file into a PDF with a basic TeX template (in my case, LiX Novel).

Then I realized that writing a 40-chapter story in a single markdown file was even dumber than whatever I made in Python. So I added a little combination logic. In the process, I had to write Lua filters as well, and then I added versioning, and then I added conversion to multiple different PDFs, and then I added EPUB support and—you know what? That was a dumb idea. The Python script soon reached 200 lines of code, which was untenable.

So yeah, I decided to make a new book. And of course - ***everything*** broke. Instantly. I had to copy and paste things, adjust my Python script, rewrote it a bit, and boom - suddenly I had two different projects with different processes, different outputs, different versions, different everything.

And then… I started a third book. Aaaand the Python script didn't really fulfill my needs, so I rewrote it in Bash. But worse.

I thought I had it all figured out. With Python. Then Bash. Then I started a short story and lost my ******* mind.

2.2. How, oh wise programmer, did you solve this problem?

I'm glad you asked! I'm glad. I… I hope you asked? Well, regardless of whether or not you did, I'll tell you.

I learned Rust.

For real this time, I learned a completely new programming language just for this. But there was a reason, or a few rather:

1. I wanted cross-platform support.
2. I wanted a single executable.
3. I needed a language with good CLI support because, believe it or not, I'm *awful* at GUIs.
4. I'm crazy.

These reasons led me to two options: Python, a language I was somewhat familiar with but didn't particularly enjoy writing in, and Rust, a language I had never written in before but was very interested in.

Evidently, I chose Rust.

So I started: a CLI interface, command-line calls, and so on. Here's the rundown of how it works internally:

- You initialize a project with `tiefdownconverter init`. This creates a few bits and bobs, but importantly the `manifest.toml` file. This contains all the information needed to actually manage and convert the project.
- You can then manipulate the project, and so on.
- When you add your markdown files to the Markdown directory, running `tiefdownconverter convert` will do a few things:
  - Create a new folder for the current compilation. That way, you have a history.
  - Combine all the markdown files into one megafile called `combined.md`.
  - Run Pandoc conversion to TeX, EPUB, or Typst. This uses Lua filters that are defined in the `manifest.toml` file.
  - Run XeLaTeX on all TeX templates, Typst on all Typst templates, and so on. It even supports EPUB conversion.
  - Copy the files around so you end up with your output files in the right places.

Isn't that simple?

It isn't. But oh well. We've got a lot of work to do on this, and if you're interested, don't shy away from the Contributing section!

2.3. So, what's the point?

Really? Making my life easier. I wanted to export my novel as a PDF in A4, 8x5in, so on. If I can make your life easier as well, then I'm the happiest woman alive.

2.4. Use Cases

So where does TiefDownConverter actually come in handy? Well, anywhere you need Markdown to turn into something nice without manually fiddling with formats every time. Here are a few scenarios where it saves the day:

- **Writing Books** - Markdown is great for writing, but formatting a 300-page novel? Not so much. TiefDown handles that for you. Well, at least the part where you need to convert stuff, you still need to write out your templates.

- **Technical Documentation** - Software projects need structured documentation, and TiefDown makes sure it's consistent. Case and point, this documentation is managed as a TiefDown project.
- **Multi-format Exports** - Need a A4 PDF, a Book PDF, a letter PDF, EPUB, so on? TiefDown can generate them all from the same source.

Basically, if your workflow involves Markdown and you're sick of manually converting everything, TiefDown is your new best friend.

3. Usage

The basic usage of `tiefdownconverter` is relatively simple. The difficult part is understanding the templating system and how to customise it for your usecases. Presets can only do so much.

3.1. Installation

Currently the only way to install `tiefdownconverter` is to either build it yourself or download a precompiled binary from the releases page°. Then just add it to the path and you're good to go. You can of course also just call it relatively by placing the binary in your project folder or something like that.

There are a few dependencies that you need to install.

- Pandoc°: Conversion from Markdown to TeX, Typst and Epub.
- A TeX distribution: For converting TeX files to PDF. It has to include xelatex.
  - If using TeX Live° you may need to additionally install `texlive-xetex` depending on your system.
  - If using MikTeX°, no need to do anything.
- Typst°: For converting Typst files to PDF.

Now you should be able to run `tiefdownconverter` from the command line. You can test it by initialising a test project using `tiefdownconverter init testproject` and running `tiefdownconverter convert` in the project directory or `tiefdownconverter convert -p testproject`.

3.2. Getting started

TL;DR: Make a folder, go into it and run `tiefdownconverter init` and `tiefdownconverter convert`. That's it.

Long anser: First off, you need to create a project using `tiefdownconverter init`. This will create a new project **in the current directory**. You can (and maybe should) specify a project using the -p flag.

This command creates the basic template structure like so:

```
your_project/
├── Markdown/
│   └── Chapter 1 - Introduction.md
├── template/
│   ├── meta.tex
│   └── template.tex
└── manifest.toml
```

The Markdown folder contains an example Markdown file. When placing your markdown files in this folder, make sure they're named like `Chapter X.md`, with anything following the number being ignored. *This is important*, as the converter will use this to sort the files for conversion, as otherwise it'd have no idea in which order they should be converted.

Now you should be able to run `tiefdownconverter convert -p path/to/your_project` (or ommitting the -p flag if you're already in the project directory) and it should generate a PDF file in the project directory. You can now adjust the template, add your own Markdown files, and so on.

3.3. Adjusting the markdown directory

You can change what directory the converter looks for markdown files in by changing the `markdown_dir` field in the manifest.toml file or saying `-m path/to/markdown/dir` when initialising the project. You can also change it post-initialisation using `tiefdownconverter project update-manifest -m path/to/markdown/dir`. If you don't do so, the converter will look for markdown files in the `project_dir/Markdown` directory.

3.4. Customising the template

The key idea behind tiefdownconverter is, that it can handle multiple templates at the same time. This is done by creating a template file in the template directory and adding it to the project's manifest.toml file.

You could do this manually, if you were so inclined, but using `tiefdownconverter project add-template` is much easier. Check the Usage Details for the usage of this command. But importantly, once you created the template and added it to the manifest, you will be able to convert using it. `tiefdownconverter convert -p path/to/your_project --templates <TEMPLATE_NAME>` will convert only the selected template, aiding in debugging.

And now, you're pretty much free to do whatever you want with the template. Write tex or typst templates, use custom filters, so on.

3.5. Adjusting template behaviour

You have a few options for editing template behaviour using `tiefdownconverter`. You can of course edit the template files directly, but there are a few more options.

Mainly and most interestingly, lua filters can adjust the behaviour of the markdown conversion. These are lua scripts that are run before the markdown is converted to tex or typst. You can add lua filters to a template by either editing the manifest or using `tiefdownconverter project update-template <TEMPLATE_NAME> --add-filters <FILTER_NAME>`. This can be either the path to a lua filter (relative to the project directory) or a directory containing lua filters.

You can also change the name of the exported file by setting the `output` option. For example, `tiefdownconverter project update-template <TEMPLATE_NAME> --output <NEW_NAME>`. This will export the template to `<NEW_NAME>.pdf` instead of the default `<TEMPLATE_NAME>.pdf`.

Similarly, you could change the template file and type, though I advice against it, as this may break the template. I advice to just add a new template and remove the old one using `tiefdownconverter project remove-template <TEMPLATE_NAME>`.

3.6. Writing templates

Importantly, when you write your own template, you need to include the content somehow. That somehow is done via `\input{output.tex}` or `#include "./output.typ"`. This will include the output of the Markdown conversion in your template file. In the future, it may be possible to specify per template, what the output should be called, but for now, it's just `output.tex`.

3.7. Writing filters

Note: This section only really addresses LaTeX, but the concepts are the same for Typst.

If you are in the business of writing filters (and don't just solve everything in TeX itself), I advice checking out the documentation at https://pandoc.org/lua-filters.html°. But here's a quick rundown of what you can do. For example, if you wanted to change the font of all block quotes, there's a few things you'd need to do. First off, in your template, you will need to define a font. It could look something like this:

```tex
\usepackage{fontspec}
\newfontfamily\blockquotefont{Noto Sans}
```

Then, add a filter to your template as described above. The filter could look something like this:

```lua
function BlockQuote(el)
  local tt_start = pandoc.RawBlock('latex', '\\blockquotefont\\small')

  table.insert(el.content, 1, tt_start)

  return el
end
```

Of course, you could just redefine the font in TeX but I think this is a bit more flexible. One usecase that is quite important is to change the way chapters are handled for LiX. In case of LiX, they expect `\h{chapter_name}` instead of `\section`, which is the standard behaviour of pandoc. So when you create a LiX backed template, you have to add a filter to change that behaviour. Something like this:

```lua
function Header(elem)
  if elem.level == 1 then
    return pandoc.RawBlock("latex", "\\h{" .. pandoc.utils.stringify(elem.content) .. "}")
  end
  if elem.level == 2 then
    return pandoc.RawBlock("latex", "\\hh{" .. pandoc.utils.stringify(elem.content) .. "}")
  end
  -- add more levels here if needed
end
```

4. Usage Details

Below are the usage details for the various commands. **Note:** These are autogenerated! For clearer documentation, please see the Usage section.

4.1. `tiefdownconverter`

Version: `tiefdownconverter 0.3.0`

4.1.1. Usage:

```
A CLI tool for managing TiefDown Projects

Usage: tiefdownconverter <COMMAND>

Commands:
  convert  Convert a TiefDown project. By default, it will convert the
    current directory.
  init     Initialize a new TiefDown project.
  project  Update the TiefDown project.
  help     Print this message or the help of the given subcommand(s)

Options:
  -h, --help     Print help
  -V, --version  Print version
```

4.1.2. Subcommands:

- convert
- init
- project

4.2. `tiefdownconverter convert`

Version: `tiefdownconverter 0.3.0`

4.2.1. Usage:

```
Convert a TiefDown project. By default, it will convert the current
    directory.

Usage: tiefdownconverter convert [OPTIONS]

Options:
  -p, --project <PROJECT>         The project to convert. If not provided,
    the current directory will be used.
  -t, --templates <TEMPLATES>...  The templates to use. If not provided,
    the default templates from the manifest file will be used.
  -h, --help                      Print help
```

4.3. `tiefdownconverter init`

Version: `tiefdownconverter 0.3.0`

4.3.1. Usage:

```
Initialize a new TiefDown project.

Usage: tiefdownconverter init [OPTIONS] [PROJECT]

Arguments:
  [PROJECT]  The project to initialize. If not provided, the current
    directory will be used.

Options:
  -t, --templates <TEMPLATES>...     The preset templates to use. If not
    provided, the default template.tex will be used.
                                     For custom templates, use the update
    command after initializing the project.
                                     If using a LiX template, make sure to
    install the corresponding .sty and .cls files from
    https://github.com/NicklasVraa/LiX. Adjust the metadata in
    template/meta.tex accordingly.
                                      [possible values: template.tex,
    booklet.tex, lix_novel_a4.tex, lix_novel_book.tex, template_typ.typ,
    default_epub]
  -n, --no-templates                 Do not include the default templates.
    You will need to add templates manually with Update
  -f, --force                        Delete the project if it already
    exists.
  -m, --markdown-dir <MARKDOWN_DIR>  The directory where the Markdown files
    are located. If not provided, Markdown/ will be used.
  -h, --help                         Print help
```

4.4. `tiefdownconverter project`

Version: `tiefdownconverter 0.3.0`

4.4.1. Usage:

```
Update the TiefDown project.

Usage: tiefdownconverter project [PROJECT] <COMMAND>

Commands:
  add-template     Add a new template to the project.
  remove-template  Remove a template from the project.
  update-template  Update a template in the project.
  update-manifest  Update the project manifest.
  list-templates   List the templates in the project.
  validate         Validate the TiefDown project structure and metadata.
  clean            Clean temporary files from the TiefDown project.
  help             Print this message or the help of the given
    subcommand(s)

Arguments:
  [PROJECT]  The project to edit. If not provided, the current directory
    will be used.

Options:
  -h, --help  Print help
```

4.4.2. Subcommands:

- add-template
- remove-template
- update-template
- update-manifest
- list-templates
- validate
- clean

4.5. `tiefdownconverter project add-template`

Version: `tiefdownconverter 0.3.0`

4.5.1. Usage:

```
Add a new template to the project.

Usage: tiefdownconverter project add-template <TEMPLATE> [TEMPLATE_FILE]
    [TEMPLATE_TYPE] [OUTPUT] [FILTERS]...

Arguments:
  <TEMPLATE>       The name of the template to create. If using a LiX
    template, make sure to install the corresponding .sty and .cls files
    from https://github.com/NicklasVraa/LiX. Adjust the metadata in
    template/meta.tex accordingly.
  [TEMPLATE_FILE]  The file to use as the template. If not provided, the
    template name will be used.
  [TEMPLATE_TYPE]  The type of the template. If not provided, the type will
    be inferred from the template file. [possible values: tex, typst, epub]
  [OUTPUT]         The output file. If not provided, the template name will
    be used.
  [FILTERS]...     The luafilters to use for pandoc conversion of this
    templates markdown.

Options:
  -h, --help  Print help
```

4.6. `tiefdownconverter project remove-template`

Version: `tiefdownconverter 0.3.0`

4.6.1. Usage:

```
Remove a template from the project.

Usage: tiefdownconverter project remove-template --template <TEMPLATE>

Options:
  -t, --template <TEMPLATE>  The template to remove.
  -h, --help                 Print help
```

4.7. `tiefdownconverter project update-template`

Version: `tiefdownconverter 0.3.0`

4.7.1. Usage:

```
Update a template in the project.

Usage: tiefdownconverter project update-template [OPTIONS] <TEMPLATE>

Arguments:
  <TEMPLATE>  The template to update.

Options:
      --template-file <TEMPLATE_FILE>
          The file to use as the template. If not provided, the template
    name will be used.
      --template-type <TEMPLATE_TYPE>
          The type of the template. If not provided, the type will be
    inferred from the template file.
          Changing this is not recommended, as it is highly unlikely the
    type and only the type has changed. It is recommended to create a new
    template instead. [possible values: tex, typst, epub]
      --output <OUTPUT>
          The output file. If not provided, the template name will be used.
      --filters <FILTERS>...
          The luafilters to use for pandoc conversion of this templates
    markdown.
      --add-filters <ADD_FILTERS>...
          The luafilters add to the template.
      --remove-filters <REMOVE_FILTERS>...
          The luafilters to remove from the template.
  -h, --help
          Print help
```

4.8. `tiefdownconverter project update-manifest`

Version: `tiefdownconverter 0.3.0`

4.8.1. Usage:

```
Update the project manifest.

Usage: tiefdownconverter project update-manifest [OPTIONS] [PROJECT]

Arguments:
  [PROJECT]  The project to manipulate. If not provided, the current
    directory will be used.

Options:
  -m, --markdown-dir <MARKDOWN_DIR>  The directory where the Markdown files
    are located.
  -h, --help                         Print help
```

4.9. `tiefdownconverter project list-templates`

Version: `tiefdownconverter 0.3.0`

4.9.1. Usage:

```
List the templates in the project.

Usage: tiefdownconverter project list-templates [PROJECT]

Arguments:
  [PROJECT]  The project to manipulate. If not provided, the current
    directory will be used.

Options:
  -h, --help  Print help
```

4.10. `tiefdownconverter project validate`

Version: `tiefdownconverter 0.3.0`

4.10.1. Usage:

```
Validate the TiefDown project structure and metadata.

Usage: tiefdownconverter project validate [PROJECT]

Arguments:
  [PROJECT]  The project to validate. If not provided, the current
    directory will be used.

Options:
  -h, --help  Print help
```

4.11. `tiefdownconverter project clean`

Version: `tiefdownconverter 0.3.0`

4.11.1. Usage:

```
Clean temporary files from the TiefDown project.

Usage: tiefdownconverter project clean [PROJECT]

Arguments:
  [PROJECT]  The project to clean. If not provided, the current directory
    will be used.

Options:
  -h, --help  Print help
```

5. Contributing

This project is open source, and I'd love for you to contribute! There's a few things you should know before you start.

5.1. Conversion

Conversion is split in a few different steps:

1. Combine all the markdown files into one megafile called `combined.md`.
2. Run Pandoc conversion to TeX, EPUB, or Typst. This uses Lua filters that are defined in the `manifest.toml` file.
3. Run XeLaTeX on all TeX templates, Typst on all Typst templates, and so on.

Say you were to add a new conversion type. In `converters.rs`, you'd need to add a new function that handles the full conversion. Including handling lua filters, markdown conversion, so on. This converter function has to then be included in our conversion decision logic in `conversion_decider.rs`. And for that you need to add a new TemplateType, which includes editing the implementations. Then you need to add the new template type decision logic to `get_template_type_from_path`.

5.2. Presets

> **NOTE:** This is a bit of a niche usecase, so documentation is lacking. You can always ask for help on this in a GitHub issue.

You can also add new presets, but that's a bit more involved. You should check the implementation for the existing presets, I don't think it's useful to document this nieche usecase for now.

5.3. Manifest

Hope you don't have to change the manifest.toml file. If you do, change the manifest model, increase the version number in `consts.rs` and add a upgrade logic to `manifest_model.rs`.

5.4. Tests

Practically non-existant but I would be happy if you added some.

I'm using `rstest` for tests, and you can see an example in `manifest_model.rs`. Just make sure to somewhat follow that example.

5.5. Documentation

Make sure this documentation is up to date. If you add a command or flag, make sure to run `tools/generate_docs.py` as well as `tiefdownconverter convert -p docs`. Then commit the changes.

5.6. Code Style

I don't have one. I'm sorry.