# TiefDownConverter Documentation

Tiefseetauchner et al.

March 27, 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 and preprocessors 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 in 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.

## 2.5. Support

Now, you want support? Check out the Discord or write an issue on GitHub!

- Discord Server (https://discord.gg/EG3zU9cTFx)
- GitHub Issues (https://github.com/Tiefseetauchner/TiefDownConverter/issues)

# 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, install it from cargo, 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.

If you build from source, run `cargo build [--release]` or `cargo install --path .`.

That said, the recommended way to install TiefDownConverter is `cargo install tiefdownconverter`. This will always install the latest version.

Downloading from the release is as simple as downloading the appropriate version (Windows, Mac, Linux) and adding it to a folder in the path. You could also add tiefdownconverter to a folder and run it from there.

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.

Windows is easy enough: `winget install miktex pandoc typst`.

Linux varies by distro of course, but for ubuntu it's `apt install texlive-xetex pandoc` and `cargo install typst` or downloading the typst binary and adding it to the path.

Mac is still to be tested, but MacTex should have XeTeX installed.

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`. You could also, as a test, clone the Github Repo° and run `tiefdownconverter convert -p docs` (this may however throw warnings if you don't have the appropriate fonts installed).

## 3.2. Getting started

TL;DR: Make a folder, go into it and run `tiefdownconverter init` [1] and `tiefdownconverter convert` [2]. That's it.

Long anser: First off, you need to create a project using `tiefdownconverter init` [1]. This will create a new project **in the current directory**. You can (and maybe should) specify a project.

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[3] 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` [2] (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. The markdown directory

Markdown files are the main input for the converter, and as such their structure is important. The converter will look for markdown files in the `Markdown` directory, and will sort them by a chapter number. Namely, your files should be named `Chapter X Whatever else.md`, where X is a number (you don't have to name them 01, 02 etc., as we parse the number as an integer anyways). The converter will then sort them by the number and combine them in that order.

You can also add subdirectories in the Markdown directory. These will be combined after the file with the same number. For example, consider the following directory structure:

```
Markdown/
├── Chapter 1 - Introduction.md
├── Chapter 2 - Usage.md
├── Chapter 2 - Usage/
│   ├── Chapter 1 - Usage detail 1.md
│   └── Chapter 2 - Usage detail 2.md
└── Chapter 3 - Customisation.md
```

The converter will combine the files in the following order:

1. Chapter 1 - Introduction.md
2. Chapter 2 - Usage.md
3. Chapter 2 - Usage/Chapter 1 - Usage detail 1.md
4. Chapter 2 - Usage/Chapter 2 - Usage detail 2.md
5. Chapter 3 - Customisation.md

That is, the converter orders a directory by the same logic as other files (and even does so recursively), and directories are combined after the file with the same number.

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[1]. You can also change it post-initialisation using

`tiefdownconverter project update-manifest -m path/to/markdown/dir` [4]. 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` [5] 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>` [2] 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>` [6]. 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>` [6]. This will export the template to `<NEW_NAME>` 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>` [7].

## 3.6. Conversion Profiles

A conversion profile is a shortcut to defining templates for the conversion. If you're dealing with a lot of templates, you may be considering only converting some at any time - for example, web ready PDFs vs. print ready PDFs, or only converting a certain size of PDF.

For that, there are conversion profiles which simply are a list of templates. It's essentially like saving your –templates[2] arguments.

You can create these profiles with the `project add-profile` [8] command, setting a name and a comma seperated list of templates. Removing a profile is also possible with the `project remove-profile` [9] command.

Running a conversion with a profile is as simple as adding the `--profile` flag[2].

The manifest file can optionally contain a section for this, if you desire to configure them manually:

```toml
[[profiles]]
name = "PDF"
templates = ["PDF Documentation LaTeX", "PDF Documentation"]
```

## 3.7. 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. If you're using custom preprocessors, you can change the output file of the conversion. See Preprocessing for more information.

## 3.8. Epub Support

EPUB support in TiefDownConverter isn't as fancy as LaTeX or Typst, but you can still tweak it to look nice. You don't get full-blown templates, but you can mess with CSS, fonts, and Lua filters to make it work how you want.

### 3.8.1. Customizing CSS

EPUBs use stylesheets to control how everything looks. The good news? Any `.css` file you drop into `template/my_epub_template/` gets automatically loaded. No need to mess with the manifest - just throw in your styles and you're good.

Example CSS:

```css
body {
    font-family: "Noto Serif", serif;
    line-height: 1.6;
    margin: 1em;
}
blockquote {
    font-style: italic;
    border-left: 3px solid #ccc;
    padding-left: 10px;
}
```

### 3.8.2. Adding Fonts

Fonts go into `template/my_epub_template/fonts/`, and TiefDownConverter will automatically pick them up. To use them, you just need to reference them properly in your CSS:

```css
@font-face {
  font-family: 'EB Garamond';
  font-style: normal;
  font-weight: normal;
  src: url('../fonts/EBGaramond-Regular.ttf');
```

```
}

body {
    font-family: "EB Garamond", serif;
}
```

### 3.8.3. Metadata and Structure

EPUBs need some basic metadata, which you define in the YAML front matter of your Markdown files. Stuff like title, author, and language goes here:

```yaml
---
title:
- type: main
  text: "My Publication"
- type: subtitle
  text: "A tale of loss and partying hard"
creator:
- role: author
  text: Your Name
rights: "Copyright © 2012 Your Name"
---
```

This makes sure your EPUB doesn't look like a nameless file when opened in an e-reader.

### 3.8.4. Using Lua Filters

Want to tweak the structure? That's what Lua filters are for. You can use them to rename chapters, remove junk, or modify how elements are processed.

Example: Automatically renaming chapter headers:

```lua
function Header(el)
  if el.level == 1 then
    return pandoc.Header(el.level, "Chapter: " .. pandoc.utils.stringify(el.content))
  end
end
```

And that's it. You get a customized EPUB without having to fight with the defaults. Enjoy!

## 3.9. Conversion Engines

There are currently four ways to convert your Markdown files. All of them are based on the same system. The main difference is the output format and the program it gets converted with.

### 3.9.1. LaTeX

LaTeX is the best supported by TiefDownConverter, with the most presets. But as TiefDownConverter is a general-purpose Markdown to PDF converter, the format doesn't matter. LaTeX provides the highest degree of customization, making it ideal for structured documents, novels, and academic papers.

The primary way to interact with LaTeX is through templates. Lua filters and such are secondary, but an important part of the conversion process to adjust behavior for different document classes.

#### 3.9.2. Typst

Typst is another supported engine, offering a more modern alternative to LaTeX with a simpler syntax and automatic layout adjustments. TiefDownConverter allows you to specify Typst templates in the project manifest.

Typst templates work similarly to LaTeX templates but are easier to modify if you need structured documents without deep LaTeX knowledge.

As far as I could tell, typst templates are also far more adherent to the general typst syntax, so Lua filters are not as important. But they can still be used to adjust the output, especially for more advanced use cases.

#### 3.9.3. EPUB

TiefDownConverter also supports EPUB conversion, making it suitable for e-book generation. The conversion process uses Pandoc to transform the Markdown content into EPUB, applying any Lua filters defined in the manifest.

This however does not really support much in the way of templating. Customization should be done primarily via Lua filters. Custom preprocessors are currently not supported at all.

However, you can still get some customization by including CSS and font files in your template folder. That's the reason epub has to have a folder in the first place, so you can place CSS and font files in there. Of course you can add multiple epub templates, but I don't know why you would want to.

EPUB output is particularly useful for digital publishing, ensuring compatibility with e-readers and mobile devices.

#### 3.9.4. Custom Pandoc Converter

Okay. Stick with me here. The idea is, you are already converting my Markdown files with Pandoc, why not let me convert them to whatever format? Well, this is where Custom Pandoc Conversion comes in. This long fabled feature is the most complicated one, and you need a deep understanding of how TiefDownConverter works and at least the ability to read Pandoc's documentation to even use it. But if you're willing to put in the effort, you can do some pretty cool things.

The basic idea is, just, let the user decide what pandoc does. The result is chaos.

I'm being facetious, but this is actually the most powerful way to customize the output. You add a preprocessor as described in Preprocessing and set the output path of the preprocessor and template to the same path. Then you can do whatever pandoc allows. Want to convert to RTF? No issue. But beware: you need to actually understand what's going on, otherwise you'll end up in implementation hell.

## 3.10. Writing filters

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

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
```

## 3.11. Preprocessing

A "Preprocessor" is a stupid word for defining your own pandoc conversion parameters. You can (and should) use this to adjust the behaviour of the converter. For example, you could define a preprocessor to add `--listings` to the pandoc command. This is useful if you want to have reasonable code output in your pdf.

If no preprocessor is defined, the converter will use default pandoc parameters, converting to the intermediate output file (in case of LaTeX, this is `output.tex`). But if you for example are

using lua filters, you may want to export to a different path. This can be done by defining a preprocessor.

If you want to define a preprocessor, you can do so by running

```bash
tiefdownconverter project update-template <TEMPLATE_NAME> --preprocessor <PREPROCESSOR_NAME>
```

[9] to assign it to a template and

```bash
tiefdownconverter project add-preprocessor <PREPROCESSOR_NAME> -- [PANDOC_ARGS]
```

[10] to assign it to a template and

to create a new preprocessor.

For example, if you want to add `--listings` to the pandoc command, you could do so by adding `--listings` to the preprocessor. But importantly, **this overwrites the default preprocessor**. So you will have to add the `-o output.tex` argument to the preprocessor as well. The full command then would be:

```bash
tiefdownconverter project add-preprocessor "Enable Listings" -- -o output.tex --listings
```

[10]

The manifest would look something like this:

```toml
...

[[custom_processors.preprocessors]]
name = "Enable Listings"
pandoc_args = ["-o", "output.tex", "--listings"]

[[templates]]
filters = ["luafilters/chapter_filter.lua"]
name = "PDF Documentation LaTeX"
output = "docs_tex.pdf"
preprocessor = "Enable Listings"
template_file = "docs.tex"
template_type = "Tex"

...
```

## 3.12. Custom Pandoc Conversion

I already hinted at it in Custom Pandoc Converter, but I'll go into more detail here. The idea is to run a preprocessor and just skip any further processing. Straight from pandoc to the output.

You can do this by first defining a preprocessor, for example:

```bash
```

```
tiefdownconverter project add-preprocessor "RTF Preprocessor" -- -o documentation.rtf
```

[10]

As you can see, we're outputting as an RTF file, and the file name is `documentation.rtf`. This means we need to add a template that deals with the same output:

bash

```
tiefdownconverter project add-template "RTF Template" -o documentation.rtf -t custompandoc
```

[10]

And that's it. TiefDownConverter will run the preprocessor, which outputs to documentation.rtf, and then the templating system will copy that output to your directory. Hopefully. Did I mention that this is experimental? Yeah, so if you have issues, please report them. Even if you're thinking "this is not a bug, it's a feature". It likely isn't.

## 3.13. Smart Cleaning

Smart cleaning is a feature that is relatively simple. If you enable it in your manifest, it will automatically remove stale or old conversion directories.

Enable it with the `--smart-clean` and set the threshold with `--smart-clean-threshold`. The threshold is 5 by default. [1] [4]

You can also manually trigger a smart clean with `tiefdownconverter project smart-clean` [11] or a normal clean with `tiefdownconverter project clean` [12]. The latter will remove all conversion directories, while the former will only remove the ones that are older than the threshold.

# 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 0.7.0-alpha`

### 4.1.1. Usage:

```
TiefDownConverter manages TiefDown projects.
TiefDown is a project structure meant to simplify the conversion process from Markdown to PDFs.
TiefDownConverter consolidates multiple conversion processes and templating systems to generate a configurable set or subset of output documents.
It is not in itself a converter, but a wrapper around pandoc, xelatex and typst. As such, it requires these dependencies to be installed.

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.
  check-dependencies  Validate dependencies are installed.
  help                Print this message or the help of the given subcommand(s)

Options:
  -h, --help
          Print help (see a summary with '-h')

  -V, --version
          Print version
```

### 4.1.2. Subcommands:

- convert
- init
- project
- check-dependencies

## 4.2. `tiefdownconverter convert`

**Version:** `tiefdownconverter 0.7.0-alpha`

### 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. Cannot be used with --profile.
  -P, --profile <PROFILE>         The conversion profile to use. Cannot be used with
--templates.
  -h, --help                      Print help
```

## 4.3. `tiefdownconverter init`

**Version:** `tiefdownconverter 0.7.0-alpha`

### 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.

      --smart-clean
          Enables smart clean for the project with a default threshold of 5.
          If the number of conversion folders in the project is above this threshold, old
folders will be cleaned, leaving only the threshold amount of folders.

      --smart-clean-threshold <SMART_CLEAN_THRESHOLD>
          The threshold for smart clean. If not provided, the default threshold of 5 will
be used.
          If the number of conversion folders in the project is above this threshold, old
folders will be cleaned, leaving only the threshold amount of folders.
```

```
34
35   -h, --help
36           Print help (see a summary with '-h')
```

## 4.4. `tiefdownconverter project`

**Version:** `tiefdownconverter 0.7.0-alpha`

### 4.4.1. Usage:

```
 1 Update the TiefDown project.
 2
 3 Usage: tiefdownconverter project [PROJECT] <COMMAND>
 4
 5 Commands:
 6   add-template         Add a new template to the project.
 7   remove-template      Remove a template from the project.
 8   update-template      Update a template in the project.
 9   update-manifest      Update the project manifest.
10   add-preprocessor     Add a new preprocessor to the project.
11   remove-preprocessor  Remove a preprocessor from the project.
12   add-profile          Add a new conversion profile to the project.
13   remove-profile       Remove a conversion profile from the project.
14   list-templates       List the templates in the project.
15   list-profiles        List the conversion profiles in the project.
16   list-preprocessors   List the preprocessors in the project.
17   validate             Validate the TiefDown project structure and metadata.
18   clean                Clean temporary files from the TiefDown project.
19   smart-clean          Clean temporary files from the TiefDown project, leaving only the
   threshold amount of folders.
20   help                 Print this message or the help of the given subcommand(s)
21
22 Arguments:
23   [PROJECT]  The project to edit. If not provided, the current directory will be used.
24
25 Options:
26   -h, --help  Print help
```

### 4.4.2. Subcommands:

- add-template
- remove-template
- update-template
- update-manifest
- add-preprocessor
- remove-preprocessor
- add-profile
- remove-profile
- list-templates
- list-profiles
- list-preprocessors
- validate
- clean

- smart-clean

## 4.5. `tiefdownconverter project add-template`

**Version:** `tiefdownconverter 0.7.0-alpha`

### 4.5.1. Usage:

```
Add a new template to the project.

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

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.

Options:
  -f, --template-file <TEMPLATE_FILE>  The file to use as the template. If not provided, the template name will be used.
  -t, --template-type <TEMPLATE_TYPE>  The type of the template. If not provided, the type will be inferred from the template file. [possible values: tex, typst, epub, custom-pandoc]
  -o, --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.
      --preprocessor <PREPROCESSOR>    The preprocessor to use for this template.
  -h, --help                           Print help
```

## 4.6. `tiefdownconverter project remove-template`

**Version:** `tiefdownconverter 0.7.0-alpha`

### 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 0.7.0-alpha`

### 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, custom-pandoc]
      --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.
      --preprocessor <PREPROCESSOR>
          The preprocessor to use for this template.
  -h, --help
          Print help
```

## 4.8. `tiefdownconverter project update-manifest`

**Version:** `tiefdownconverter 0.7.0-alpha`

### 4.8.1. Usage:

```
Update the project manifest.

Usage: tiefdownconverter project update-manifest [OPTIONS]

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

      --smart-clean <SMART_CLEAN>
          Enables smart clean for the project with a default threshold of 5.
          If the number of conversion folders in the project is above this threshold, old folders will be cleaned, leaving only the threshold amount of folders.

          [possible values: true, false]

      --smart-clean-threshold <SMART_CLEAN_THRESHOLD>
          The threshold for smart clean. If not provided, the default threshold of 5 will be used.
          If the number of conversion folders in the project is above this threshold, old folders will be cleaned, leaving only the threshold amount of folders.

  -h, --help
          Print help (see a summary with '-h')
```

## 4.9. `tiefdownconverter project add-preprocessor`

**Version:** `tiefdownconverter 0.7.0-alpha`

#### 4.9.1. Usage:

```
Add a new preprocessor to the project.

Usage: tiefdownconverter project add-preprocessor <NAME> [-- <PANDOC_ARGS>...]

Arguments:
  <NAME>            The name of the preprocessor to create.
  [PANDOC_ARGS]...  The arguments to pass to the preprocessor.

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

### 4.10. `tiefdownconverter project remove-preprocessor`

**Version:** `tiefdownconverter 0.7.0-alpha`

#### 4.10.1. Usage:

```
Remove a preprocessor from the project.

Usage: tiefdownconverter project remove-preprocessor <NAME>

Arguments:
  <NAME>  The name of the preprocessor to remove.

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

### 4.11. `tiefdownconverter project add-profile`

**Version:** `tiefdownconverter 0.7.0-alpha`

#### 4.11.1. Usage:

```
Add a new conversion profile to the project. These profiles contain a list of templates to
preset conversion workflows.

Usage: tiefdownconverter project add-profile <NAME> [TEMPLATES]...

Arguments:
  <NAME>
          The name of the profile to create.

  [TEMPLATES]...
          The templates to add to the profile.

Options:
  -h, --help
          Print help (see a summary with '-h')
```

### 4.12. `tiefdownconverter project remove-profile`

**Version:** `tiefdownconverter 0.7.0-alpha`

#### 4.12.1. Usage:

```
Remove a conversion profile from the project.

Usage: tiefdownconverter project remove-profile <NAME>

Arguments:
  <NAME>  The name of the profile to remove.

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

### 4.13. `tiefdownconverter project list-templates`

**Version:** `tiefdownconverter 0.7.0-alpha`

#### 4.13.1. Usage:

```
List the templates in the project.

Usage: tiefdownconverter project list-templates

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

### 4.14. `tiefdownconverter project list-profiles`

**Version:** `tiefdownconverter 0.7.0-alpha`

#### 4.14.1. Usage:

```
List the conversion profiles in the project.

Usage: tiefdownconverter project list-profiles

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

### 4.15. `tiefdownconverter project list-preprocessors`

**Version:** `tiefdownconverter 0.7.0-alpha`

#### 4.15.1. Usage:

```
List the preprocessors in the project.

Usage: tiefdownconverter project list-preprocessors

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

### 4.16. `tiefdownconverter project validate`

**Version:** `tiefdownconverter 0.7.0-alpha`

#### 4.16.1. Usage:

```
Validate the TiefDown project structure and metadata.

Usage: tiefdownconverter project validate

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

### 4.17. `tiefdownconverter project clean`

**Version:** `tiefdownconverter 0.7.0-alpha`

#### 4.17.1. Usage:

```
Clean temporary files from the TiefDown project.

Usage: tiefdownconverter project clean

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

### 4.18. `tiefdownconverter project smart-clean`

**Version:** `tiefdownconverter 0.7.0-alpha`

#### 4.18.1. Usage:

```
Clean temporary files from the TiefDown project.
If the number of conversion folders in the project is above this threshold, old folders will be cleaned, leaving only the threshold amount of folders.
The threshold is set to 5 by default, and is overwritten by the threshold in the manifest.

Usage: tiefdownconverter project smart-clean

Options:
  -h, --help
          Print help (see a summary with '-h')
```

### 4.19. `tiefdownconverter check-dependencies`

**Version:** `tiefdownconverter 0.7.0-alpha`

#### 4.19.1. Usage:

```
Validate dependencies are installed.

Usage: tiefdownconverter check-dependencies

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. Pull Requests

Pull Requests should be made with either a link to an issue or an explanation of

1. What was the problem
2. How is it solved now
3. How did it affect the documentation

It takes a lot of work to understand the intention of code you didn't write and then judging whether this was indeed the intended outcome. That's why it's helpful for everyone if there's an explanation on what was changed and why.

## 5.2. 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`.

## 5.3. 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.

## 5.4. 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`.

## 5.5. Tests

Currently primarily integration tests. See the `tests` folder for examples. Any pull request to main will automatically run tests, and the expectation is that at least the existing tests work. If they break, fix your code or, if you changed behavior on purpose, the tests.

I appreciate it if you add test coverage for your changes. I especially would appreciate more unit tests, but the tests I have are sufficient for now. Integration tests take priority over unit tests for me, as the overall behavior is more important to me than the individual functions, and I only have so much time that I want to spend on this project.

## 5.6. Documentation

When changing the documentation, it is of utmost importance that the documentation outputs are correctly generated. *These are not automatically generated on release* but rather held in git to more easily track changes during a pull request.

To make sure this documentation is up to date, consider whether your changes significantly affect the workflow of using TiefDownConverter. If you add a command or flag, make sure to run `tools/generate_docs.py`. Either way, when changing the documentation, always run `tiefdownconverter convert -p docs` before committing the changes.

**You need to have the font Iosevka installed!** If you don't, we cannot accept documentation changes. TiefDownConverter will throw a warning should the fonts not be installed. **The warning about Fira Mono missing is normal. This is the fallback font and not a requirement to compile the documentation**

## 5.7. Code Style

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