The mkbook Book

Kenton Hamaluik

Dec 20, 2019

The mkbook Book
©2019 Kenton Hamaluik
https://hamaluik.github.io/mkbook/

Contents

Preface

mkbook is my simpler alternative to *mdbook*[1] which is a great tool, however I really dislike some of the decisions they took—such as relying on javascript for highlighting and navigation and including a lot of bells and whistles such as javascript-based search.

This tool aims to work somewhat similarly to *mdbook*, but is generally intended to be a more minimal alternative that is customized more towards my needs and desires than anything else.

If you're not familiar with *mdbook*, *mkbook* is a tool to convert a collection of Markdown[2] files into a static website / book which can be published online. It was created to help me write documentation with minimum fuss while presenting it in an easy-to-consume manner.

[1] `https://crates.io/crates/mdbook`
[2] `https://commonmark.org/`

Chapter 1

Command-line Interface

mkbook may be installed using *Cargo* (`cargo install --force --path .` in the *mkbook* repo directory), and after that it presents a command-line interface:

```
$ mkbook
mkbook 0.3.0
Kenton Hamaluik <kenton@hamaluik.ca>


USAGE:
    mkbook [SUBCOMMAND]

FLAGS:
    -h, --help
            Prints help information

    -V, --version
            Prints version information


SUBCOMMANDS:
    build    build the book
    help     Prints this message or the help of the given
    ↪  subcommand(s)
    init     initialize a mkbook directory tree
    watch    build the book and continually rebuild whenever the
    ↪  source changes
```

1.1 The Init Command

The init command is a tool to help you get started, and will create an initial `README.md` file and a stub of your first chapter.

```
$ mkbook init --help
mkbook-init
initialize a mkbook directory tree

USAGE:
    mkbook init [OPTIONS]

FLAGS:
    -h, --help       Prints help information
    -V, --version    Prints version information

OPTIONS:
    -d, --directory <directory>    an optional directory to
    ↪  initialize into [default: src]
```

1.2 The Build Command

The build command is the primary command for *mkbook*, and is responsible for taking the `.md` files and building the resulting website.

```
$ mkbook build --help
mkbook-build
build the book

USAGE:
    mkbook build [OPTIONS]

FLAGS:
    -h, --help       Prints help information
    -V, --version    Prints version information

OPTIONS:
    -i, --in <in>      an optional directory to take the book sources
    ↪  from [default: src]
    -o, --out <out>    an optional directory to render the contents
    ↪  into [default: book]
```

1.3 The Watch Command

The watch command is basically the same as the `build` command, however after building it continues to monitor the source directory and if *any* changes are made (a file is saved, renamed, removed, created, etc), the entire book is re-built. In the future, this will hopefully be smarter but for now it just the whole thing at once. Stop watching using Ctrl + C or sending `SIGINT`.

```
$ mkbook build --help
mkbook-watch
build the book and continually rebuild whenever the source changes

USAGE:
    mkbook watch [OPTIONS]

FLAGS:
    -h, --help       Prints help information
    -V, --version    Prints version information

OPTIONS:
    -i, --in <in>      an optional directory to take the book sources
    ↪  from [default: src]
    -o, --out <out>    an optional directory to render the contents
    ↪  into [default: book]
```

1.4 Sample Usages

Build the GitHub Pages[1] document (this book):

```
mkbook build -i docs-src -o docs
```

Build the book, continually watching for changes and enabling auto-reloading in the browser so you can see the book update as you write:

```
mkbook watch -i docs-src -o docs --reload
```

Build a LaTeX[2] version of the book, then compile it to a PDF[3] and open it in evince[4]:

[1] https://pages.github.com/
[2] https://www.latex-project.org/
[3] https://en.wikipedia.org/wiki/PDF
[4] https://wiki.gnome.org/Apps/Evince

```
mkdir build
mkbook build -i docs-src -o docs --latex build/book.tex
cd build
xelatex -shell-escape book.tex
xelatex -shell-escape book.tex
evince book.pdf
```

Chapter 2

Markdown

mkbook relies pretty extensively on Markdown[1] for its ease of use. If you're not familiar with *Markdown*, it is a simple markup language that is design to be easy to read and write in plain text, and then (relatively) easy for a computer to convert into other formats such as HTML or LaTeX.

The above paragraph looks like this:

```
_mkbook_ relies pretty extensively on
[Markdown](https://daringfireball.net/projects/markdown/) for
its ease of use. If you're not familiar with _Markdown_, it is
a simple markup language that is designed to be easy to read
and write in plain text, and then (relatively) easy for a
computer to convert into other formats such as HTML or LaTeX.
```

Markdown by itself isn't quite enough for most purposes, so *mkbook* actually uses the *CommonMark* spec with some additional extensions to make life easier.

2.1 CommonMark

mkbook nominally utilizes CommonMark[2] with some GFM[3] extensions through the use of the comrak[4] crate. In using *comrak*, a specific set of options are used, which are listed here:

[1] https://daringfireball.net/projects/markdown/
[2] https://commonmark.org/
[3] https://github.github.com/gfm/
[4] https://crates.io/crates/comrak

```
let options: ComrakOptions = ComrakOptions {
    hardbreaks: false,
    smart: true,
    github_pre_lang: false,
    default_info_string: None,
    unsafe_: true,
    ext_strikethrough: true,
    ext_tagfilter: false,
    ext_table: true,
    ext_autolink: true,
    ext_tasklist: true,
    ext_superscript: true,
    ext_header_ids: Some("header".to_owned()),
    ext_footnotes: true,
    ext_description_lists: true,
    ..ComrakOptions::default()
};
```

Mostly, know that the following extensions are enabled:

- Strikethrough[5]
- Tables[6]
- Autolinks[7]
- Task Lists[8]
- Superscripts (`e = mc^2^.` → `e = mc<sup>2</sup>.`)
- Description Lists:

[5] https://github.github.com/gfm/#strikethrough-extension-
[6] https://github.github.com/gfm/#tables-extension-
[7] https://github.github.com/gfm/#autolinks-extension-
[8] https://github.github.com/gfm/#task-list-items-extension-

```
First term

: Details for the **first term**

Second term

: Details for the **second term**

    More details in second paragraph.
```

2.2 Syntax Highlighting

GFM syntax highlighting is also available by using fenced code tags with a label denoting the language, as such:

````
```c++
#include <stdio>

int main() {
    std::cout << ”Hello, world!” << std::endl;
    return 0;
}
```
````

which results in:

```
#include <stdio>

int main() {
    std::cout << "Hello, world!" << std::endl;
    return 0;
}
```

To denote the language you can either use one the language's extensions as the label, or the full name of the language (which is **not** case-sensitive).

The list of supported languages is currently as follows:

ASP
: `asa`

ActionScript
: `as`

AppleScript
: `applescript`, `script editor`

Batch File
: `bat`, `cmd`

BibTeX
: `bib`

Bourne Again Shell (bash)
: `sh`, `bash`, `zsh`, `fish`, `.bash_aliases`, `.bash_completions`, `.bash_functions`, `.bash_login`, `.bash_logout`, `.bash_profile`, `.bash_variables`, `.bashrc`, `.profile`, `.textmate_init`

C
: `c`, `h`

C#
: `cs`, `csx`

C++
: `cpp`, `cc`, `cp`, `cxx`, `c++`, `C`, `h`, `hh`, `hpp`, `hxx`, `h++`, `inl`, `ipp`

CSS
: `css`, `css.erb`, `css.liquid`

Cargo Build Results
: “

Clojure
: `clj`

D
: `d`, `di`

Diff
: `diff`, `patch`

Erlang
: `erl`, `hrl`, `Emakefile`, `emakefile`

Go
: `go`

Graphviz (DOT)
: `dot`, `DOT`, `gv`

Groovy
: `groovy`, `gvy`, `gradle`

HTML
: `html`, `htm`, `shtml`, `xhtml`, `inc`, `tmpl`, `tpl`

HTML (ASP)
: `asp`

HTML (Erlang)
: `yaws`

HTML (Rails)
: `rails`, `rhtml`, `erb`, `html.erb`

HTML (Tcl)
: `adp`

Haskell
: `hs`

Haxe
: `hx`, `hxsl`, `hscript`

Hxml
: `hxml`

JSON
: `json`

Java
: `java`, `bsh`

Java Properties
: `properties`

Java Server Page (JSP)
: `jsp`

JavaDoc
: “

JavaScript
: `js`, `htc`

JavaScript (Rails)
: `js.erb`

LaTeX
: `tex`, `ltx`

LaTeX Log
: “

Lisp
: `lisp`, `cl`, `clisp`, `l`, `mud`, `el`, `scm`, `ss`, `lsp`, `fasl`

Literate Haskell
: `lhs`

Lua
: `lua`

MATLAB
: `matlab`

Make Output
: “

Makefile
: `make`, `GNUmakefile`, `makefile`, `Makefile`, `OCamlMakefile`, `mak`, `mk`

Markdown
: `md`, `mdown`, `markdown`, `markdn`

MultiMarkdown
: “

NAnt Build File
: `build`

OCaml
: `ml`, `mli`

OCamllex
: `mll`

OCamlyacc
: `mly`

Objective-C
: `m`, `h`

Objective-C++
: `mm`, `M`, `h`

PHP
: `php`, `php3`, `php4`, `php5`, `php7`, `phps`, `phpt`, `phtml`

PHP Source
: “

Pascal
: `pas`, `p`, `dpr`

Perl
: `pl`, `pm`, `pod`, `t`, `PL`

Plain Text
: `txt`

Python
: `py`, `py3`, `pyw`, `pyi`, `pyx`, `pyx.in`, `pxd`, `pxd.in`, `pxi`, `pxi.in`, `rpy`, `cpy`, `SConstruct`, `Sconstruct`, `sconstruct`, `SConscript`, `gyp`, `gypi`, `Snakefile`, `wscript`

R
: `R`, `r`, `s`, `S`, `Rprofile`

R Console
: “

Rd (R Documentation)
: `rd`

Regular Expression
: `re`

Regular Expressions (Javascript)
: “

Regular Expressions (Python)
: “

Ruby
: `rb`, `Appfile`, `Appraisals`, `Berksfile`, `Brewfile`, `capfile`, `cgi`, `Cheffile`, `config.ru`, `Deliverfile`, `Fastfile`, `fcgi`, `Gemfile`, `gemspec`, `Guardfile`, `irbrc`, `jbuilder`, `podspec`, `prawn`, `rabl`, `rake`, `Rakefile`, `Rantfile`, `rbx`, `rjs`, `ruby.rail`, `Scanfile`, `simplecov`, `Snapfile`, `thor`, `Thorfile`, `Vagrantfile`

Ruby Haml
: `haml`, `sass`

Ruby on Rails
: `rxml`, `builder`

Rust
: `rs`

SCSS
: `scss`

SQL
: `sql`, `ddl`, `dml`

SQL (Rails)
: `erbsql`, `sql.erb`

Sass
: `sass`

Scala
: `scala`, `sbt`

Shell-Unix-Generic
: “

TOML
: `toml`, `tml`, `Cargo.lock`, `Gopkg.lock`, `Pipfile`

Tcl
: `tcl`

TeX
: `sty`, `cls`

Textile
: `textile`

XML
: `xml`, `xsd`, `xslt`, `tld`, `dtml`, `rss`, `opml`, `svg`

YAML
: `yaml`, `yml`, `sublime-syntax`

camlp4
: “

commands-builtin-shell-bash
: “

reStructuredText
: `rst`, `rest`

2.3 PlantUML Diagrams

If you have PlantUML[9] installed and available on your path, *mkbook* will try to render any code blocks with a language tag of `plantuml` as inline SVG images.

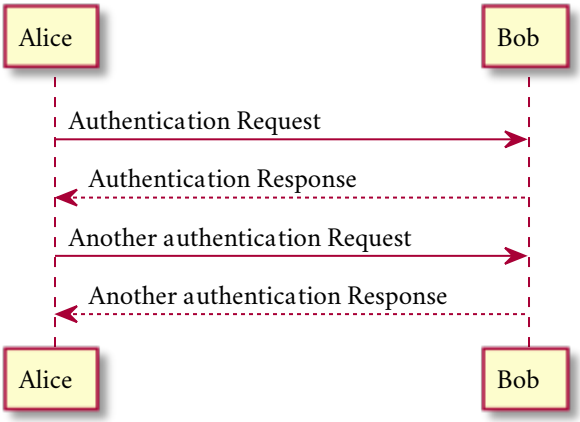
For example:

[9] `http://plantuml.com/`

```
```plantuml
@startuml
Alice -> Bob: Authentication Request
Bob --> Alice: Authentication Response

Alice -> Bob: Another authentication Request
Alice <-- Bob: Another authentication Response
@enduml
```
```

is rendered as:



This feature is still experimental, but I find it handy for my books.

2.4 KaTeX (Math) Formulas

If you have KaTeX[10] installed and available on your path, *mkbook* will try to render any code blocks with a language tag of `katex` as inline math blocks.

For example:

[10] `https://github.com/KaTeX/KaTeX`

```
```katex
x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}
```
```

is rendered as:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{2.1}$$

This feature is still experimental, but I find it handy for my books.

2.5 Images

To include an image, use the standard markdown format:

```
![alt](url "title")
```

This will wrap the image in a `figure` with an associated `figcaption` containing the title of the image, as so:

```
![a bear](https://placebear.com/g/512/256 "A majestic bear")
```

will render as:

Figure 2.1: A majestic bear

2.6 Tables

Tables are created using the pipe syntax[11], for example the following:

```
| Tables        | Are           | Cool  |
| ------------- |:-------------:| -----:|
| col 3 is      | right-aligned | $1600 |
| col 2 is      | centered      |   $12 |
| zebra stripes | are neat      |    $1 |
```

renders as:

Tables	Are	Cool
col 3 is	right-aligned	$1600
col 2 is	centered	$12
zebra stripes	are neat	$1

2.7 Task Lists

You can also use GFM[12]-style task lists[13] to indicate a TODO list:

- ☐ a task list item
- ☐ list syntax required
- ☐ normal **formatting**
- ☐ incomplete
- ☑ completed

2.8 Links

mkbook uses standard *MarkDown* notation for links:

```
[link text](link url)
```

Links can be separated into three types:

[11] https://github.github.com/gfm/#tables-extension-
[12] https://help.github.com/en/github/writing-on-github
[13] https://github.blog/2013-01-09-task-lists-in-gfm-issues-pulls-comments/

1. External links (prepended by `http://` or `https://`)
2. Internal links (relative path names)
3. Reference links (prepended by `ref://` and then followed by the chapter title) to refer to other chapters in the book**

Note: Reference links aren't implemented yet!

Chapter 3

Front Matter

Each `.md` file can optionally contain a header with metadata describing the document. If the header isn't present, or if any keys are missing, default values will be used.

To insert a header into a `.md` file, insert three dashes (–), followed by a new-line, followed by the front matter contents, followed by a newline, then another three dashes and a new-line. The metadata is in the TOML[1] format, so for example the front-matter (and first line) for a file could look like this:

[1] `https://github.com/toml-lang/toml`

```
---
title = "Front Matter"
author = "Kenton Hamaluik"
pubdate = 2019-11-29T15:22:00-07:00
---

Each `.md` file can optionally contain a header with metadata
↪  describing the document. If the header isn't present, or if any
↪  keys are missing, default values will be used.
```

3.1 Supported Keys

The list of supported keys is subject to change, but for now it is as follows:

title
: A human-readable title for the document (defaults to the filename)

author
: The author (or authors) who wrote the chapter (defaults to "Anonymous")

pubdate
: The RFC 3339[2] timestamp of when the chapter was published (defaults to the time at build)

url
: The relative URL of the file, defaults to the generated route (you probably shouldn't set this one)

[2] http://tools.ietf.org/html/rfc3339

Chapter 4

Structure

mkbook follows a fairly simple directory structure for now, with a `README.md` file declaring the book's metadata, and `.md` files defining each chapter of the book.

4.1 README.md

mkbook generally requires a `README.md` file to reside in your source directory. This file is responsible for defining the metadata associated with your book:

- The book's title (`title`)
- The book's author (`author`)
- The publication date (`pubdate`)
- The canonical URL for the book (`url`)
- A markdown-formatted description of the book

If the `README.md` file or any of the entries are missing, default values will be used. The `README.md` file should be formatted as any other page, with the `title`, `author`, `pubdate`, and `url` specified in the frontmatter, and the book description the *Markdown* contents of the `README.md` file.

4.1.1 Sample

```
---
title = "The mkbook Book"
author = "Kenton Hamaluik"
url = "https://hamaluik.github.io/mkbook/"
---

_mkbook_ is my simpler alternative to
↪  [mdbook](https://crates.io/crates/mdbook)
which is a great tool, but for which I really dislike some of the
↪  decisions they
took, such as relying on javascript for highlighting and navigation,
↪  and
including a lot of bells and whistles such as javascript-based
↪  search.

This tool aims to work somewhat similarly to _mdbook_, but is
↪  generally intended
to be a more minimal alternative that is customized more towards my
↪  needs and
desires than anything else.
```

4.1.2 Default Values

`title`
: "My Cool Book"

`author`
: "Anonymous"

`pubdate`
: The date the book was built from the command line, in UTC time

`url`
: ""

`description`
: ""

4.2 Assets

Any files in the `src` directory which are not included in `.gitignore` and do not end in the extension `.md` will be copied to the output folder. You can use this to include images, files, etc, for example the following image is an asset bundled with the book:

```
![chapter-six](book-chapter-six-5834.jpg "Photo by Kaboompics.com
↪  from Pexels")
```

Figure 4.1: Photo by Kaboompics.com from Pexels

4.3 Documents

mkbook works on mostly a flat directory structure, however one level of sub-directories are supported in order to create sections within chapters. Files that don't end in a `.md` extension are completely ignored. Each `.md` file in the root source directly is it's own chapter. To create chapters with sub-sections, create a sub-directory in the root directory and then create a `README.md` file, which will become the root of the chapter, with all `.md` files in the sub-directory becoming sections in the chapter. The `title` in the `README.md` file's frontmatter will be used as the name of the chapter.

The order of the book is based on the alphabetical order of the file names (actually it's based on Rust's implementation of `PartialOrd` for str[1]). Thus,

[1] `https://doc.rust-lang.org/std/cmp/trait.PartialOrd.html#impl-PartialOrd%3Cstr%3E`

it is recommended to lay out your book chapters with manual numbering of the file names, as such:

```
src/
├── README.md
├── 00-foreword.md
├── 01-introduction.md
├── my-picture.jpg
└── 02-my-first-chapter
    ├── README.md
    ├── 01-my-first-section.md
    ├── 02-my-second-section.md
    └── etc...
```

An index and navigation will be automatically generated from these files, taking the information for each file from it's front-matter.

Chapter 5

Customization

There isn't any way to customize the templates nor the CSS yet, though I will investigate this if the need arises. This is because both the templates and CSS are currently compiled at compile-time instead of run-time.

Chapter 6

How it Works

mkbook generates a completely static, javascript-free website from a series of Markdown files. All of the layout and styling is controlled purely by hand-crafted CSS specific to this book's purpose.

6.1 Assets

mkbook currently bundles two assets which get written into the book directory: `favicon.ico`, and `icons.svg`. `favicon.ico` is the Font Awesome 5 book icon[1], and `icons.svg` contains 3 Font Awesome 5[2] arrow icons: arrow-left[3], arrow-right[4], and arrow-up[5] which are used for navigation. These files are compiled into the *mkbook* binary using the `include_bytes!` macro[6], and written to the output folder on each build.

6.2 Styling

mkbook utilizes Sass[7] to define it's styles; you can view the sources on github[8]. In *mkbook*'s build script, the styles are compiled from their native `.scss` for-

[1] `https://fontawesome.com/icons/book?style=solid`
[2] `https://fontawesome.com/`
[3] `https://fontawesome.com/icons/arrow-left?style=solid`
[4] `https://fontawesome.com/icons/arrow-right?style=solid`
[5] `https://fontawesome.com/icons/arrow-up?style=solid`
[6] `https://doc.rust-lang.org/std/macro.include_bytes.html`
[7] `https://sass-lang.com/`
[8] `https://github.com/hamaluik/mkbook/tree/master/style`

mat into a single, compressed `.css` file using sass-rs[9]. The resulting `.css` file is then bundled into the binary using the `include_str!` macro[10]. When a book is generated, this `.css` is written to the output folder as `style.css`, where it is included by each generated `.html` file.

6.3 Templates

mkbook contains two template files: one for the index, and one for each page / chapter, and uses Askama[11] to render the templates. Since the *Askama* templates are compiled when *mkbook* is compiled, it is not currently possible to change the templates at run time. You can view the sources for these templates on github[12].

6.4 Markdown Formatting

Markdown is formatted usiing comrak[13] with some specific options, see the Markdown chapter[14] for more information.

6.5 Syntax Highlighting

Code is syntax-highlighted using syntect[15] with the default langauges and the `base16-eighties` colour scheme. Some additional languages above the base list supported by *syntect* have been aded:

- haxe[16]
- hxml[17]
- sass[18]
- scss[19]

[9] `https://crates.io/crates/sass-rs`
[10] `https://doc.rust-lang.org/std/macro.include_str.html`
[11] `https://crates.io/crates/askama`
[12] `https://github.com/hamaluik/mkbook/tree/master/templates`
[13] `https://crates.io/crates/comrak`
[14] `02-markdown.html`
[15] `https://crates.io/crates/syntect`
[16] `https://haxe.org/`
[17] `https://haxe.org/manual/compiler-usage-hxml.html`
[18] `https://sass-lang.com/documentation/syntax#the-indented-syntax`
[19] `https://sass-lang.com/documentation/syntax`

- toml[20]

[20] https://github.com/toml-lang/toml

Chapter 7

LaTeX Output

mkbook can also export a LaTeX[1] file which can be used to convert your book to a beatiful, ready-to-print PDF[2]. This feature is still under heavy development as it's not quite as smooth as I would like, and the generated `.tex` document is perhaps a bit too customized—I'm still exploring this.

For now, however, you can convert your book into a single `.tex` file with the following command which will create the file `./print/book.tex` along with any images needed to render the book:

[1] `https://www.latex-project.org/`
[2] `https://en.wikipedia.org/wiki/PDF`

```
mkbook build -l ./print/book.tex
```

Note that this command is more about preparing a `.tex` file that you can then further customize for your own book than having a complete, ready-to-go PDF that is entirely your own—the current LaTeX template that gets generated works for me but it may not work for you.

7.1 Images

If an image in the document is an external image (i.e. it starts with `http://` or `https://`), *mkbook* will attempt to download the image the same directory that the generated LaTeX document resides in. If it cannot do so, it will tell you. If, on the other hand, the image is in the source tree, it will be copied over the same way that any other asset is and should be available to the LaTeX file.

Similar to this, *mkbook* will attempt to render any `plantuml` code sections into `.svg` files which also get included in the book.

7.2 Building the Book

The current LaTeX template requires the following packages to be installed:

- ulem[3]
- fontspec[4]
- sectsty[5]
- xcolor[6]
- minted[7]
- amsmath[8]
- amssymb[9]

[3] `https://ctan.org/pkg/ulem`
[4] `https://ctan.org/pkg/fontspec`
[5] `https://ctan.org/pkg/sectsty`
[6] `https://ctan.org/pkg/xcolor`
[7] `https://ctan.org/pkg/minted`
[8] `https://ctan.org/pkg/amsmath`
[9] `https://ctan.org/pkg/amssymb`

- enumitem[10]
- textcomp[11]
- graphicx[12]
- float[13]
- svg[14]
- menukeys[15]

The template also requires XeTeX[16] and the following fonts to be available on your system:

- Crimson[17]
- Poppins[18]
- Source Code Pro[19]

Finally, in order to color the source code, you must have Pygments[20] installed and the `pygmentize` executable must be available on your path.

If you meet all these requirements, you can build the book using `xelatex` (better yet, use `latexmk`). Assuming you built the `book.tex` file in the `print` directory as above:

[10] `https://ctan.org/pkg/enumitem`
[11] `https://ctan.org/pkg/textcomp`
[12] `https://ctan.org/pkg/graphicx`
[13] `https://ctan.org/pkg/float`
[14] `https://ctan.org/pkg/svg`
[15] `https://ctan.org/pkg/svg`
[16] `https://www.tug.org/xetex/`
[17] `https://github.com/skosch/Crimson`
[18] `https://www.fontsquirrel.com/fonts/poppins`
[19] `https://github.com/adobe-fonts/source-code-pro`
[20] `https://pygments.org/`

```
cd print
latexmk -xelatex -shell-escape book.tex
```

Note that the `-shell-escape` argument is required in order to get *Pygments* to colour your source code, and the `xelatex` command is run twice in order to properly build the table of contents.

Note also that in the current template, the pages that are created are 5.5 inches by 8 inches. This is to facilitate booklet printing on North American letter paper. Feel free to change this in the generated `book.tex` file before compiling if you need to.

7.2.1 Compiling a Booklet

If you want to easily print this book as a booklet, you can take one more step to arrange the pages so that a simple duplex print on any printer will produce signatures that you can easily bind yourself (there are many tutorials online for doing this, I recommend Easy paperback book binding how-to[21] by Rubén Berenguel).

The first step is to create a file alongside your compiled `book.pdf` file called `printbook.tex` with the contents as such:

[21] `https://mostlymaths.net/2009/04/easy-paperback-book-binding-how-to.html/`

```
\documentclass[letterpaper]{article}
\usepackage[final]{pdfpages}
\begin{document}
\includepdf[pages=-,nup=1x2,landscape,signature=32]{book.pdf}
\end{document}
```

You can change the value of `signature` as you like, but keep it a multiple of 4. The signature[22] is the number of pages (**not** sheets of paper) which get combined into a "mini-booklet", and the final book is a combination of all of the signatures ("mini-booklets") to make the full book. Essentially, if you divide this number by 4, you'll get the number of sheets of paper that you'll have to staple together at a time. For a signature of 32 pages, this will mean stapling together 8 pages at a time.

Note that if you have a relatively short book, it may be advantageous to just do all of the book's pages into one signature, in this case make the signature the next multiple-of-four value higher than the total number of sheets in the `book.pdf` file. For example: if `book.pdf` contains 45 pages, make `signature=48` to put everything into a single signature.

Finally, compile `printbook.tex` using `pdflatex`:

```
pdflatex printbook.tex
```

As a sample, you can view the compiled book[23] and printbook[24] files for this book to see how this can turn out.

[22] https://en.wikipedia.org/wiki/Section_(bookbinding)
[23] book.pdf
[24] printbook.pdf