Directory and file structure

Hans Petter Langtangen[1,2]

[1]Center for Biomedical Computing, Simula Research Laboratory
[2]Department of Informatics, University of Oslo

Mar 25, 2015

Contents

This document describes the file structure of book or thesis projects.

1 Directory structure

We shall outline a directory structure that can be effective when assembling different DocOnce documents into a book. The root directory for all documentation is called `doc`, with two subdirectories: `src` for all the DocOnce source code, and

`pub` for compiled (published) documents in various formats. A third subdirectory, `web`, is often present as an entry point for the web pages on GitHub. This directory typically contains the autogenerated `index.html` and additional style files on GitHub. The `index.html` file should have links to published documents in `../pub`.

Under `src` we may have a directory `chapters` for the individual chapters and a directory `book` for the assembly into a book. One may also think of more than one book directory if a set of documents (chapters) naturally leads to multiple books. All chapters can then be put in the `chapters` directory.

Each chapter has a short nickname, say `ch2` for simplicity for Chapter 2 (a more descriptive name related to the content is obviously much better!). Figures are placed in subdirectory `fig-ch2` and computer code in subdirectory `src-ch2`. These two latter directories may have subdirectories if desired. We may also include a directory `mov-ch2` for video files, `exer-ch2` for answers to exercises, etc.

An outline of the directory structure is listed below.

```
doc
  src
    chapters
      ch2
        fig-ch2
        src-ch2
        mov-ch2
        exer-ch2
    book
  pub
    chapters
    book
```

Under `book`, we typically have a document `book.do.txt` for the complete book. This is a file with a lot of `# #include "...do.txt"` for including the files for the various chapters, see Section 3 for details. Additional files in the `book` directory include make files for compiling the book, scripts for packing the book for publishing, perhaps an errata document, etc.

2 Newcommands

Files with names `newcommands*.tex` are by DocOnce treated as files with definition of newcommands for LaTeX mathematics. These files must reside in the same directory as the DocOnce source files. However, for a book project, it is common to have one newcommands file shared by all chapters. This file is placed in `doc/src/chapters/newcommands.p.tex` and copied to a specific chapter by the make script for that chapter. The extension of the file is `.p.tex`, indicating that the file has to be *preprocessed* by `preprocess` prior to being copied. The reason is that one occasionally wants the definitions of the newcommands to depend on the output format (standard LaTeX or MathJax). For example,

subscripts in `mbox` font look best with footnotesize font in plain LaTeX, while the larger `small` font is more appropriate for MathJax. We can then put the following definitions in `newcommands.p.tex`:

```
% #if FORMAT in ("latex", "pdflatex")
% Use footnotesize in subscripts
\newcommand{\subsc}[2]{#1_{\mbox{\footnotesize #2}}}
% #else
% In MathJax, a different construction is used
\newcommand{\subsc}[2]{#1_{\small\mbox{#2}}}
% #endif
```

The make script will then run `preprocess` on this file, typically

```
preprocess -DFORMAT=pdflatex ../newcommands.p.tex > newcommands.tex
# or
preprocess -DFORMAT=html ../newcommands.p.tex > newcommands.tex
```

DocOnce newcommands are for mathematics only!

Note that newcommands in DocOnce context are only used for mathematics, rendered by LaTeX or MathJax. Newcommands for other LaTeX constructions (such as section or boxes) should not be used in the DocOnce source code as these are confined to the LaTeX format. Use instead Mako functions.

3 Assembling different pieces to a book

Many smaller writings in the DocOnce format can be assembled into a single, large document such as a book or thesis. The recipe for doing this appears below.

3.1 Organization of a chapter

Suppose one chapter) of the book has the nickname `ch2` and may hold all text or just include text in other DocOnce files, e.g., `part1.do.txt`, `part2.do.txt`, and `part3.do.txt`. In this latter case, `ch2.do.txt` has the simple content

```
# #include "part1.do.txt"

# #include "part2.do.txt"

# #include "part3.do.txt"
```

Note that the `ch2.do.txt` file contains just plain text without any `TITLE`, `AUTHOR`, or `DATE` lines and without any table of contents (`TOC`) and bibliography (`BIBITEM`). This property makes `ch2.do.txt` suitable for being including in other documents like a book. However, to compile `ch2.do.txt` to a stand-alone document, we normally want a title, an author, a date, and perhaps a table of contents. We also want a bibliography if any of the included files has `cite` tags. To this end, we create a wrapper file, say `main_ch2.do.txt`, with the content

```
TITLE: Some chapter title
AUTHOR: A. Name Email:somename@someplace.net at Institute One
AUTHOR: A. Two at Institute One & Institute Two
DATE: today

TOC: on

# #include "ch2.do.txt"

======= References =======

BIBFILE: ../papers.pub
```

Recall that DocOnce relies on the Publish software for handling bibliographies. It is easy to import from BibTeX to Publish and create a database of references (`papers.pub`) to get started (but we recommend to continue working with the Publish database directly and collect new items in the `papers.pub` file as Publish is more flexible than BibTeX).

3.2 Figures and source code

As described in Section 1, we recommend to put figures and source codes (to be included in the document) in separate directories. Although such directories could have natural names like `fig` and `src`, it will cause trouble if we do not use unique names for these directories, like `fig-ch2` and `src-ch2`. Otherwise, we would need to copy all figures in all pieces into a common `fig` directory for the book and all source code files into a `src` directory. With unique names, figures and source code files can always reside in their original locations, and we can easily reach them through links. This will be described next.

3.3 Assembly of chapters to a book

All the files associated with the `ch2` document and chapter reside in the `ch2` directory. A fundamental principle of DocOnce is to have just one copy of the files ("Document once!"). To include the `ch2` text in a larger document like a book, we just need to include the `ch2.do.txt` file and a chapter heading. Here is an example of a document `book.do.txt` for a complete book:

```
TITLE: This is a book title
AUTHOR: A. Name Email:somename@someplace.net at Institute One
AUTHOR: A. Two at Institute One & Institute Two
DATE: today

TOC: on

========= Preface =========

# #include "../chapters/preface/preface.do.txt"

========= Heading of a chapter =========

# #include "../chapters/ch2/ch2.do.txt"

# Similar inclusion of other chapters

========= Appendix: Heading of an appendix =========

# #include "../chapters/nickname/nickname.do.txt"

======= References =======

BIBFILE: ../papers.pub
```

When running `doconce format` on `book.do.txt`, the entire document is contained in *one* big file[1] (!). To see exactly what has been included, you can examine the result of running the first preprocessor, `preprocess`, on `book.do.txt`. All the includes are handled by this preprocessor. The result is contained in the file `tmp_preprocess__book.do.txt`, which then contains the entire DocOnce source code of the book. The second preprocessor, `mako`, is thereafter run (if DocOnce detects that it is necessary). The result of that step is available in `tmp_mako__book.do.txt`. It is important to examine this file if there are problems with Mako variables or functions. The `tmp_mako__book.do.txt` file is thereafter translated to the desired output format.

Say we want to produce a LaTeX document:

Terminal

```
Terminal> doconce format pdflatex book [options]
```

If the DocOnce source contains copying of source code from files in `@@@CODE` constructs, it is important that `doconce` finds the files. For example,

```
@@@CODE src-ch2/myprog.py  fromto: def test1@def test2
```

will try to open the file `src-ch2/myprog.py`. Since this file is actually located in `../ch2/src-ch2/myprog.py`, `pdflatex` will report an error message. A local link to that directory resolves the problem:

[1] A single DocOnce file and consequently a single `.tex` file works out well on today's laptops. A book with 900 pages [1] has been tested!

Terminal

```
Terminal> ln -s ../chapters/ch2/src-ch2 src-ch2
```

Similarly, the LaTeX code in `book.tex` for inclusion of a figure may contain

```
\includegraphics[width=0.9\linewith]{fig-ch2/fig1.pdf}
```

For this command to work, it is paramount that there is a link `fig-ch2` in the present `book` directory where the `pdflatex` command is run to the directory `../chapters/ch2/fig-ch2` where the figure file `fig1.pdf` is located.

It is recommended to use the function `make_links` in `scripts.py` to automatically set up all convenient links from the `book` directory to the individual chapter directories. Provided the *list of chapter nicknames* at the top of `scripts.py` *is correct*, you can just run

```
>>> import scripts
>>> scripts.make_links()
```

to automatically set up all links to all `src-*`, `fig-*`, and `mov-*` directories. You need to rerun this `make_links` function after inclusion of a new chapter in the `chapters` tree.

Identify LaTeX errors in the original chapter files!

When you run `pdflatex book` and get LaTeX errors, you need to see where they are in `book.tex` and use this information to find the appropriate DocOnce source file in some chapter. Usually, there are few errors at the "book level" if each individual chapter has been compiled. To this end, you can use `scripts.py` to automatically compile each chapter separately. The process is stopped as soon as a DocOnce or LaTeX error is encountered.

```
>>> import scripts
>>> scripts.compile_chapters()
```

3.4 About figures when publishing HTML

There will be `<img src=fig-ch2/fig1.png>` type of tags in HTML code produced by DocOnce, so it is very important to ensure that the *published* `.html` files

have access to a subdirectory `fig-ch2`. Normally, one needs to copy `fig-ch2` from the `ch2` chapter source directory to some publishing directory that stores all the files necessary for accessing the entire HTML document on the web.

4 Tools

You can start a new, future, potential book project by simply copying the directory structure of the setup4book-doconce[2] repository on GitHub. Then you can follow the instructions below to start writing and adapting the structure to your project's needs.

4.1 Making a new chapter

Under `doc/src/chapters` you find the chapters in this "sample book" as well as a script `doc/chapters/mkdir.sh` that creates a new directory for you with the typical files needed for a new chapter. You can either edit existing chapters, or make a brand new empty chapter by running

Terminal

```
Terminal> sh mkdir.sh mychap
```

This command makes a directory `mychap` for a new chapter with nickname `mychap`. (Files from the `template` directory are used to populate `mychap`.)

4.2 Compiling the chapter

PDF. To make a stand-alone document of a chapter, by compiling to LaTeX and PDF, we propose the convention to have a `make.sh` in each chapter directory. This `make.sh` can in most cases just call up a common `../make.sh` script,

```
bash -x ../make.sh main_mychap
```

or optionally with some command-line arguments,

```
bash -x ../make.sh main_mychap --encoding=utf-8
```

The `doc/src/chapters/make.sh` script is quite general and may be edited according to your layout preferences of the LaTeX documents.

[2] `https://github.com/hplgit/setup4book-doconce`

Remark. The suggested `make.sh` file applies the `--latex_code_style=` option to `doconce format` for specifying the typesetting of blocks of computer code in LaTeX. Originally, DocOnce applied the `ptex2tex` program to select such typesetting, but the new method is more flexible and simpler (in that it gives cleaner LaTeX code). (With `ptex2tex` one would need a common configuration file `doc/chapters/.ptex2tex.cfg` to be copied by `doc/chapters/make.sh` to the chapter directory prior to running 'ptex2tex.)

HTML. There is also a script `doc/src/chapters/make_html.sh` for making HTML versions of the chapter. Just call this as

Terminal

```
Terminal> bash ../make_html.sh main_mychap
```

to make HTML versions of the `mychap` chapter. The current version of `make_html.sh` creates three types of HTML layouts and an `index.html` file with a list of links to these three files.

4.3 Compiling the book

Go to `doc/src/book` and run `make.sh` to compile the book. This requires that `book.do.txt` performs the right include of chapters, table of contents, and bibliography.

There are many other tools in `doc/src/book` too, e.g., the mentioned library of handy scripts in `scripts.py`, and an example on how to pack all files of the entire book projects for publishing with Springer (`pack_Springer.sh`).

The current book layout created by `make.sh` makes use of a (now outdated) Springer T2 style for textbooks (requires the `.cls` and `.sty` files in the `book` directory). Other Springer styles supported by DocOnce are Lecture Notes in Computational Science and Engineering (monographs and proceedings), Lecture Notes in Computer Science (proceedings), and Undergraduate Texts in Physics. Other book styles will require some manual work, either working out a LaTeX preamble for a special style and use that when compiling `book.do.txt` or actually extending the DocOnce source code.

5 Study guides and slides

DocOnce has good support for creating slides from ordinary documents with running text. Rather than speaking about slides, we think of *study guides* where the material is presented in a very condensed, effective, summarizing form for overview, use in lectures, and repetition. The slide format is a good way of writing study guides, but by explicitly thinking of study guides the slide format can be made more effective for self-study when overview and repetition are necessary - with a particular emphasis on gaining understanding.

It is, of course, a very challenging balance between enough information for self-study by reading slides and overwhelmingly much text and information in slides for oral presentations. Text must anyway be minimized all the way on slides, and the reader of a study guide is supposed to also be a reader of the underlying running text in the chapter.

5.1 Slide directory

For each DoOnce file in the chapter `ch2` it can be wise to make a corresponding study guide file in the subdirectory `lec-ch2`. For example, `part1.do.txt` has its counterpart with slides in `lec-ch2/part1.do.txt`. Then there is a file `lectures_ch2.do.txt` which assembles the parts if `lec-ch2`, typically with a content like

```
TITLE: Study Guide: Some title
AUTHOR: Author Name Email:somename@someplace.net at Institute One
DATE: today

# #ifdef WITH_TOC
!split
TOC: on
# #endif

# #include "lec-ch2/part1.do.txt"

# #include "lec-ch2/part2.do.txt"

# #include "lec-ch2/part3.do.txt"
```

5.2 Generating slides from running text

The author has the following work flow for generating slides for a chapter file, say `part1.do.txt`.

1. Copy `part1.do.txt` to `lec-ch2/part1.do.txt`.
2. Make `lectures_ch2.do.txt` and include `lec-ch2/part1.do.txt`.
3. Decide on *parts* of the slide collection. Often a part can be a section in the parent `ch2.do.txt` file, but sometimes it can be more natural to have larger parts than sections in the slide collection.
4. Each part in the slide file has a DocOnce section heading with 7 `=`, while each slide has a DocOnce subsection heading with 5 `=`.
5. Edit `lec-ch2/part1.do.txt`:
    - One can keep subsection headings from the running text for the most part, but slides need many more subsection headings.

- Try to let the heading summarize explicitly a conclusion/rule from the slide (the slide table of contents is then a set of conclusions/rules!)
- Remember a `!split` right above very slide heading!
- Compile frequently and look at the slides: they become over-full very quickly so there is a constant need for dividing slides into new ones with new headings.
- Read a paragraph, focus on its main idea and result, and see how it can be condensed to one sentence or a few bullet points. *Making effective slides is the art of condensing the most important information in the text to a eye-catching format.*
- Do not remove figures without a very good reason. Figures are important!
- Add new images to live up the presentation. In slides you may think of cartoons or entertaining images that would never be suitable in a chapter/book, but they may help attract attention, communicate ideas, and enhance the memory process.
- Condense every mathematical derivation. Make sure the goal and end result is clear before diving into details.
- Detailed derivations are seldom of interest in a study guide or oral presentation - refer to the underlying running text in the chapter for the details. Focus on ideas and key mathematical steps (if they are important enough).
- Remember that equations are sometimes excellent images for ideas! Complicated equations can therefore be important slide elements although the details will never be addressed.
- It is quite often wise to remove equation numbers in slides. You can edit the LaTeX math environment manually, or use `--denumber_all_equations` to do it automatically. Remember that references to equations numbers must be removed from the slides too!
- Movies are effective in slides. It is still a hassle to get them displayed correctly in PDF files, so using a test on `FORMAT` and writing `MOVIE` for HTML output and just a link in PDF output might be necessary. See the manual[3] for how to work with movies in DocOnce.

The slides are to fulfill three purposes:

1. reading as a study guide to get overview before reading the full text of chapter,
2. watching as slides during an oral presentation,
3. reading as a study guide to repeat and enforce overview of the material.

[3] `http://hplgit.github.io/doconce/doc/pub/manual/html/manual.html#movies`

It is highly non-trivial to meet all these purposes: limit the information on the slides, make them as visual as *feasible*, make them self contained, and provide the *sufficient* amount of information. Considerable iterations are always needed. Reading the slides as a study guide is easy to accomplish. The slides' properties in live presentations can only be tested by speaking to them (making a rough draft of a video podcast is a very effective way of testing the slides' quality).

5.3 Compiling slides

There is a quite general script in `doc/src/chapters/make_lectures.sh` for compiling a slide collection defined in a file like `lectures_ch2.do.txt`. Just run

Terminal

```
Terminal> bash ../make_lectures.sh lectures_ch2.do.txt
```

from the chapter directory. Note that the script will first spell check the slide files. This is done in the `lec-ch2` directory. Errors are reported in files located in `lec-ch2`. To update the chapter's dictionary for spell checking, you need to do

Terminal

```
Terminal> cp lec-ch2/new_dictionary.txt~ .dict4spell
```

in the `ch2` chapter directory.

Similarly, to look at misspellings, the file `lec-ch2/misspellings.txt~` is the relevant file.

The `make_lectures.sh` script compiles a variety of slides:

- First a plain LaTeX PDF document to catch as many errors in the DocOnce source as early as possible. This document can also be used for compact printing of the contents of the study guide (and the output looks definitely like a study guide and not slides!).
- HTML5 `reveal.js` slides with different colors.
- HTML5 `deck.js` slides. This format is usually inferior to `reveal.js`, but is also very much personal taste.
- LaTeX Beamer slides. Edit the `theme=red_shadow` line in `make_lectures.sh` to control the Beamer theme.
- Remark (Markdown) slides for viewing in a browser.

References

[1] H. P. Langtangen. *A Primer on Scientific Programming With Python*. Texts in Computational Science and Engineering. Springer, fourth edition, 2014.