Using DocStrip for multiple document variants

Matthew Leingang*

Abstract

I describe a method of keeping multiple variants of the same document within a single file, using DocStrip.

1 Introduction

As a college professor, there are several times when I need to keep teaching materials in several different forms. For example:

- A single class day's lesson might consist of lecture slides in the `beamer` class, a handout of the same slides printed 2–3 on a page for student notes, my own lecture notes as a manuscript, a worksheet for in-class activity, and solutions to that worksheet.
- A single week's homework assignment might consist of problem statements, with hints and reading notes, a LaTeX template for students to fill in with their own answers, and solutions with comments to be published after the assignment has been graded.

Over the years I have developed a workflow for maintaining these "bundles" of documents in the same file, using the LaTeX DocStrip utility. This workflow allows me to programmatically vary the content and formatting, and avoids external scripts or filesystem hacks. In this article I will introduce the reader to DocStrip and explain how I use it.

2 The DocStrip utility

DocStrip [5] was originally designed as a literate programming method. LaTeX package and class authors use it to write documentation for their code in the same file as the implementation, within commented lines. DocStrip would *strip* out the comment lines and use them to produce *doc*umentation. The slimmer package and class files would be installed to save compile time. In subsequent versions DocStrip developed the ability to write lines to several different files in one batch, through the use of *options*.

DocStrip batches are programmed in a TeX file as in Listing 1. The `\input` line loads the DocStrip code. The `\generate` line instructs TeX, effectively, to "read `foo.dtx` and write `foo.sty`, setting the `package` option".

When generating files, DocStrip ignores all lines beginning only with a single `%`. Non-commented lines are written to all destination files. Lines beginning with a *guard* will be written to the destination file depending on the options set. Each guard begins with a `%` and contains a boolean expression enclosed by angle brackets. For example, a line beginning with `%<bar>` will only be passed to generated files when the `bar` option is set. DocStrip can generate more than TeX files, too; for example, BibTeX files, data files, and shell scripts can be embedded in the master file and extracted.

* The author wishes to thank the editors and reviewers for their thoughtful and productive feedback.

Listing 1: A minimal DocStrip batch file

```
\input docstrip.tex
\generate{\file{foo.sty}
        {\from{foo.dtx}{package}}}
\endbatchfile
```

Now putting guards at the start of every line would be cumbersome to type. So guard modifiers are used to delimit blocks of code with the same guard. Any expression preceded by '`*`' will apply the indicated guard to every line that follows, until the identical expression is encountered with the '`/`' modifier. This gives an almost HTML-like layer to the DocStrip source file, where blocks of code between lines starting with `%<*bar>` and `%</bar>` will be written to any file generated with the `bar` option.

A DocStrip batch declaration such as in Listing 1 often resides in a separate file. If this code were in `foo.ins`, running TeX on `foo.ins` would extract `foo.sty` from `foo.dtx` and quit. But DocStrip files can be also be configured to "self-extract" by putting the batch declaration at the beginning of the file. In this configuration, running TeX on `foo.dtx` will instruct TeX to parse `foo.dtx` a second time, this time writing `foo.sty`. Thus, the document content and extraction instructions can reside in a single file.

3 Example: A problem set with answer template and solutions

As a small but not quite minimal example, let's consider a DocStrip file called `hw.dtx`. The entire file can be viewed online as part of the github repository for this article: `github.com/leingang/tugboat-docstrip`.

3.1 Batch header

Listing 2 shows the beginning of `hw.dtx`, which loads `docstrip.tex` and declares the `\generate` batch. It's surrounded with `driver` guards. Since no generated file sets the `driver` option, this block is not written to *any* file.

We generate four files:

Listing 2: The header block of `hw.dtx`, declaring `\generate` batch

```
%<*driver>
\input docstrip.tex
\askforoverwritefalse
\generate{
    \file{\jobname.qns.tex}{\from{\jobname.dtx}{questions}}
    \file{\jobname.ans.tex}{\from{\jobname.dtx}{questions,answers}}
    \file{\jobname.sol.tex}{\from{\jobname.dtx}{questions,solutions}}
    \file{\jobname.bib}{\from{\jobname.dtx}{bib}}
}
\endbatchfile
%</driver>
```

- `hw.qns.tex`, which sets the option `questions`. This document will be the prepared questions sheet for the instructor to distribute to the students.
- `hw.ans.tex`, which sets the options `questions` and `answers`. This file will be distributed to students as LaTeX source, so that they can fill in their answers without having to create their own file from scratch.
- `hw.sol.tex`, which sets the options `questions` and `solutions`. This can be published once the assignment is collected and graded.
- `hw.bib`, which only sets the `bib` option. This is a BibTeX file that can be included in any of the LaTeX files. If the assignment needs to be copied with only a few changes, such as the year and the due date, only one file must be copied from the old directory to the new.

Lines 12–89 of `hw.dtx` are delimited with the `<questions>` guards and enclose an entire LaTeX document from `\documentclass{article}` through to `\end{document}`. Lines 90 and onward (not shown in this article) are delimited with `<bib>` and comprise the complete `hw.bib` file.

3.2 Using guards to conditionally include text

Listing 3 (following page) shows an excerpt of `hw.dtx` that declares a question. Notice that the `hint` environment is surrounded by a guard with a compound boolean expression `<!answers&!solutions>`. The effect is that the `hint` is shown when the `questions` option is selected, but `answers` and `solutions` are *not* selected; that is, only in the `hw.qns.tex` file. The resulting block that is written to the questions file is shown in Listing 4.

Comment lines that begin with a single `%` are stripped from the input and do not get printed to any output file. But comment lines beginning with *two* `%` characters remain. So the comment on line 71 of `hw.dtx` is retained in the `hw.ans.tex` file (Listing 5). It is a note to the student where to write their answer. Finally, the `solution` environment and subsequent commentary paragraph are written to the `hw.sol.tex` file (Listing 6).

3.3 Using guards to conditionally define environments

The implementation of the environments `question`, `answer`, `hint`, and `solutions` have to be set up in the preambles of the generated LaTeX files (or in packages used by them). But guards can be used in the preambles too. In this way, we can conditionally style the document.

For instance, I prefer that the question text be upright in the questions file and italicized in the answers/solutions file. This is accomplished in Listing 7. Line 34 is written to the answers and solutions file, and overrides line 33. The preamble of the questions file defines `question` under the `definition` theorem style, with bold header and upright body font. But in the answers and solutions file, the `plain` theorem style is in force, so `question` sets its body in italic.

Listing 7: An excerpt of `hw.dtx` (lines 35–39) showing conditional styling of the `question` environment

```
\usepackage{amsthm}
\usepackage{amssymb}
\theoremstyle{definition}
%<answers|solutions>\theoremstyle{plain}
\newtheorem{question}{Question}
```

3.4 Advanced tricks

If you look in the full `hw.dtx` file online, you'll see a few more automatic variations with DocStrip:

- The document title is specified in `hw.qns.tex`. In `hw.ans.tex`, the text '`Answers to `' is prepended to the title (using the `\preto` command from the `etoolbox` package). In `hw.sol.tex`, the phrase '`Solutions to `' is prepended. In

Listing 3: An excerpt of `hw.dtx` declaring a question (the question is from [7])

```
\begin{question}
    \cite[Exercise 6.6]{Scheinerman}.
    Disprove: if $p$ is prime, then $2^p-1$ is also prime.
\end{question}
%<*!answers&!solutions>
\begin{hint}
    All you need is one counterexample.  Guess and check, and be persistent.
\end{hint}
%</!answers&!solutions>
%<*answers>
%% Student: put your answer between the next two lines.
\begin{answer}
\end{answer}
%</answers>
%<*solutions>
\begin{solution}
    Let $p=11$.  Then $p$ is prime.  But $2^p-1 = 2^{11}-1 = 2047 = 23 \times 89$.
    So the statement is false.
\end{solution}
A prime number that is equal to $2^n-1$ for some $n$ is called a \emph{Mersenne
Prime}.  Examples of Mersenne primes are $3 = 2^2 - 1$ and $127 = 2^7-1$.  It is
unknown whether the number of Mersenne primes is finite!
%</solutions>
```

Listing 4: The generated block in `hw.qns.tex`

```
\begin{question}
    \cite[Exercise 6.6]{Scheinerman}.
    Disprove: if $p$ is prime, then $2^p-1$ is also prime.
\end{question}
\begin{hint}
    All you need is one counterexample.  Guess and check, and be persistent.
\end{hint}
```

this way, the original title only needs to be put in one place.

- The document author is set differently in the different document variants. In `hw.qns.tex` and `hw.sol.tex`, the author is the professor. In `hw.ans.tex`, the author is set to a generic student name, and `\LaTeXWarning` is used to remind the student to change the generic name to their own name.

4 Comparing alternatives

The problem of maintaining the sources for different, but closely related documents in the same file, and specifying which documents are to be typeset at the time of compilation, has been encountered by users before, for instance on the Stack Exchange network [1, 4, 6]. Let's consider some of the alternatives in the context of the example use case from Section 3.

4.1 Separate files

In this strawman workflow, separate, nearly identical files are kept side-by-side. Any correction (to a problem, for instance) requires three files to be updated. Further, copying a question to another assignment requires copying from three different files to three different files. This setup is ripe for inconsistency and headache.

4.2 Option setting in pure (LA)TEX

In this workflow, certain (LA)TEX booleans are defined and set, and code is conditionally executed depending on those booleans. The booleans can be set by adding TEX code on the command line as optional arguments to the executable (e.g., *pdflatex*). Or, a shell file can be created which sets options, then inputs a common master file.

When options are set within the document, the `\jobname` is the same independent of the options

Listing 5: The generated block in `hw.ans.tex`

```
\begin{question}
    \cite[Exercise 6.6]{Scheinerman}.
    Disprove: if $p$ is prime, then $2^p-1$ is also prime.
\end{question}
%% Student: put your answer between the next two lines.
\begin{answer}
\end{answer}
```

Listing 6: The generated block in `hw.sol.tex`

```
\begin{question}
    \cite[Exercise 6.6]{Scheinerman}.
    Disprove: if $p$ is prime, then $2^p-1$ is also prime.
\end{question}
\begin{solution}
    Let $p=11$.  Then $p$ is prime.  But $2^p-1 = 2^{11}-1 = 2047 = 23 \times 89$.
    So the statement is false.
\end{solution}
A prime number that is equal to $2^n-1$ for some $n$ is called a \emph{Mersenne
Prime}.  Examples of Mersenne primes are $3 = 2^2 - 1$ and $127 = 2^7-1$.  It is
unknown whether the number of Mersenne primes is finite!
```

set. In our homework example, this would mean the problems file and solutions file would both end up named `hw.pdf`. Not only does this mean that the problem set and solutions PDFs cannot inhabit the same directory at the same time, one can be mistaken for another. Imagine the poor professor distributing what he thought was the questions-only PDF, only to realize that he had instead shared all the solutions!

With shell files, the `\jobname` is different for each document variant, avoiding the possibility of such a mistake. The `\jobname` can also be set on the command line when invoking (LA)TEX. But either way, extra files are needed to support this workflow.

Also, in the context of a homework assignment, none of these methods allow the distribution of an answer template as a LATEX source file. If the solutions are in the master TEX file, and conditionally typeset depending on options, they still remain the source.

4.3 Symlinks

In this workflow, a single TEX file is created, and each variant document is a symbolic link (or symlink) to the original file with a different file name. The operating system treats a symlink to a file as a different name for that file. Editing one of these “files” affects the contents referenced by the file and all of its symlinks. But the `\jobname` is determined by the file name, so it can be tested in order to conditionally execute certain code.

This avoids the colliding output file issue of command-line arguments, and is more lightweight than shell files. A new variant just requires a new symlink. It has the same disadvantages of the master-plus-shell files workflow, though. Code cannot be stripped out of the master file for a template, only gobbled and discarded. Not every operating system and file system has this capability, either.

4.4 Other preprocessors

DocStrip functions as a *preprocessor* — it converts one source file to another (or several others). There are other tools for this job, among them GPP and `m4`. Using another program requires installing, learning, and maintaining another program, whereas DocStrip is available wherever TEX is.

The DocStrip method described here is operating system independent. It is secure in that each document variant gets a distinct `\jobname` and output file name. It is lightweight in that only one DocStrip file needs to be created for every bundle of documents needed, and no programming other than TEX is necessary.

4.5 Disadvantages of the DocStrip method

One drawback of this method is that it requires an extra TEX run. First, the DocStrip file is compiled, extracting the various TEX files. Then each of them must be compiled. A bit of programming can automate this process (as well as decide when certain files don't need to be recompiled), as we'll describe in the next section.

Not every TEX editor can be used for DocStrip-files. In particular, WYSIWYG or WYSIWYM editors such as LYX and TEXmacs expect the source file to be a regular (LA)TEX file. But any editor designed to operate on text files can be configured for DocStrip. Some of the most popular TEX editors (for example, TEXShop, TEXworks, TEXstudio, AUCTEX) support DocStrip out of the box, as does Visual Studio Code.

Another, more uncomfortable drawback is that TEX errors are only discovered when the generated TEX files are compiled. The line numbers reported by TEX at the time of the error are different from the line numbers in the DocStrip file. So diagnosing errors needs to be done without navigating to specific line numbers. Rather, the token list before the error can be used for a search string.

Finally, errors in the first run of TEX (when DocStrip is extracting) can arise from unbalanced guard modifiers, e.g., a `%<*solutions>` line with no closing `%</solutions>`. These are hard to isolate since DocStrip does not log its processing with much context, and the error isn't discovered until the end of the file is reached.. I have been able to find these through a combination of retracing my steps, and selectively commenting out blocks.

5 Automation

To recap, this workflow requires editing a DocStrip file marked up as in Section 3, compiling that DocStrip file to generate separate TEX files, then compiling the desired TEX file. The first run can be done in any TEX engine, because the batch declaration header (and `docstrip.tex`) is in core TEX. The second set of compilations require whatever engine your destination documents require.

The *latexmk* program [2] works like *make* for TEX projects. It examines a TEX file to find dependencies, watches for warnings about re-running LATEX, and runs the necessary commands to get the entire document stable. *latexmk* is written in Perl, distributed with TEX Live and MiKTEX, and actively maintained.

A one-line Unix command that does both of these is:

```
   tex foo.dtx && latexmk
```

This processes `foo.dtx` and, upon success of that command, runs *latexmk*. Without file arguments, *latexmk* looks for any TEX files in the current working directory and makes them. Any command-line options to *latexmk* (notably, `-pdf` to make sure the *pdftex* engine is chosen, `-pdfxe` to ensure *xelatex*, or `-pdflua` to ensure *lualatex*) will be passed when making each TEX file.

The process can be further automated and integrated into various TEX editors. I have written a `.latexmkrc` (configuration file for *latexmk*) that looks for DocStrip files in the argument list, and when found, parses their log files for names of generated files to make automatically. With this configuration, `latexmk foo.dtx` will take care of all generated files in one fell swoop.

Any editor that can run a program can be configured to run *latexmk*. For instance, I have written a TEXShop "engine" script that wraps around *latexmk* so configured. I edit the DocStrip and press Command-T once. I open one of the generated files in "preview" mode, which gives the PDF window but not the TEX window (we won't be editing the generated file directly, so we don't need it). The preview window updates each time the underlying file is changed.

I have also gotten this workflow to succeed in Visual Studio Code with the LATEX Workshop [3] These script files and corresponding documentation are in the github repository.

6 Conclusion

I will continue to maintain and update the github repository referenced above. If you would like to try this method, and find that additional documentation would be useful, I will be happy to include it.

References

[1] Caramdir. Passing parameters to a document, 2010. `tex.stackexchange.com/q/1492`

[2] J. Collins, E. McLean, D. J. Musliner. latexmk — fully automated LATEX document generation, 2019. `ctan.org/pkg/latexmk`

[3] J. Lelong, T. Tamura, et al. Visual Studio Code LATEX Workshop Extension, 2020. `github.com/James-Yu/LaTeX-Workshop`

[4] meduz. What Makefile to produce slides and handouts [in] a common file?, 2014. `tex.stackexchange.com/q/170542`

[5] F. Mittelbach, D. Duchier, et al. The DocStrip program, 2006. `ctan.org/pkg/docstrip`

[6] reprogrammer. Passing command-line arguments to LATEX document, 2009. `stackoverflow.com/q/1465665`

[7] E. R. Scheinerman. *Mathematics: A Discrete Introduction.* Thomson Brooks/Cole, Belmont, MA, 2nd edition, 2005.

◇ Matthew Leingang
New York University
`leingang (at) nyu dot edu`
`www.cims.nyu.edu/~leingang/`