How to Write a Shiny Paper☆

Thomas de Graaff

Department of Spatial Economics, Vrije Universiteit

Abstract

This papers explains what tools you need to write a paper that (in this case) looks like an Elsevier article. The underlying theory, however, of why certain tools are better than others is applicable to almost all types of output (e.g., theses, papers, presentations, posters, and articles in any other format). With this paper I hope that (advanced) students are willing to give the tools discussed a try—even if the learning curve in the beginning might be quite steep.

Keywords: Automation, workflow, text files, LaTeX, citation manager

1. Why bother?

Well, first of all, you obviously want to write a shiny paper; a paper that looks really nice, is easy to read, and where each of the elements—including references, mathematics, tables and figures—have a consistent look. And the point is: this is far more difficult (though possible) with conventional—read Word—tools. Secondly, and more persuasive perhaps, conventional proprietary tools do not lend themselves easily to customisation and automisation. For instance, it is rather difficult to directly import Stata output in tables in Word; it always has to go indirectly, or using Excel or using Rich Text Format (rtf).

Although this tutorial advocates the use of some particular tools, it actually does not matter that much as long as they are able to use plain text files. Plain text files namely have a number of advantages over other type of files; they are transparent (you can always read and edit them), portable (they are the same on each system) and they are not likely to become obsolete in the coming decades. And a lot of files or even tools just consists of plain text files, such as HTML code, most open source tools and Stata 'do' files. Most proprietary tools have formats that are not made of plain text, such as Word, Excel, or powerpoint (as well as the Stata data format by the way). This makes them not only less flexible but the risk of getting stuck in an obsolete data format[1] is nontrivial, while output is not similar on each operation system or even across versions.

2. Managing workflow

It is difficult to define what 'good workflow' is, but it certainly has to satisfy each of the following elements to some degree:[2]

1. Transparency: A good workflow organizes the elements of the project logically and clearly, to make it easy for an observer (including yourself) to understand how the pieces come together.
2. Maintainability: A good workflow makes it easy to modify and adapt the project. Standardized script names and good commenting practices are key here.
3. Modularity: A good workflow encapsulates discrete tasks into separate components (e.g. scripts), so that it's always clear where modifications need to be made (and only made in one place), and components are re-usable for other projects.
4. Portability: A good workflow makes it easy to move the project to another system, or hand it over to another person to work on, in such a way that it can still easily be run elsewhere. (By using relative (not absolute) pathnames is one example.)
5. Reproducibility: A good workflow makes it easy for you, or others, to reproduce your results.
6. Efficiency: A good workflow saves you time, by making it easier to work on the project, and by automating as much of the process as possible.

3. LaTeX instead of 'What You See Is What You Get' tools

Word and Powerpoint are usually described as 'What You See Is What You Get' (WYSIWYG) tools. But what you get is sometimes below par. That's why some advocate the use of 'What You Mean Is What You Get' (WYMIWYG) tools. And these tools usually apply to markup languages. The most famous of these languages is HTML,

Email address: t.de.graaff@vu.nl (Thomas de Graaff)
URL: www.thomasdegraaff.net (Thomas de Graaff)

[1] The case of the late but much beloved wordprocessor 'Word Perfect' comes to mind.

[2] Almost directly taken from this blog; another wonderful source is Healy, 2013.

which is just a bunch of code describing how text *should* look like on screen.

LaTeX is a markup language as well, that just consists of code describing how a text should look like (on paper, screen or beamer). Instead of a word processing program, LaTeX is a typesetting program that determines for itself how words, sentences, paragraphs, etcetera, are placed on paper, slide or pdf file. Actually, LaTeX is a program (or compiler) that is fed by an input text file. Novice users find this most difficult; everything should be more or less coded. However, this allows for great flexibility, transparency, portability, and even efficiency. The same code can namely be used for papers, presentations and posters alike.

Where LaTeX truly shines is in the scientific domain.[3] If one wants to consistently typeset text, figures, tables, references and especially equations, nothing outperforms LaTeX.[4] And in the end, that is what you want to do: write a nice shiny paper.

3.1. I got it, and now what?

There are numerous tutorials on the web, but I recommend to first take a look at the latex project website, which comes with a whole bunch of tutorials (and yes, they are all free).

Without really going into the nitty gritty of this language, let's write our quintessential 'hello world' program.

```
% hello.tex - A very simple LaTeX example!
\documentclass{article}
\begin{document}
Hello World!
\end{document}
```

with the following output (but then as a separate document):

Hello World!

Okay, this perhaps doesn't look like much, but it actually contains quite some interesting elements. First, lines starting with a % sign are used as comments. Second, you first have to specify what kind of text you are typesetting (in this case an article). And finally, every element needs a 'begin' and an 'end' (just as we all do). In this case, the whole document needs to be specified. Things start to heat up a bit, when you add formulas such as when we add the following lines to our simple example

```
You know, I just discovered that:
\begin{equation}
E = mc^2
\end{equation}
```

with the following output:

You know, I just discovered that:

$$E = mc^2 \tag{1}$$

3.2. And what about figures, diagrams and graphs?

Fortunately, figures, diagrams and graphs are much easier to handle in LaTeX than tables. And, as long as you want to draw something in a schematic way, you can even make your own diagrams and graphs using a couple of additional packages!

3.2.1. Figures

Because the output of LaTeX is already in pdf format it can easily incorporate figures from many format, such as the printer languages .pdf and .ps, but as well .jpeg and others. For example, this command:

```
\begin{figure}[h!]
\center
\includegraphics[width=0.3\textwidth]
{ligatures_latex} \caption{Correct
use of ligatures in \LaTeX (Source:
\href{http://nitens.org/ taraborelli/latex}
{Taborelli})}
\end{figure}
```

produces Figure (1):

Figure 1: Correct use of ligatures in LaTeX (Source: Taborelli)

The command `h!` states that the figure should very explicitly (the `!` command) be placed about here (the `h` command).

When captions are incuded they are correctly numbered and size and position of the figure (even across columns) can easily be adjusted.

4. Always use a citation manager

The first thing I do recommend to all students (and researchers) is to use a citation manager. Even when using applications such as Microsoft Word (the horror...). This is one tool that immediately delivers in terms of efficiency. Even if initially setting up a database with references might be time-consuming it still pays off in later stages. The point is that you only once have to type in a reference within a database. Later on you can always use this reference within the text and the reference or bibliography list should be automatically generated. And there are even reference managers that can import folders with articles in pdf format and automatically generate a reference database.

[3] But not only given the nice examples on this website.

[4] LaTeX itself is just a bunch of macros around TeX, a typesetting program devised by the notorious computer scientist Donald Knuth. LaTeX was created by Leslie Lamport in 1984 and TeXwas created in 1978. Yes, it is that old.

Within the LaTeX environment, the application BibTeX is usually used. There are separate editors for this system, but the easiest way is using a more generic reference manager (the free Mendeley application comes to mind) that can write files to .bib files (the BibTeX format) or incorporate references directly within Microsoft Word. In this way, your remain as flexible as possible.

5. From here . . . into the abyss

Right. so we know now how to write a shiny paper with tables, references and perhaps even customated diagrams. Would that be it? No, it is just the start of a completely automated, customated, transparent and (hopefully) more efficient workflow. For that you need a set of tools that are complementary to each other. And fortunately, most of them are. However, choose your tools carefully. All of them are costly in terms of learning curves and there is only so much time available to you. Having said that, there are a couple of choices you might want to make eventually, of which the following three are probably most important.

First of all, the choice of editor. Of course, TexShop or TeXNicCenter are nice editors, but are mostly suited for LaTeX. In the end, you might want to use a more versatile editor. One editor capable of editing R scripts, stata do files, html code, etcetera. All with proper highlighting and a consistent set of shortcuts. I am not saying which editor you should use. Too many have found their demise at the editor war. There are many good (and free) ones out there. And even the ones you have to pay for are well worth the money. Pick up and stick to it for some time. For OsX there is now a free version of Textmate, which is quite good. And cross platforms, sublime is another editor well worth looking into. If you really want to go nerdy, opt for Emacs of Vi(m). Longstanding, very capable (and fast) editors. However, be aware of a steep learning curve.

Second, as already hinted upon, the combination of LaTeX with statistical programs such as Stata and R is a killer app. If one has to choose, I would advise you to invest in R. Stata is more an econometric package, while R is much more statistical. And the userbase of the latter is much larger than the former, ensuring that with R most problems dealing with statistics (or even GIS) have already been dealt with. Moreover, there is a very nice editor, Rstudio, able to integrate them both. Heck, you can even weave LaTeX and R code together (called Sweave or Knitr) with mindblowing results. This comes rather close to what Donald Knuth coined literate programming: the simultaneous generation of the output of both text as analysis.

Third, and finally, think very carefully about backup programs. You know, you never need it until you really need it. The first advise is to always use an external disk with a backup application (such as Time Machine on OsX). Secondly, having some of your applications (especially your code, and thus text, files) somewhere in the cloud has additional benefits. Apart from a backup advantage, it also facilitates working together on a project. The best known of these applications is Dropbox, although cooperation using this is still a bit cumbersome. Another, more nerdy, example is using Git, which is a full blown distributed revision control and source code management system, created by Linus Torvards (that guy from Linux). It takes some time to get use to, but in the end it grows on you and becomes not only easier to use, but makes you in the end even more efficient.

As a last example about complementarity, the before mentioned Rstudio application does not only edit LaTeX and R, but is able as well to set up a Git repository system. In the end, it comes all together.

6. Some final thoughts

This might all seem a bit daunting. The best advise for novices would be: start one step at a time ("A journey of a thousand miles begins with a single step", Lao-tzu). In this case this simply means that you start with a template file (such as that from the Elvesier article class), incorporate or copy in some text, divide it by section and subsection headers and see whether it works.

Granted, the learning curves of most of these tools are rather steep (except for that of using citation managers—you really should use them). However, it does pay-off in the end. And most of these tools have the fortunate feature to be very *complementary* to each other. A good example is making slides. There is a package out there called beamer, which allows you to create slides from your text. This basically means, that all your codes used for your paper can be used again for your slides (within a LaTeX context, so equations won't fly all over the slide as sometimes happens with more conventionals presentation programs). If you know one application really well, it becomes easier to understand others or even use the same commands. And there are quite a few applications or languages out there that work with text files and are very handy for a researcher—HTML, Markdown, Latex, scripting languages from Stata and R and even languages as Ruby and Python come to mind. The latter, for example, is the native language of ArcGis scripts.

Obviously, this paper is way too short to do justice to them. However, starting with LaTeX would be a good introduction to a more customated and automated workflow.