Reproducible Research II: sharing files

LPO 9951 | Fall 2015

Contents

PURPOSE

In the last lecture on reproducible research, we discussed version control through git. Git works better with some types of file formats than others (e.g., plain text vs. MS Word). Today we will discuss a few of these not only in the service of using git, but with the purpose of improving how you can collaborate with others and share your work. We'll also discuss attributions and ownership.

The common thread

Despite their seemingly different approaches to structuring text, the following syntaxes share two primary characteristics:

1. They are structured and saved using plain text files
2. They follow a *WYSIWYM* rather than *WYSIWYG* philosophy

Plain text

The simple MS Word `.doc` or `.docx` is a single file, right? Yes and no. The reason non-PC systems sometimes have trouble with MS Word documents is (1) they are saved in a proprietary format, and (2) those files are better thought of as file systems rather than as a single file. In other words, your MS Word document isn't simply a record of the words on your screen, but also of a large number of format and system settings. This file structure is why, if you have tried, GitHub doesn't display Windows documents that you may have pushed to it.

Git works best with plain text (ASCII) files. There are a limited number of characters available, but for most of your scripting, you'll be fine (do you really need that umlaut in your variable names?).

WYSIWYG vs WYSIWYM

Huh? WYSIWYG and WYSIWYM, respectively, stand for:

- **W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et
- **W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **M**ean

It's the last word, *get* vs *mean*, that makes the difference. MS Word is WYSIWYG. The content and the formatting are combined. HTML, for example, is WYSIWYM, meaning that the formatting is called only at the end, as the final document is assembled. In reality the difference between WYSIWYG and WYSIWYM isn't binary, either/or. Rather the *get* and *mean* exist on ends of a spectrum. On the far *get* side, MS Word. On the far *mean* side, LaTeX. Each has its benefits and drawbacks. But for reproducible documents that facilitate version control and collaborative sharing, WYSIWYM is the rule.

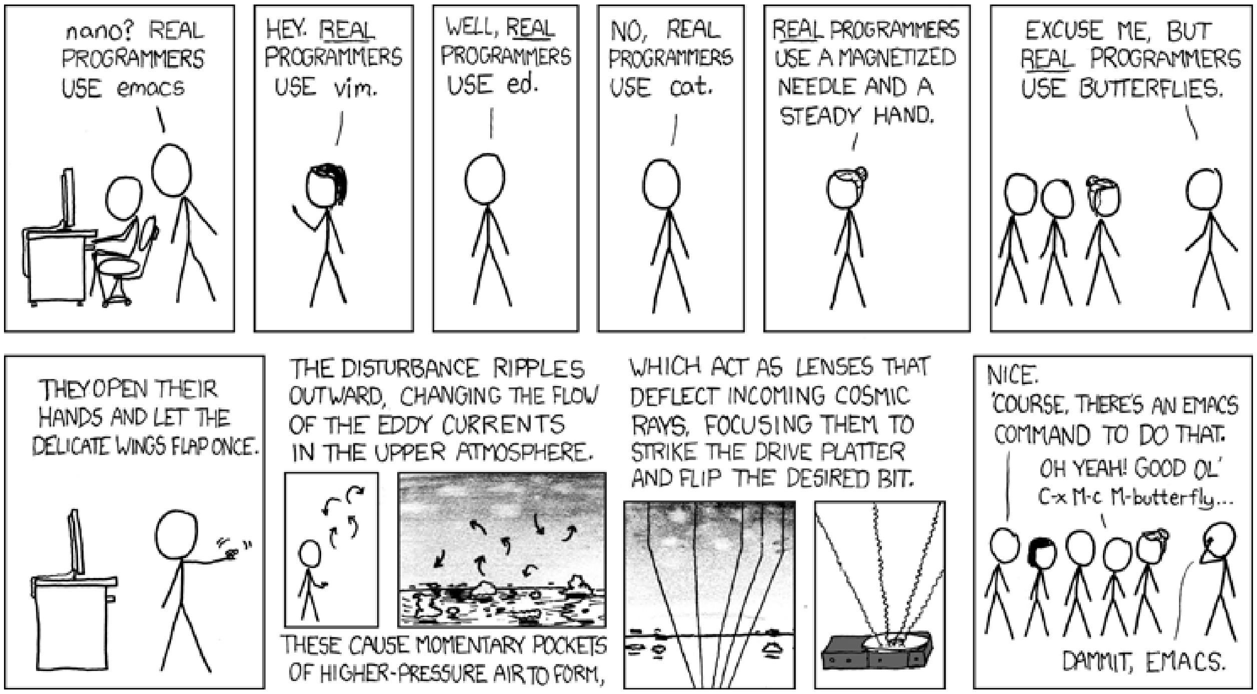
Writing

There are a number of syntaxes that you can use when producing text. We'll go through each below, roughly from least to most complex.

Plain text

You can always write in plain text. Your Stata .do files are in plain text; the only difference is instead of saving them with a `.txt` ending, we save them with `.do` so that Stata knows what they are and how to treat them.

Your computer likely has a text editor already installed. Some allow you write in rich text. Turn that off if you want pure plain text. More sophisticated editors such as emacs/aquamacs and vim offer powerful functionality above and beyond those found in a simple text editor. Now that the editor wars are largely over, you can pick anything you want and still be a "real" programmer.


nano? REAL PROGRAMMERS USE emacs
HEY. REAL PROGRAMMERS USE vim.
WELL, REAL PROGRAMMERS USE ed.
NO, REAL PROGRAMMERS USE cat.
REAL PROGRAMMERS USE A MAGNETIZED NEEDLE AND A STEADY HAND.
EXCUSE ME, BUT REAL PROGRAMMERS USE BUTTERFLIES.
THEY OPEN THEIR HANDS AND LET THE DELICATE WINGS FLAP ONCE.
THE DISTURBANCE RIPPLES OUTWARD, CHANGING THE FLOW OF THE EDDY CURRENTS IN THE UPPER ATMOSPHERE.
THESE CAUSE MOMENTARY POCKETS OF HIGHER-PRESSURE AIR TO FORM,
WHICH ACT AS LENSES THAT DEFLECT INCOMING COSMIC RAYS, FOCUSING THEM TO STRIKE THE DRIVE PLATTER AND FLIP THE DESIRED BIT.
NICE.
'COURSE, THERE'S AN EMACS COMMAND TO DO THAT.
OH YEAH! GOOD OL' C-x M-c M-butterfly...
DAMMIT, EMACS.

[Real programmers set the universal constants at the start such that the universe evolves to contain the disk with the data they want.] Credit: xkcd

For quick notes, embrace plain text. Do you really need everything formatted just so for something that only you or a small number of people will see? Probably not. So stop focusing on the margins and just type!

Markdown

Okay, sometimes you want a little formatting. For that you have Markdown. Originated by John Gruber in 2004,

> Markdown is a text-to-HTML conversion tool for web writers. Markdown allows you to write using an easy-to-read, easy-to-write plain text format, then convert it to structurally valid XHTML (or HTML).

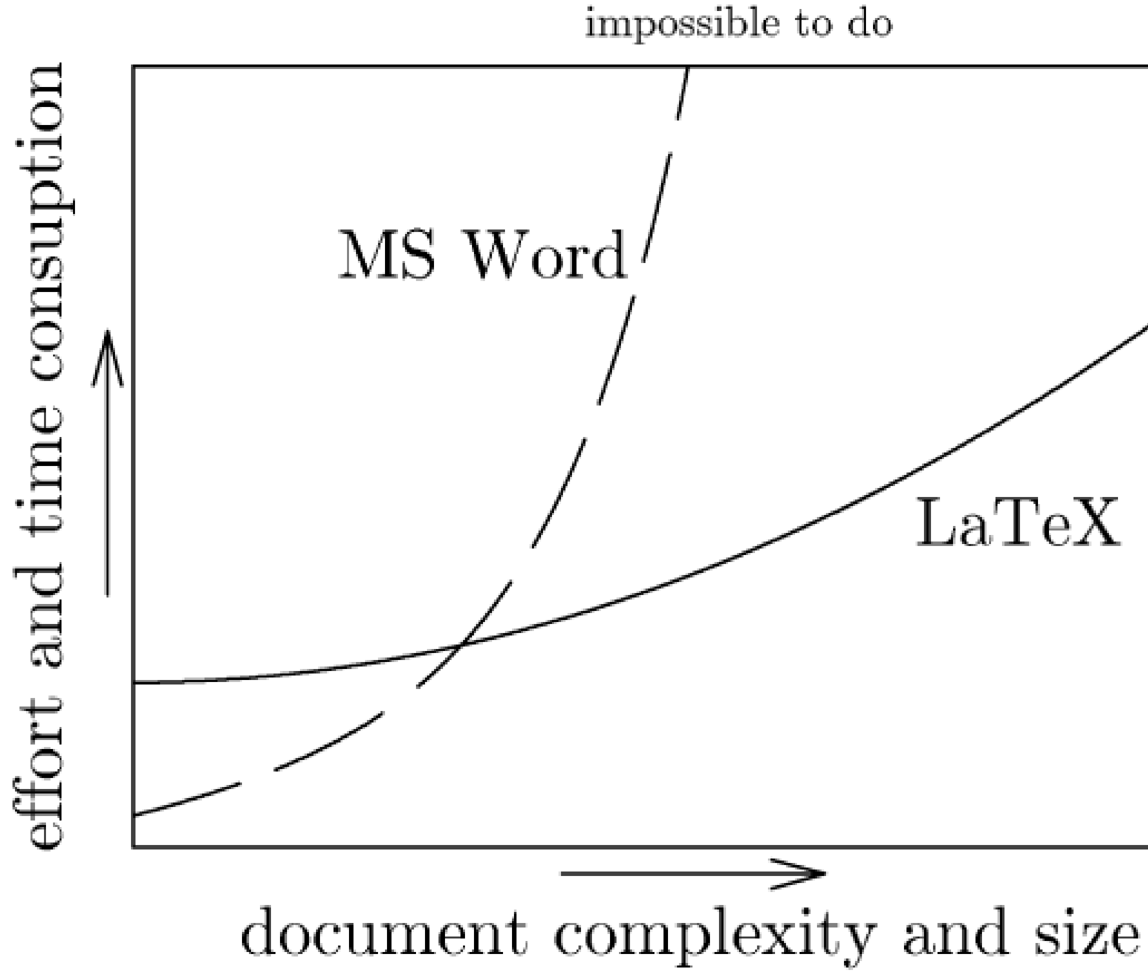
Source: http://daringfireball.net/projects/markdown/

In the years since, Markdown has become one of the default syntaxes for web-based documents. GitHub makes extensive use of Markdown, both in its README files (hence the `.md` ending) and on its website. This course website was compiled using Markdown. Though originally intended to convert plain text to HTML, most markdown editors will export PDF documents as well.

To use Markdown on your computer, you will need to download and save it per instructions found online. But lucky for us, we can play with Markdown in our web browser via the site Dillinger.

LaTeX

Let's get this out of the way: LaTeX is a pain. Why even bother? This chart pretty much sums it up:



Credit: Marko Pinteric

So for small documents, LaTeX isn't worth it. But as your documents become more complex or if you wish to follow the philosophy of literate programming, then learning LaTeX is on your horizon. TeX and its variants hew closely to the WYSIWYM ideal. Formatting requires tags around the relevant text and the loading of special packages for extended features. The end result of an uncompiled `.tex` document is almost unreadable, but it succeeds at giving the writer almost complete control of the formatting process while separating it from the content.

Discussing the finer points of LaTeX are outside of the scope of this course. If you want to try it, you'll need to load a TeX engine on your computer. Luckily, people have put together packages that not only install TeX on your machine, but also include editors to make your life a little easier. I've also put together a getting started with LaTeX document that you may find helpful. If nothing else, it will give you links to download the proper TeX distribution for your machine.

To give you just a quick idea of how LaTeX works, visit https://tex.mendelu.cz/en/ and paste the following code in the window (be sure to select *3* passes before compiling):

```
\documentclass{article}
\begin{document}

\begin{center}
{\Huge Title}
\end{center}

\section{First Section}

This is the first section.

\subsection{First subsection}

This is the first subsection.

Here's another paragraph, this time indented.

\subsection{Second subsection}

\begin{enumerate}
\item one
\item two
\item three
\begin{itemize}
\item three.i
\item three.ii
\item three.iii
\end{itemize}
\end{enumerate}

\section{Second Section}

See Table \ref{tab:one} below.

\begin{table}[!h]
\caption{A table}
\label{tab:one}
\begin{tabular}{lrrr}
\hline
Letter & Number & Greek: Upper & Greek: Lower \\
\hline
A & 1 &  & \alpha \\
B & 2 &  & \beta \\
C & 3 & \Gamma & \gamma \\
\hline
\end{tabular}
\end{table}

\end{document}
```

Equations

Even if you decide that LaTeX is more trouble than it's worth, you would still do well to learn how to write equations in it. All those fancy math books with the well-formatted equations? LaTeX. MS Word has an equation editor, but it can be frustrating. Take the time to learn the syntax and you'll be able to format clean equations for your presentations and papers (even if you choose to import them as image files).

With LaTeX syntax, this

```
f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(x - \mu)^2}{2\sigma^2}}
```

becomes

$$f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(x - \mu)^2}{2\sigma^2}}$$

The good news is that you can use this syntax in Markdown with MathJax. Again, this might be overkill for a quick note, but when your equation need to be correct, knowing how this syntax is structured is invaluable. There are a number of online equation editors that will let you download your finished equation as an image file. Here is one.

References

At this point, you likely have some system for managing your references. Many good standalone programs and browser extensions are available such as Mendeley and Zotero. If you choose to use LaTeX, its reference system uses BibTeX to add your inline citations and produce a nicely formatted bibliography. Like our discussion about LaTeX in general, BibTeX gives the most flexibility but at the highest start up cost.

`*.bib` files

Even if you don't choose to use BibTeX, you should know about its basic database file type, the `*.bib` file. All other reference management systems will import and export bib files. Why should you care? Because a bib file is simply a text file that organizes references by their constitutive parts. For example:

```
@article{knuth1984literate,
  title={Literate Programming},
  author={Knuth, Donald Ervin},
  journal={The Computer Journal},
  volume={27},
  number={2},
  pages={97--111},
  year={1984},
  doi={10.1093/comjnl/27.2.97}
}
```

With the `@` sign, the entry tells us what type of reference it is. Immediately following the open bracket is the key, which many programs use to call the reference. Everything else together comprises the relevant parts of the citation.

What's missing? The format. We don't need it. That can be set later. The important point is that our citation is ready to be used and can be shared easily with someone else. Even a bib file with nearly a thousand references (my general archive file) is only around 2.5 MB.

Graphics

You will produce a lot of plots in your research. You will likely make a poster or two for conferences. People may ask you for a copy of your cool graph or even your poster. Two important things to keep in mind: the format of the file and making sure you get attribution.

In terms of formatting, the primary consideration is choosing between raster and vector graphics. I'll briefly discuss the ideas behind each below.

Raster

Raster graphics produce their images by manipulating cells in a grid. Each cell is given a certain value in terms of color/brightness and the sum total of cells produces the image. As the number of cells increase, so does the potential resolution as well as the file size.

The inherent problem with raster graphics is that they don't scale up. So you if save a graph in the `.jpg` format and later need to enlarge it, you'll notice some pixelation. This is because you no longer have enough cells per unit of measure to have the resolution you want. Some programs will try to solve this problem by interpolating new cells based on surrounding values, but the results aren't always great.

Save your headshot in a high quality `.tiff` file and give people a decently sized `.jpg` if they ask. Otherwise, no rasters.

Common types

- `.bmp`
- `.gif`
- `.jpg/.jpeg`
- `.tiff`

Vector

Vector graphics use equations to model the image they produce. Because of this, they are infinitely expandable. They are often smaller in size than similar raster images, though not always. They don't offer the level of detail that raster images can produce, but unless you need photorealistic images, that's not a concern. In our case, vector images might even be clearer as they model the lines of our plots with equations and are therefore only limited in resolution by the display and not the number of sampled grid cells.

One important point to remember with vector graphics: they are often produced in layers. This means that a plot with a thousand data points has at least a thousand layers unless you flatten it. This is one of those situations where the vector image file could balloon in size beyond that of a raster image.

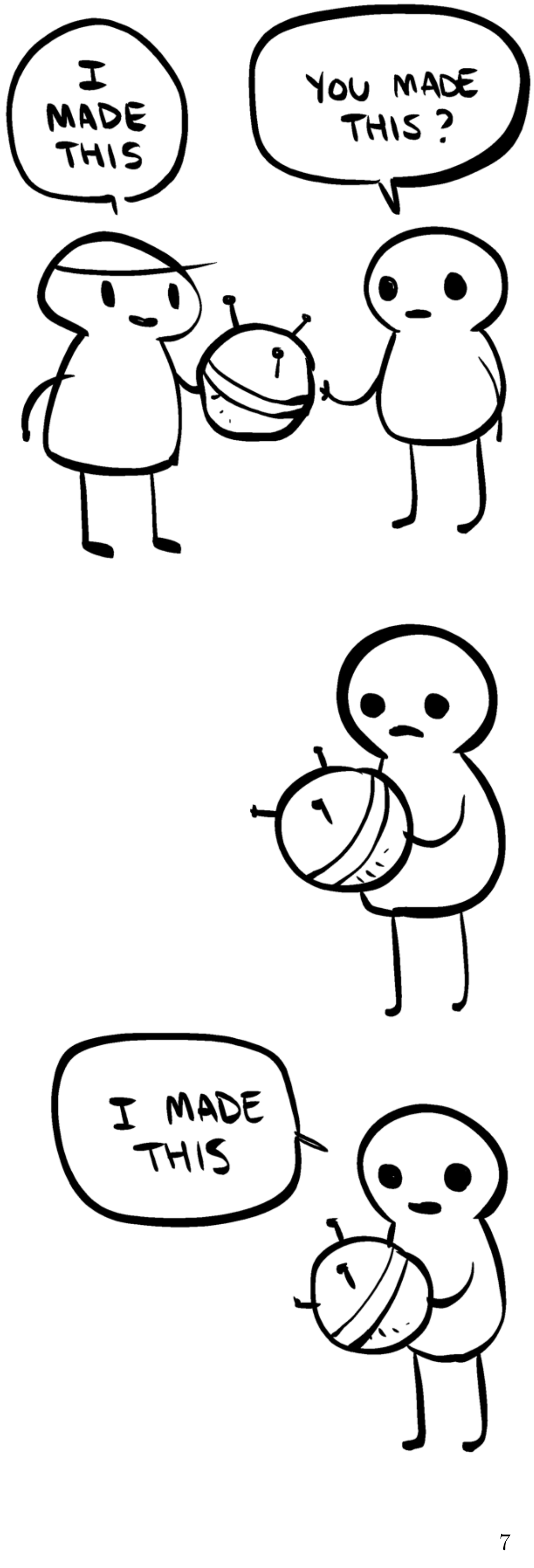
Most of your plots should be in some vector format.

Common types

- `.pdf`
- `.png`
- `.ps/.eps`
- `.svg`

Attribution

The internet is still the wild west in terms of copyright. At worst unscrupulous people will take credit for your work.


I MADE THIS
YOU MADE THIS?
I MADE THIS

Credit: Anthony Clark (Case in point: it took me a while to figure out who created this comic.)

When you feel ready to share your work with the wider world (which includes posting it on your website), be sure to have your name on it. This won't stop someone who really wants to pass off your work as his or her own, but it is a first step. Stealing code is much easier to do since a person can just remove your name from the file. With images, make sure your name is part of the image itself. This will make sure that someone who simply downloads your files or links to it will have cite you, if for no other reason than your name is on it.

All this goes without saying that you should similarly credit the work of others, even if it's just found on the internet.

Ownership

Another aspect of attribution is ownership. When attributing work to someone, you make a claim of ownership on his or her behalf. This means that you have to understand who owns the work and under what conditions. Very rarely will you have a situation in which someone writes a bit of code, builds a dataset, or creates a graphic, and then formally transfers ownership rights to you to do as you will. There will be restrictions concerning when, how, and under what conditions you can use the product. Only because of the free-wheeling nature of the internet and lax understanding of copyright do people feel as free to borrow as they do.

Licenses

Maybe you've seen software described as *open source* or seen some code as falling under the *GNU General Public License v3 (GPL-3)* or *MIT License*. While both of the licenses describe works that you are allowed to copy for free (as in beer), they have different rules regarding how free (as in speech) you are to use them. Many other licenses exist. Some allow you to incorporate the free code into a proprietary program; others say the free code is like ice-nine in that all other code it touches will itself fall under the same license (not so good if you wanted to sell it).

Be aware of the license that other code/images/data fall under when attempting to use. Most snippets and items you find won't come with any license. This does not mean you can just take it! The assumption, unless told otherwise, is that it belongs to the original owner. Ask permission!

As far as your files, decide if and how you want to share. tldrlegal.com will give you a nice summary of various software licenses. GitHub has common license files ready to add to your project.

But before you run off to share everything…

Collaboration and sharing

Are you working on someone else's project? Is some organization funding this project with grant money? Are you using sensitive or restricted data? If the answer to any of these questions is *yes* or even, *I think so?*, then don't share unless you have express permission to do so. And if given permission, make sure you abide by the conditions of that permission.

Final word

Do you have to use git, LaTeX, markdown, etc. or make all of your work free in order to be a good researcher? No. If you do these things, are you thereby a good researcher? Not necessarily. The point in introducing you to these various syntaxes, techniques, and philosophies is not to say that "this is the **one** way to do research" but rather to offer what might be a better way. Pick and choose from these techniques and others as you need, but actively choose something. MS Word and Dropbox may work fine… for now. Or maybe forever. But don't let them be your default simply because you don't know anything else.

Init: 30 August 2015; Updated: 17 September 2019