How to improve your relationship with your future self.

Jake Bowers* Maarten Voors†

June 24, 2016

> Do I contradict myself?
> Very well then I contradict myself,
> (I am large, I contain multitudes.)
> (Whitman 1855)
>
> If you tell the truth, you don't have to remember anything. (Twain 1975)

Memory is tricky. When we really want to remember something we should practice it until it becomes internalized. In fact, learning requires effort. And when we do not practice, we tend to be overconfident in our recall abilities (Koriat and Bjork 2005) and false memories can be implanted.[1] If we cannot count on memory, then how can we do science?

How long does it take from planning a study to publication? And subsequently, how much time passes to the first reproduction of a study? Is three years too short? Is ten years too long? We suspect that few of our colleagues in the social and behavioral sciences conceive of, field, write, and publish a data driven study faster than about three years. We also suspect that, if some other scholar decides to re-examine the analyses of a published study, it will occur after publication. Moreover, this new scholarly activity of learning from one anothers' data and analyses, can occur at anytime, years past the initial publication of the article.[2]

If we cannot count on our memories about why we made such and such a design or analysis decision, then what should we do? How can we minimize regret with our past decisions? How can we improve our relationship with our future self? This essay is a heavily revised and updated version of Bowers (2011b) and provides some suggestions for practices that will make such reproduction occur much more easily and quickly. Specifically, this piece aims to amplify some of what we already ought to know, and to add to some of those ideas given

*jwbowers@illinois.edu, University of Illinois at Urbana-Champaign, Departments of Political Science and Statistics; Fellow, White House Social and Behavioral Sciences Team. Nothing in this paper reflect the official position of the US Government. Many thanks to Mark Fredrickson, Brian Gaines, Kieran Healy, Kevin Quinn and Cara Wong for direct help on this document and to Mika LaVaque-Manty and Ben Hansen for many useful discussions on this topic. The source code for this document may be freely downloaded and modified from https://github.com/jwbowers/workflow.

†maarten.voors@wur.nl, Wageningen University, Development Economics Group. Special thanks, the EGAP Learning Days 2016 participants in Santiago Chile, to the BITSS team and the Department of Economics and Ted Miguel at UC Berkeley where Maarten was a visiting researcher during spring 2016.

[1] See the following site for a nice overview of what we know about memory including the fact that learning requires practice: http://www.spring.org.uk/2012/10/how-memory-works-10-things-most-people-get-wrong.php. On false memory see Wikipedia and linked studies https://en.wikipedia.org/wiki/False_memory.

[2] We note that sometimes this reproduction occurs within the context of classes and learning about statistics and data analysis. Other times, this process is organized more broadly (cite IIIE program, researchers have started to implement psychological experiments using the same protocols across many locations, see Open Science Foundation 2014, 2015, within experimental economics a team did the same, see Camerer et al 2016 Science paper. Another example is to share analysis across many researchers. in a their 'crowdsourcing analysis" Silberzahn and Uhlmann (Nature 2015) asked 29 research teams to run the analysis for a project that looked at the impact of skin color on the probability of getting a foul card in a football match.

current practices, platforms, and possibilities.[3] This essay is an attempt at providing practical advice about how to *do work* such that one complies with such recommendations as a matter of course and, in so doing, can focus personal regret on other bad past decisions but not on decisions about data analysis and the production of scholarly papers.

We organize the paper around a series of homilies that lead to certain concrete actions.

1 Data analysis is computer programming.

All results (numbers, comparisons, tables, figures) should arise from code, not from a series of mouse clicks or copying and pasting. If you wanted to re-create the figure from say 2011 but include a new variable or specification, you should be able to do so with just a few edits to the code rather than knowledge of how you used a pointing device in your graphical user interface all those years ago.

For example, using R (R Development Core Team 2011), a open-source statistical software program, I might specify that a file, called `fig1.pdf` was produced by the following set of commands in a file called `makefig1.R`.[4] Lets look at some annotated R code you could produce:

```
## This file produces a plot relating the explanatory variable to the outcome.
thedata <- read.csv("Data/thedata-15-03-2011.csv") ## Read the data
pdf("fig1.pdf") ## begin writing to the pdf file
please-plot(outcome by explanatory using thedata. red lines please.)
please-add-a-line(using model1)
## Note to self: a quadratic term does not add to the substance
## model2 <- please-fit(outcome by explanatory+explanatory^2 using thedata
## summary(abs(fitted(model1)-fitted(model2)))
dev.off() ## stop writing to the pdf file
```

Now, in the future if you wonder how 'that plot on page 10' was created, you will know: (1) 'that plot' is from a file called `fig1.pdf`and (2) `fig1.pdf`was created in `makefig1.R`. Any time in a future (provided R still exists), changing the figure will require some quick edits of commands already written. In a future where R does not exist, you (or someone else) will at least be able to read the plain text R commands and use them to write code in a new favorite statistical computing language: R scripts are written in plain text, and plain text is a format that will be around as long as computer programmers write computer programs.[5]

Moreover, coding saves time. Often, a lot of time, especially if projects increase (in the number of files, partners, analysis). Manually importing a data file into R, may be effortless. Importing 100 files is another issue, and the costs of each manual action add up quickly.

Also, realize that *file names send messages* to your future self. This means that if you name your files with evocative and descriptive names, your collaborators are less likely to call you at midnight asking for help and

[3] King (1995) and Nagler (1995) were two of the first pieces introducing these kinds of ideas to political scientists. Now, the efforts to encourage transparency and ease of learning from the data and analyses of others has become institutionalized with the DA-RT inititiave `http://www.dartstatement.org/` Lupia and Elman (2014). These ideas are spreading beyond political science as well Freese (2007) and Asendorpf et al. (2013).

[4] The command `please-plot`and some other R functions used in this essay come from the `MayIPleaseDoStatistics` package which emphasizes politeness in data analysis. Functions like `please-plot`can be blocked and more polite versions such as `may-I-please-have-a-plot`can be required using `options(politeness=99)`. We do not recommend setting negative values for the politeness option should you use this package.

[5] Since R is open source, you will also be able to download an old version of R, download an old-fashioned open-source operating system (like Ubuntu 10), and run the old-fashioned statistical computing environment in the old-fashioned operating system in a virtual machine on my new-fashioned actual machine.

you will remove some regret from your future self. For example, if you are studying inequality and protest, you might try naming a file something like `inequality-and-protest-figures.R` instead of `temp9`or `supercalifragilisticexpialidocious`. More on this below. By the way, the extension `.R` tells us and the operating system that the file contains R commands. This part of the filename enables us to quickly search our antique hard drives for files containing R scripts.

Finally, coding helps us to avoid making mistakes. For example, in our example above we may be interested in how many people protest. We may use a data file containing all protests for several years. We can then create a table with the number of protesters by copying the results manually to a document you use to write your paper. In copying we can make mistakes. So a better approach is to ask R to automatically create the table (in the format you like, with horizontal lines, 3 decimals, a title, etc) and output this in whatever file type you need (.pdf, .rtf, etc). Now when we obtain new data (for additional years, or updated figures for existing data) a new table can be created quickly, so we make less mistakes and save time!

Step 1 Code everything that can be coded. If we know the provenance of results, future or current collaborators make less mistakes and can quickly and easily reproduce (and thus change and improve) upon the work.

2 No data analyst is an island for long.

Data analysis involves a long series of decisions. Each decision requires a justification. Some decisions will be too small and technical for inclusion in the published article itself. These need to be documented in the code itself (Nagler 1995). Paragraphs and citations in the publication will justify the most important decisions. So, one must code to communicate with yourself and others. There are two main ways to avoid forgetting the reasons you did something with data: comment your code and tightly link your code with your writing.[6] [I FEEL THIS COMMENT IS FUNNY BUT CONFUSING FOR OUR LATIN AMERICAN READERS PROBABLY]

2.1 Code is communication: Comment code

Comments, unexecuted text inside of a script, are a message to collaborators (including your future self) and other consumers of your work. In the above code chunk, we used comments to explain the lines to readers unfamiliar with R and to remember that we had tried a different specification but decided not to use it because adding the squared term did not really change the substantive story arising from the model.[7] Messages left for your future self (or near-future others) help retrace and justify your decisions as the work moves from seminar paper to conference paper to poster back to paper to dissertation and onwards.

Notice one other benefit of coding for an audience: we learn by teaching. By assuming that others will look at your code, you will be more likely to write clearer code, or perhaps even to think more deeply about what you are doing as you do it.

Comment liberally. Comments are discarded when R runs analysis or LaTeX or pandoc turns plain text into pdf or html files so only those who dig into your work will see them.

2.2 Code to communicate: Literate programming.

> Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do. (Knuth 1984, 97)

[6] One can also try the R command `put-it-in-my-mind(reason,importance="high")` to firmly place a reason for a decision into the mind of the analyst. I myself have not had much luck with this function.

[7] R considers text marked with # as a comment.

Imagine you discover something new (or confirm something old). You produce a nice little report on your work for use in discussions of your working group or as a memo for a web or reviewer appendix. The report itself is a pdf file or some other format which displays page images to ease reading rather than to encourage reanalysis and rewriting. Eventually pieces of that report (tables, graphs, paragraphs) ought to show up in, or at least inform, the publishable paper. Re-creating those analyses by pointing, clicking, copying, or pasting would invite typing error and waste time. Re-creating your arguments justifying your analysis decisions would also waste time.

More importantly, we and others want to know why we did what we did. Such explanations may not be very clear if we have some pages of printed code in one hand and a manuscript in the other. Keep in mind the distinction between the 'source code' of a document (i.e. what computation was required to produce it) and the visible, type-set page image. Page images are great for reading, but not great for reproducing or collaborating. The source code of any document exchanged by the group must be available and executable.[8]

How might one avoid these problems? Literate programming is the practice of weaving code into a document — paragraphs, equations, and diagrams can explain the code, and the code can produce numbers, figures, and tables (and diagrams and even equations and paragraphs). Literate programming is not merely fancy commenting but is about enabling the practice of programming itself to facilitate easy reproduction and communication.

For example, in sec. 1, we suggested that we know where 'that plot on page 10' comes from by making sure we had a `fig1.pdf` file produced from a clearly commented plain text file called something like `makefig1.R`. An even easier solution would be to directly include a chunk of code to produce the figure inside of the paper itself. This paper, for example, was written in plain text using markdown markup with R code chunks to make things like Figure **??**. This combination of markdown and R is called R Markdown.

```
##  Make a scatterplot of Protest by Inequality
par(bty="n",xpd=TRUE,pty="s",tcl=-.25)
with(good.df,plot(gini04,protest05,
                  xlab='Gini Coefficient 2004 (UNDP)',
                  ylab='Proportion Reporting Protest Activities\n(World Values Survey 2005)',
                  cex=.8))
##  Label a few interesting points
with(good.df[c("Brazil","United Kingdom","United States","Sweden","Chile"),],
     text(gini04,protest05,labels=Nation,srt=0,cex=.6,pos=3,offset=.1))
```

Markdown (and LaTeX and HTML) all have ways to cross-reference within a document. For example, by using `\label{fig:giniprot}` in LaTeX or `{#fig:giniprot}` in the pandoc version of markdown, I do not need to keep track of the figure number, nor do extra work when I reorganize the document in response to reviewer suggestions. Nor do I need a separate `makefig1.R`file or `fig1.pdf` file. Tables and other numerical results are also possible to generate within the source code of a scholarly paper. Those who view the code for this essay will see how Table 1 was also generated directly from a regression object.[9]

[8] As of the 2016 version of this paper, this idea is now widespread and made much easier than before via online services for code sharing and collaboration such as GitHub, Open Science Framework and BitBucket, see below.

[9] @beck2010reg inspired this particular presentation of a linear model.

	Coef	Std. Err.	95% CI	
Intercept	43.3	11.5	20.0	66.7
Income Inequality (lower=more equal)	43.0	28.9	−15.6	101.5
Mean Political Rights (lower=more rights)	−8.5	1.7	−11.9	−5.2
	n: 41, resid.sd: 18, R^2: 0.41			

Table 1: People living in countries with unequal income distributions report less protest activity to World Values Survey interviewers than people living in countries with relatively more equal income distributions, adjusting for average political rights as measured by Freedom House 1980–2000. Data from [@norris2015data].

Literate data analysis is not the same as Sweave, even if Sweave is a nice implementation.ˆ[The R project has a task view devoted to reproducible research listing many of the different approaches to literate programming for R.} If your workflow does not involve R, you can still implement some of the principles here. Imagine creating a style in Microsoft Word called 'code 'which hides your code when you print your document, but which allows you to at least run each code chunk piece by piece (or perhaps there are ways to extract all text of style code from a Microsoft Word document). Or imagine just using some other kind of indication linking paragraphs to specific places in code files. There are many ways that creative people can program in a literate way.

Literate programming need not go against the principle of modular data analysis (Nagler 1995). Jake routinely uses several different Sweave files that fulfill different functions, some of them create LaTeXcode that he can `\input` into his `main.tex` file, others setup the data, run simulations, or allow him to record his journeys down blind alleys. Of course, when we have flying cars running on autopilot, perhaps something other than Sweave will make our lives even easier. Then we'll change.

Step 2 We analyze data in order to explain something about the world to other scholars and policy makers. If we focus on explaining how we got our computers to do data analysis, we will do a better job with the data analysis itself: we will learn as we focus on teaching, and we will avoid errors and save time as we ensure that others (including our future selves) can retrace our steps. A document than can be 'run' to reproduce all of the analyses also instills confidence in readers and can more effectively spur discussion and learning and cumulation of research.

3 Meaningful code requires data.

All files containing commands operating on data must refer to a data file. A reference to a data file is a line of code the analysis program will use to operate on ('load'/ 'open'/ 'get'/ 'use') the data file. One should not have to edit this line on different computers or platforms in order to execute this command. Using R, for example, all analysis files should have `load("thedata.rda")`or `read.csv("thedata.csv")`or some equivalent line in them, and `thedata.csv`should be stored in some place easy to find (like in the same directory as the file or perhaps in `"Data/thedata.rda"`). Of course, its even better to include a comment pointing to the data file.

[MAARTEN IDEAS HERE? PLUS ANOTHER SECTION ABOUT ORGANIZATION?]

Where should one store data files? An obvious solution is always to make sure that the data file used by a command file is in the same directory as the command file. More elegant solutions require all co-authors to have the same directory structure so that `load("Data/thedata.rda")` means the same thing on all computers used to work on the project. This kind of solution is one of the things that formal version control systems do well (as discussed a bit more in sec. 4).

The principle of modularity suggests that you separate data cleaning, processing, recoding, and merging from analysis in different files (Nagler 1995). So, perhaps your analysis oriented files will `load("cleandata.rda")` and a comment in the code will alert the future you (among others) that `cleandata.rda` was created from `create-cleandata.R` which in turn begins with `read.csv(\url("http://www.greatfi`
Such a data processing file will typically end with something like `save("cleandata.rda")` so that we are doubly certain about the provenance of the data.[10]

Now, if in the future we wonder where `cleandata.rda` came from, we might search for occurrences of `cleandata` in the files on our system. However, if such searching among files is a burden, an even nicer solution is to maintain a file for each project called `MANIFEST.txt` or INDEX.txt or README.txt which lists the data and command files with brief descriptions of their functions and relations.

Step 3 We should know where the data came from and what operations were performed on which set of data.

In the good old days, when we executed our LISREL code in batch mode, we had no choice but to tell the machine clearly, in a few easy to understand and informative lines, what files (with filenames no longer than 8 characters) to use. See for example, this extremely clear code snippet:

```
DA NI=19 NO=199 MA=CM
LA=basic.lab
CM FI=basic.cov
```

The fact that we need to articulate this idea at all arises because of graphical user interfaces: it is all too easy to use the mouse to load a data file into memory and then to write a script to analyze this file without ever noting the actual name or location of the data file.

4 Version control prevents clobbering, reconciles history, and helps organize work.

Group work requires version control. Many people are familiar with the track changes feature in modern WYSIWYG (what you see is what you get) word processors or the fact that Dropbox allows one to recover previous versions of files. These are both kinds of version control. More generally, when we collaborate, we'd like to do a variety of actions with our shared files. Collaboration on data analytic projects is more productive and better when (1) it is easy to see what has changed between versions of files; (2) members of the team feel free to experiment and then to dump parts of the experimentation in favor of previous work while merging the successful parts into the main body of the paper; (3) the team can produce have 'releases' of the same document (one to MPSA, one to APSR, one to your parents) without spawning many possibly conflicting copies of the same document; (4) people can work on the same files at the same time without conflicting with one another, and can reconcile their changes without too much confusion and clobbering. Clobbering is what happens when your future self or your current collaborator saves an old version of a file over a new version, erasing good work by accident.

Of course if you rely on Dropbox or 'track changes' in Word for version control, you must communicate with other folks in your group before you edit existing files. Only one of you can edit and save a given file at a time. This prevents your work (or your colleagues work) from getting lost when you both try to save the same file on top of each other. If you find that you need to work on the same files at the same time, then you should work on establishing your own shared version control system. Free options include launchpad, github,

[10] Of course, if you need math or paragraphs to explain what is happening in these files, you might prefer to make them into R markdown or Sweave files, for which the conventional extension is .Rmd or .Rnw respectively. So you'd have 'create-cleandata.Rnw' written as a mixture of laTeX and R which might explain and explore the different coding decisions you made, perhaps involving a factor analysis and diagnostic plots.

Open Science Framework, sourceforge for open source projects (i.e. papers you are writing which you are happy to share with others as you write, see more below). Each has a slightly difference language (some say folders other repositories, some say download othere clone, etc) and it takes a bit of time adjusting. But as we'll argue: it is well worth the time investment as it will save (lots!) of time later on. Currently Google Docs does an excellent job with shared editing of files. The downside is: you have to be online. For people with irregular or slow internet connections, this may not be an option. MORE ON GOOGLE? One may also use Dropbox as a kind of server for version control (though the downside is if your collaborations or project grows large (in terms of MBs) you need to switch to a paid service). Maarten has been using this for most of his collaborations. You can copy files from the Dropbox directory into a local working directory so as to avoid clobbering and then work on merging changes by hand before copying back to the Dropbox directory and replacing existing files. You will however need to agree on an iron file and directory naming rule with your collaborators.

Jake uses github within his research group, and used it for all of his own projects. MORE ON GITHUB You may even obtain a two year unlimited repositories with an academic affiliation.

An excellent, simple, and robust version control system is to rename your files with the date and time of saving them: yyyymmdd_project.docx (for changes within the same day, just before a deadline for example, you may even add a time, so it becomes 20160623_4PM_inequalitypaper.docx). This communicates very clear what version you are working. Remember the days when you received a file called inequalityMV4_APSRsubmit_reallyfinal2.tex? That should quickly become something of the past! We find ourselves preaching the gospel to our collaborators frequently and will keep renaming files until a collaborator has been successfully pacified. Be sure to include year in the file names — remember, the life of an idea is measured in years. If you are wise enough to have saved your documents as plain text then you can easily compare documents using the many utilities available for comparing text files.[11] When you reach certain milestones you can rename the file accordingly: thedocAPSA2009.tex — for the one sent to discussants at APSA — or thedocAPSR2015.tex — for the version eventually sent to the APSR six years after you presented it at APSA. The formal version control systems mentioned above all allow this kind of thing and are much more elegant and capable, but you can do it by hand too as long as you don't mind taking up a lot of disk space and having many 'thedoc…' files around. Indeed, the number of files can add up quickly (especially around submission time!) and you may want to create an "0_Archive" folder to store older versions. If you do version control by hand, spend a little extra time to ensure that you do not clobber files when you make mistakes typing in the file-names. And, if you find yourself spending extra time reconciling changes made by different collaborators by hand, remember this is a task that modern version control systems take care of quickly and easily.

Step 5 Writing is rewriting. Thus, all writing involves versions. When we collaborate with ourselves and others we want to avoid clobbering and we want to enable graceful reconciliation of rewriting. One can do these things with formal systems of software (like subversion or git or bazaar) or with formal systems of file naming, file comparing, and communication or, even better, with both. In either case, plain text files will make such tasks easier, will take up less disk space, and be easier to read for the future you.

5 Minimize error by testing.

Anyone writing custom code should worry about getting it right. The more code one writes, the more time one has to appreciate problems arising from bugs, errors, and typos in data analysis and code. Mistakes will always be made we should just recognize we need more standardization of practice to minimize the mistakes and help others find them. One thing we can do is to include testing in our coding.

[11] Adobe Acrobat allows one to compare differences in pdf files. OpenOffice supports a 'Compare Documents 'option. Word now does the same. And Google Docs will report on the version history of a document.

If one has a moment to think in between teaching that new class, reading books for an awards committee, evaluating application files for the admissions committee, feeding popsicles to a sick child, and undertaking the odd bit of research, one might say to oneself, "Before I write new code, I should write a test of the code. I should write a little bit of code that lets me know that my double-bootstrap procedure actually does what it is supposed to do."

Of course, this idea, like most others, is not new. The desire to avoid error looms large when large groups of programmers write code for multi-million dollar programs. The idea of test driven development and the idea that one ought to create tests of small parts of one's code arose to address such concerns.[12] For the social scientist collaborating with her future self and/or a small group of collaborators, here is an example of this idea in a very simple form: Say you want to write a function to multiply a number by 2. If the function works, when you give it the number 4, you should see it return the number 8 and when you give it -4, you should get -8.

```
##  The test function:
test.times.2.fn <- function(){
  ##  This function tests times.2.fn
  if (times.2.fn(thenumber=4) ==  8 &
      times.2.fn(thenumber=-4) == -8) {
    print("It works!")
  } else { print("It does not work!")
         }
}

##  The function:
times.2.fn <- function(thenumber){
  ##  This function multiplies a scalar number by 2
  ##  thenumber is a scalar number
  thenumber+2
}

##  Use the test function
test.times.2.fn()
```

```
[1] "It does not work!"
```

Ack! we mistyped '+' for '*'. Good thing we wrote the test![13]

MAYBE SOMETHING HERE ON `stopifnot` and recoding data.

Step 6 No one can forsee all of the ways that a computer program can fail. One can, however, at least make sure that it succeeds in doing the task motivating the writing of the code in the first place.

6 Creating a reproducible workflow

Lots of people are thinking about 'reproducible research', creating a 'reproducible workflow' and 'literate programming' these days. Google those terms. Of course, the devil is in the details: Here I list a few of

[12] There are now R packages to help R package writers do this, see for example https://github.com/hadley/testthat and the article on it https://journal.r-project.org/archive/2011-1/RJournal_2011-1_Wickham.pdf.

[13] A more common example of this kind of testing occur everyday when we recode variables into new forms but look at a crosstab of the old vs. new variable before proceeding.

my own attempts at enabling reproducible research. You'll find many other inspiring examples on the web. Luckily, the open source ethos aligns nicely with academic incentives, so we are beginning to find more and more people offering their files online for copying and improvement. By the way, if you do copy and improve, it is polite to alert the person from whom you made the copy about your work.

We have experimented with several systems so far: (1) we wrote this paper in R markdown (Jake) and Atom (Maarten) and organized our files and collaboration on Github. The source code can be accessed, downloaded and modified from https://github.com/jwbowers/workflow. The program also allowed us to send notes to each other. The nice thing about github is that is enables you to make "releases" of a project which enable you to smooth the reproduction of all of the products of your research (simulations, data analyses, tables, figures, etc..). Similar features are offered by the Open Science Framework (https://osf.io/), Maarten recently participated in a workshop on research transparency organized by the Berkeley Institute for Transparency in the Social Sciences (www.bitss.org), all course material was accessible through OSF (https://osf.io/qh2nr/). Both platforms, OSF and Github nicely integrate and a key benefit is that they have been created to remain for a long time (unlike perhaps your university drive) ; (2) for one paper Jake simply included a Sweave document and data files into a compressed archive (Bowers and Drake. 2005) ; (3) for another more computing intensive paper Jake assembled a set of files that enabled reproduction of results using the `make` system (Bowers, Hansen, and Fredrickson 2008) ; (4) Jake also tried the "compendium" approach (Gentleman 2005,Gentleman and Temple Lang (2007)) which embeds an academic paper within the R package system (Bowers 2011a); The benefit of the compendium approach is that one is not required to have access to a command line for `make`: the compendium is downloadable from within R using `install.packages()` and is viewable using the `vignette()` function in any operating system than runs R.[14] The idea that one ought to be able to install and run and use an academic paper just as one installs and uses statistical software packages is very attractive, and we anticipate that it will become ever easier to turn papers into R packages as creative and energetic folks turn their attention to the question of reproducible research.[15] Finally, (5) Maarten has been using Dropbox for most of his collaborative projects (https://www.dropbox.com/). See comments above. As a tool for making files publicly available Dropbox is less useful. You can of course make replication files available through creating an public folder, but you will still need to refer to this folder on a particular project website (an example is Maarten's project on conflict and football https://www.dropbox.com/sh/wqosq99019mf04d/AAASTgJLkn6lmZJI_1veUK7ca?dl=0)

There are some noteworthy new developments also. For example, and similar to R Markdown/KnitR in R Studio, there is Markdoc for STATA users (https://github.com/haghish/MarkDoc). People at UC Berkeley have been developing a is a web app, Jupyter Notebook (http://jupyter.org/), that enables you to make and share documents that include text, code, equations and visualizations in one INCLUDE REF TO GITHUB - USING PYTHON. For researchers working across different platforms, there is Docker (https://www.docker.com/). There are also are great notes programs that help you communicate with collaborators. For example, Simplenote is very basic but allows you to share and change notes.

Step 6 We all learn by doing. When we create a reproducible workflow and share reproduction materials we improve both cumulation of knowledge and our methods for doing social science [Freese (2007),king1995replication}.

7 Remember that research ought to be credible communication.

> [I]f the empirical basis for an article or book cannot be reproduced, of what use to the discipline

[14] Notice that Jake's reproduction archives and/or instructions for using them are hosted on the Dataverse, which is another system designed to enhance academic collaboration across time and space.

[15] See for example http://r-pkgs.had.co.nz/

are its conclusions? What purpose does an article like this serve? (King 1995, 445)

We all always collaborate. Many of us collaborate with groups of people at one moment in time as we race against a deadline. All of us collaborate with ourselves over time.[16] The time-frames over which collaboration are required — whether among a group of people working together or within a single scholar's productive life or probably both — are much longer than any given version of any given software will easily exist. Plain text is the exception. Thus, even as we extol version control systems, one must have a way to ensure future access to them in a form that will still be around when sentient cockroaches finally join political science departments (by then dominated by cetaceans after humans are mostly uploads).[17]

But what if no one ever hears of your work, or, by some cruel fate, your article does not spawn debate? Why then would you spend time to communicate with your future self and others? Our own answer to this question is that we want our work to be credible and useful to ourself and other scholars regardless. What we report in our data analyses should have two main characteristics: (1) the findings of the work should not be a matter of opinion; and (2) other people should be able to reproduce the findings. That is, the work represents a shared experience — and an experience shared without respect to the identities of others (although requiring some common technical training and research resources).

Assume we want others to believe us when we say something. More narrowly, assume we want other people to believe us when we say something about data: 'data' here can be words, numbers, musical notes, images, ideas, etc ... The point is that we are making some claims about patterns in some collection of stuff. For example, when Jake was invited into the homes and offices of ordinary people in Chile in 1991, the "stuff" was recordings of conversations that we had about life during the first year of democracy and Jake was comparing the responses of people who had experienced the Pinochet dictatorship differently. Now, it might be easy to convince others that 'this collection of stuff' is different from 'that collection of stuff' if those people were looking over our shoulders the whole time that we made decisions about collecting the stuff and broke it up into understandable parts and reorganized and summarized it. Unfortunately, we can't assume that people are willing to shadow a researcher throughout her career. Rather, we do our work alone or in small groups and want to convince other distant and future people about our analyses and findings.

Now, say your collections of stuff are large or complex and your chosen tools of analyses are computer programs. How can we convince people that what we did with some data with some program is credible, not a matter of whim or opinion, and reproducible by others who didn't shadow us as we wrote our papers? This essay has suggested a few concrete ways to enhance the believability of such scholarly work. In addition, these actions (as summarized in the section headings of this essay) make collaboration within research groups more effective. Believability comes in part from reproducibility and researchers often need to be able to reproduce in part or in whole what different people in the group have done or what they, themselves, did in the past.

In the end, following these practices and those recommended by Fredrickson, Testa, and Weidmann (2011) and Healy (2011) among others working on these topics allows your computerized analyses of your collections of stuff to be credible. Finally, if the someone quibbles with your analyses, your future self can shoot them the archive required to reproduce your work.[18] You can say, 'Here is everything you need to reproduce my work.' To be extra helpful you can add 'Read the README file for further instructions.' And then you can

[16] What is a reasonable time-span for which to plan for self-collaboration on a single idea? Ask your advisers how long it took them to move from idea to dissertation to publication.

[17] The arrival of the six-legged social scientists revives Emacs and finally makes Ctrl-a Ctrl-x Esc-x Ctrl-c a reasonable key combination.

[18] Since you used plain text, the files will still be intelligible, analyzed using commented code so that folks can translate to whatever system succeeds R, or since you used R, you can include a copy of R and all of the R packages you used in your final analyses in the archive itself. You can even throw in a copy of whatever version of linux you used and an open source virtual machine running the whole environment.

get on with your life: maybe the next great idea will occur when your 4-year-old asks a wacky question after stripping and painting her overly cooperative 1-year-old brother purple, or teaching a class, or in a coffee shop, or on a quiet walk.

References

Asendorpf, Jens B et al. 2013. "Recommendations for Increasing Replicability in Psychology." *European Journal of Personality* 27(2): 108–19.

Bowers, Jake. 2011a. "Reproduction Compendium for: 'Making Effects Manifest in Randomized Experiments'." http://hdl.handle.net/1902.1/15499 (September 26, 2008).

———. 2011b. "Six Steps to a Better Relationship with Your Future Self." *The Political Methodologist* 18(2): 2–8.

Bowers, Jake, and Katherine W. Drake. 2005. "Reproduction Archive for: 'EDA for HLM:Visualization when Probabilistic Inference Fails'." http://hdl.handle.net/1902.1/13376 (2005).

Bowers, Jake, Ben B. Hansen, and Mark M. Fredrickson. 2008. "Reproduction Archive for: 'Attributing Effects to A Cluster Randomized Get-Out-The-Vote Campaign'." http://hdl.handle.net/1902.1/12174 (September 26, 2008).

Fredrickson, Mark M., Paul F. Testa, and Nils B. Weidmann. 2011. "Collaboration for Social Scientists, or Software Is the Easy Part." *The Political Methodologist* 18(2).

Freese, Jeremy. 2007. "Replication Standards for Quantitative Social Science Why Not Sociology?" *Sociological Methods & Research* 36(2): 153–72.

Gentleman, Robert. 2005. "Reproducible Research: A Bioinformatics Case Study." *Statistical Applications in Genetics and Molecular Biology* 4(1): 1034.

Gentleman, Robert, and Duncan Temple Lang. 2007. "Statistical Analyses and Reproducible Research." *Journal of Computational and Graphical Statistics* 16(1): 1–23.

Healy, Kieran. 2011. "Choosing Your Workflow Applications." *The Political Methodologist* 18(2).

King, Gary. 1995. "Replication, Replication." *PS: Political Science and Politics* 28(3): 444–52.

Knuth, Donald E. 1984. "Literate Programming." *The Computer Journal* 27(2): 97–111.

Koriat, Asher, and Robert A Bjork. 2005. "Illusions of Competence in Monitoring One's Knowledge During Study." *Journal of Experimental Psychology: Learning, Memory, and Cognition* 31(2): 187.

Lupia, Arthur, and Colin Elman. 2014. "Openness in Political Science: Data Access and Research Transparency." *PS: Political Science & Politics* 47(01): 19–42.

Nagler, Jonathan. 1995. "Coding Style and Good Computing Practices." *PS: Political Science and Politics* 28(3): 488–92.

R Development Core Team. 2011. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. http://www.R-project.org.

Twain, Mark. 1975. 8 *Mark Twain's Notebooks & Journals, Volume I:(1855-1873)*. Univ of California Press.

Whitman, Walt. 1855. "Leaves of Grass." In Project Gutenberg [2008], 51. http://www.gutenberg.org/files/1322/1322-h/1322-h.htm.