PRESENTER NOTES—How to improve the credibility of (your) social science: A practical guide for researchers

These notes accompany the `pdel_credibility_solutions.tex` *slides, a teaching module for graduate-level social sciences methods courses (available online at PDEL's GitHub page).*

- *The structure of the notes parallels the slides; each second-level header is a beamer section, each third-level header is a subsection, and each fourth-level header is a frame.*
- *Frame titles begin with a* `">"` *symbol.*
- *Some frames are self-evident and do not have corresponding notes.*
- *For the presentation itself, you can remove the word "handout" from line 8 of the beamer file ("* `\documentclass[12pt, compress, handout]{beamer}` *") in order to have everything appear in a stepwise fashion.*

These notes were prepared using the Atom text-editor, and converted from markdown (`.md` *) format into PDF using the* `markdown-themable-pdf` *package. They are licensed under Creative Commons BY-NC 4.0. You are free to share and adapt them for any non-commercial purpose with proper attribution.*

Please cite as: Clark, J., Desposato, S., and McIntosh, C. 2017. "How to improve the credibility of (your) social science: A practical guide for researchers." Policy Design and Evaluation Lab (PDEL). University of California, San Diego.

The Problem

> We often hear about ...

There appears to be a lack of credibility in much of the research that we see published (and publicized, e.g., as illustrated in the above XKCD cartoon). We often hear about this in terms of the **replication crisis** (it's unlikely that the jellybeans result would replicate!), **publication bias**, choices made by researchers (**p-hacking or researcher degrees of freedom**) as well as outright **fraud**.

These problems are related and persistent across the social sciences, including political science, economics, psychology, sociology, and also medical research. The result is a body of scientific knowledge that we don't have much confidence in.

> Why do we have a credibility crisis?

The credibility crisis has origins at both the disciplinary and researcher levels:

At the **discipline-level**, this includes:

1. **Lack of transparency**—In order to be able to replicate and extend work that came before us, we need to know: (a) the universe of previous studies, (b) Their methods and specifications, and (c) their results (from ALL studies, not just the ones that got published!).
2. **Tenure and journal standards that promote a "publish or perish" mentality**—This generates incentives to publish only "high impact" studies (i.e., those with significant and often positive results).
3. **The flexibility of many social science questions and methods**—This allows for significant researcher degrees of freedom, making unintentional or intentional specification searching, p-hacking, etc. easy.

These norms and incentives shape **researcher** behavior:

1. Researchers often have **little incentive** to share data, and/or *low capacity* in terms of tools and training to make their code and data useful to others.
2. Given publishing expectations, many researchers **never submit studies with negative or null results**, i.e., the "file drawer problem".
3. Researchers may make a number of choices that manipulate their results, either unintentionally (e.g., via **researcher degrees of freedom**) or intentionally (**fraud**)

Let's walk through the four main symptoms of these discipline- and researcher-level problems, including the **replication crisis**, **publication bias**, **researcher degrees of freedom**, and **fraud/misconduct**.

1. "Replication Crisis"

> Social, behavioral, and medical studies often don't replicate

Failure to replicate should mean that the original results were due to random chance. Unfortunately, however, they are often due to

- **A lack of transparency**—results can't replicate when complete code and data are not shared
- **Errors in data and code**—often times a 'failure to replicate' is caused by silly errors, like a missing package, or failure to copy and pasted updated regression outputs into a paper draft
- In some cases, studies don't replicate because the **results were fabricated or manipulated**

> Dewald et al. (1986)

Dewald et al. attempted one of the first mass replications in economics and encountered a number of problems that illustrate the above issues:

- To begin, only a portion of the authors actually provided their data. The response rate was the worst for papers that had already been published (only 22 out of 62), and somewhat better for those articles under review (47 out of 65).
- In addition, all but 8 of the papers with submitted data had significant problems, including failure to disclose the source of the data, incomplete datasets, datasets without variable names, etc.
- For those they were able to analyze, they found numerous transcription and coding errors, leading to a high failure to replicate.

> Fang et al. (2012)

Fortunately, data sharing and replication have become much more normalized over the past few decades, making it easier to identify and retract bad research. Looking at *why* articles were retracted also help identify some of the root causes of failure to replicate.

As we can see in this graph from Fang et al., **retraction of such articles has been on the rise in medical research**; and while a majority of recent retractions have been the result of fraud and plagiarism, a significant number are also due to errors in data analysis.

2. Publication Bias

> AKA the "file drawer problem"

Another (and the ideal) reason our studies fail to replicate is that published results may be anomalies. However, not all studies have an equal probability of being published, and those that are may be *more likely* to be anomalous than those

that remain in the file drawer.

- This is the result of "publication bias", or the fact that articles with surprising findings and large and positive results are more likely to be published (e.g., there are no headlines proclaiming "blue jellybeans NOT linked to acne!").
- This means that we are missing the full universe of studies, and the sum total of the results we do see are likely to be biased.

> Fanelli (2010 & 2011)

Publication bias appears to have increated over time across scientific disciplines; but as Fanelli shows, this trend problem also appears to be larger in the social sciences. In 2007, **approximately 90% of the 810 published papers he surveys have positive results.**

> Franco, Malhotra, Simonovits (2014)

Franco et al. go deeper into social science publication bias, and find that **strong results are 60 percentage points more likely to be written up by authors than null results**, and **40 percentage points more likely to be published.**

Simply put; authors tend not to write up null results, as null results tend not to be published.

> This has consequences!

Publication bias can be even worse when studies are linked to vested policy interests (e.g., who may be funding studies!). For example, a study by Turner et al. looked at FDA-registered studies on anti-depressants (which may or may not have been published) and compared these with the corresponding published articles. They found that

- **All but one** study with positive results were published and agreed with the FDA regulations on the anti-depressants
- Of the studies that had negative results for anti-depressant trials, **67% were not published**

3. P-Hacking

> "Torture the data until it tells you what you want to hear

Before deciding which results to write up (and before journals decide which articles are accepted), **researchers make *tens or hundreds* of decisions** that affect the results they get.

- We can think of these as **"researcher degrees of freedom (RDF)"**, including which hypotheses to test, what data to collect, how to operationalize a variable, what models to run, etc.
- These decisions may be conscious (e.g., tweaking the model to get p <0.5, or p-hacking), or may not be
- Either way, however, the result is a biased evidence base

[**OPTIONAL ACTIVITY**: take students to fivethirtyeight, p-hacker app, or [fishing app](https://macartan.shinyapps.io/fish/ to try their hand at p-hacking)]

> Brodeur et al. (2016):

Some clear evidence of the existence of p-hacking comes from Brodeur et al.'s awesomely titled paper ("Star Wars: The Empirics Strike Back"). They survey 50,000 hypothesis tests published in flagship economics journals (AER, JPE, and QJE) and find a clear camel-humped distribution for test-statistics around the 0.5 confidence-level cutoff.

> Wicherts et al. (2016)

But again, this goes beyond simple p-hacking. Wicherts et al. (in a cool project available on OSF) identify 34 key researcher degrees of freedom that can, consciously or not, bias our results. This table shows a few.

4. Misconduct and Fraud

> Rare(?) but serious

In the worst-case scenario of RDF, researchers can engage in fraud, including falsification of results, plagiarism, and other types of misconduct.

Brazen falsification of data can create high profile scandals, such as

- Mike LaCour, a polisci PhD candidate who fabricated data for a well-publicized experiment on the effect of LGBT canvassers' personal conversations on people's likelihood to support marriage equality
- Yoshitaka Fujii, an anesthesiologist who fabricated data for some 183 papers, and now holds the record for most-retracted papers
- Diederik Stapel, a social psychologist who was suspended from his university for for fabricating and manipulating data for at least 55 publications.

However, while these extreme cases might be relatively rare, lower levels of manipulation are more common.

> John et al. (2012)

In a 2012 survey of over 2000 psychologists, for example, John et al. found that a variety of bad practices were relatively common, both in self reports and in estimates by researchers of their prevalence among colleagues.

- For example, **around 40% of researchers admit to excluding data after looking at the impact of doing so** (which the author's extrapolate to mean that around 100% of researchers actually do this)
- And while only a small percent admitted to falsifying data, 9% claim that others do so, and the **authors estimate that closer to 40% of researchers have engaged in this behavior**.

> Norms are changing

This is depressing. But things appear to be getting better!

> Research lifecycle: Individual-level solutions

There are clear steps YOU can—and are increasingly expected—to take at every stage of the research process to improve the credibility of your work:

- **Design**—BEFORE data collection, register your study (addresses publication bias) and submit a pre-analysis plan (addresses researcher degrees of freedom). Complying with IRB requirements is also a key element of ethical research that meets disciplinary standards, but we won't go into that here.
- **Analysis**—During data analysis and write-up, use good data and file management practices (helps with transparency and reproducibility), and also make sure you're following your pre-analysis plan!
- **Dissemination**—Once the analysis is done, you should prepare files for replication and upload data and code in online repositories. This is increasingly required by journals, and should be done in a way to protect PII.

[For more disciplinary solutions, see additional notes at the end]

Once your results enter the universe, they can be used by others for **replication** and **meta analysis**. Your study is one datapoint among many, and replication and meta-analysis can help separate out weak from strong results across the universe of studies.

The remainder of this presentation will talk about how to implement these solutions practically, throughout each stage of the lifecycle.

Solutions I: Design

> Steps

There are two concrete steps toward credibility during the design phase: (1) **registering studies** and (2) **submitting pre-analysis plans**. Let's walk through both of them.

1. Registration

> About Registration

Registration is a partial, though not complete, solution to the file-drawer problem that results in publication bias. By registering your study in the **appropriate repository**, it is recorded as a datapoint in the universe of knowledge so that—even if not published in a journal—others can find and build off of it.

Registration is now the **norm in experimental work**, particularly for medical trials where top journals (e.g., ICMJE) won't publish RCTs that haven't been registered. It is **less but increasingly common for observational work**.

Besides helping with publication bias, registration can **benefit you personally** by allowing you to stake out intellectual claim for a project that may take a long time to complete.

> Where to Register

The ideal place to register depends on your discipline. We're going to focus on AEA and EGAP, which are most common for economics, and political science, respectively.

> AEA

The AEA is for RCTs only, and makes it fairly easy to register a study. Just **make an account** and click on **register a trial**. (But note that once you submit, you can never take it back!)

[**OPTIONAL ACTIVITY**: take students to AEA and walk through this process]

> EGAP

EGAP focuses on RCTs (in governance and politics), but you can also register non-experimental work. Registration is slightly more involved, as you first have to **register yourself in the author database**, before completing the registration form.

[**OPTIONAL ACTIVITY**: take students to EGAP and walk through this process]

2. Pre-Analysis Plan

> About Pre-Analysis Plans (PAPs)

A **pre-analysis** plan is a detailed description of your research design and data analysis plans that you submit to an official

repository BEFORE collecting or analyzing any data.

The goal here is to help **reduce RDF** by tying your hands (somewhat) and stating up front whether you will be doing an exploratory analysis or testing specific hypotheses.

PAPs can **benefit you** by making your work more credible (OSF even gives you a badge for submitting a PAP), and helping you clearly think through your research design. It also benefits others who will be able to more easily identity your methods and build off of your work.

> PAP vs. Registration

PAPs and registration are often confused, which makes sense because they often go together. But the function is different.

- **Registration** is about reducing publication bias by saying "hey, I'm doing this study!"
- **PAPs** are about reducing RDF/p-hacking by saying "hey, I promise to do my study in this way, regardless of the results get!"

> Where to Submit a PAP

Conveniently, you can generally submit PAPs to the same registries you used to register your study, though they usually do not need to be submitted at the same time (you can register before you PAP).

> OSF

- OSF calls itself a "A scholarly commons to connect the entire research cycle". The goal of OSF is to be a **one-stop-shop** for managing projects, collaboration, and materials.
- It integrates registration (e.g., soon with AEA), pre-analysis plans, file management (e.g., with Dropbox), version control (e.g., with GitHub), etc., across the social science disciplines.
- OSF even offers a **"pre-registration challenge"** (another term for PAP), where you can win $1000 if an article for which you submitted a PAP is published.

[**OPTIONAL ACTIVITY**: take students to OSF, set up an account, and explore]

> No universal standard, can include ...

So what do you include in your PAP? Unfortunately, there are **no universal standards**. A cobbling together from various sources gives us a laundry list of requirements covering the basic background of your research, design, analysis, the research team, and logistics.

> Olken's PAP Checklist (2013)

Olken distills some **minimal elements** of a PAP, including

- variable definitions
- strategies for multiple hypothesis testing
- rules for how observations will be included or excluded (e.g., missing data)
- basic model specifications
- covariates to be used
- plans for sub-group analyses
- etc.

> Tie your hands in the right places

PAPs are great, but they can go too far. At the extreme, you could **write your code or even your entire paper ahead of time**, leaving spaces only for results. This has the benefit of front-loading the work and potentially minimizing RDF, but is generally impractical. If circumstances change, it leaves little room for adaptation.

In general, finding the right balance requires thought (and experience).

> Ongoing Debate

There's also a continuing debate over the merits of PAPs, with both champions and detractors.

- As **Olken** notes, pre-specification can take a lot of time (resources), and make for a frustrating research experience if too detailed or constraining. They may be less useful (or not needed at all) when little fraud or manipulation is anticipated, or when peer review, open data, and replication standards are high and able to catch shoddy work or anomalous results.
- **Coffman & Niederle** make similar points, suggesting that replication can solve most of the RDF issues with far less headache.
- However, although many argue that PAPs are too time-consuming and not needed for **observational work**, **Neumark** has a great paper that shows how it can work well when theories are well defined and developed. By pre-specifying his hypotheses and models *before* a scheduled release of new government data, he adds significant credibility to estimates of the effect of minimum wage increases on employment, which in the 1990s had ranged from positive to negative to null, potentially influenced by subjective research choices (or ideological positions)

Again, the best way to figure out when and what type of PAP works for you is to experiment!

[IRB]

> Not covered here, but ...

Finally, a note on IRB. Dealing with this topic thoroughly would require a separate session, but it is important to mention here as a key piece of the research design process.

Universities and journals **require IRB approval** for much social science, and so failing to complete these requirements will put your research in jeopardy. As annoying as they are, protecting individual subjects is a minimum requirement for **credibility and ethical standards**.

But, just because you've fulfilled IRB requirements doesn't mean that your research is now "ethical". See **Desposato** for more.

Solutions II. Analysis

> Steps

After getting your study registered and your PAP submitted, you actually have some data. This section walks through key ways in which you can protect the integrity of you data and code and ensure that your workflow is well-documented and reproducible, both for yourself and others. If your files are a mess now, dissemination and replication will be a challenge later.

These three solutions (**file management**, **literate programming**, and **version control**) are not sequential; you should

build each of them in from the beginning!.

Yes, they are boring, but you will thank yourself later (or the next time you take a break from your project for a few weeks and have to pick up where you left off).

3. File Management

> About File Management

Manage your files in a logical way. This seems obvious, but is not as common as you would think.

Good file management helps **protect original datasets**, **increase the efficiency** of your workflow, and **makes it much easier** to share your data and code in a way that is actually useful to other people.

> Don't let your files look like this ...

But, this is easier said than done. Oftentimes, file structures spring up organically, and we end up with folders looking like this hot mess of code and data and PDFs and who-knows-what.

> Instead, use PDEL template (or similar)

If you don't feel like coming up with a system yourself, PDEL has made a generic folder template that you can download from Github and simply duplicate whenever you're starting a new project. The basic structure includes separate folders for:

- code files
- raw data—never to be modified or saved over!!!
- clean data that you have manipulated with code
- outputs (e.g., tables, figures, etc. that results from your code)
- extras (e.g., for codebooks, etc.)
- It also includes a README file template.
- You might also want to add a "paper" folder to this, or modify however you like.

Once we get to the "Dissemination" solutions later on, you'll see how spending a marginal amount of time to create a logical file structure up-front can save a lot of time and headaches when you have to prepare your files for replication.

4. Literate Programming

> About Literate Programming

Obviously your code must be machine-readable or it won't run. The goal here is to also **make your code decipherable to humans**, including collaborators, replicators, and your future,very-annoyed self who can't remember why you coded your model that way.

> (The Most) Basic Principles

There are lots of philosophies, guides, and different opinions on the best way to write code. Here are a few basics that should be relatively uncontroversial, although rules can always be broken, YMMV, etc.

Help yourself and others by:

- Structuring and name your files intuitively
- Making text and commands easy to navigate by annotating files more than you think you need to, and decluttering

your text (whitespace is your friend!)
- Streamlining commands to avoid repetition and cumbersome changes

> Structure and Name Files

- Create **separate scripts** for merging/cleaning and data analysis, or weird particular tasks that take up tons of lines of code (e.g., correcting the names of 100 districts so that two datasets will merge properly)—don't only have one file that is 5000 lines long, it will be easy to get lost and make mistakes!
- But if you have multiple files, also keep a **master-script** for running them all at once, to save yourself the hassle of opening and running multiple files
- Give code, data files, and output **logical names** where possible
  - Number folders/files sequentially in the order they should be run (e.g., `1_main_analysis.R`, `2_robust_checks.R`)
  - Label output figures with descriptive names, but ones that aren't likely to change (e.g., `figure_hte.png` is better than `figure_1.png`)

> Improve Navigation

- Add headers to your code that describe its main purpose, as well as the author and associated project for attribution (see PDEL template)
- Format scripts so they're easily readable—e.g., indent code, use ample line breaks and spaces, standardize comment syntax
- Add comments to improve reader understanding (but before you publish, make sure to go back and remove unhelpful/embarrassing comments like "#there seems to be a problem with this model, double-check")
- Clearly label code sections, main analyses, outputs
- Give variables intuitive names like `edu_percent` rather than `v76`
- Label variables and values in Stata

> Streamline Code—e.g., working directories

Your code will be cleaner—and less prone to breaking—if you avoid repetition throughout (e.g., using functions/macros rather than hardcoding things, etc.)

Covering coding best-practices is another lecture. However, one basic thing you can do to (1) decrease the length of file paths, (2) avoid having to make lots of changes if your file structure changes, and (3) make sharing with coauthors easier is to **set the working directories** of a project at the beginning of the code.

Here are some examples in **R and Stata**. Note that if you use `cd` in Stata, you can have one line for each collaborator's working directory, and Stata will scan over them until it finds one that works (i.e., the one corresponding to you!). This saves having to comment/uncomment/change the directory every time the file changes hands.

5. Version Control

> "Final.doc" [comic]

All too often, iterated manuscripts and code scripts take the form of a jumble of files with different extensions. This can quickly become a nightmare and we can do better with a good version control system!

> About Version Control

A **version control** system is one that manages iterative versions of files (code, data, manuscripts) over time and, ideally,

across collaborators. The goal of such a system is to preserve original files, protect your work, collaborate efficiently, etc., etc.

This can be done **manually** or with **version control software**. You're probably already familiar with version control software like Microsoft Word's "track changes", or Google Doc's "suggestion" feature. But they are not very powerful. In Word, comments and track changes become unwieldy very quickly. Once you accept a change, there is no way to roll-back to a previous version of the document. Also, they **don't work with code**! Today, we'll learn about some other options.

> Principles of Version Control

Whether you're doing it on your own or with software, version control systems should ensure that:

- Original, raw data files are vaulted or segregated, so that they cannot be accidentally changed or written over
- Changes to files should be documented and reversible
- "Master" versions of code are always in working order; create copies before experimenting
- Independent changes by different users can easily be reconciled

> Manual Solutions (not ideal, but better than nothing)

For code, a minimum-level of version control might include:

- Create dated versions of files (save-as) for each substantive change
- With each modification, re-run ALL code to make sure nothing is broken—this is where it helps if you have a master file to run all scripts!
- Check-in with coauthors to ensure multiple people aren't working on the same files at the same time (good luck!)
- Keep a simple log to remind yourself of the location/content of major changes (this takes energy and time)

> Or let version control software do this for you!

But there are better solutions! Today we'll learn about one: GitHub.

[To anyone already familiar with Git/Hub, the following section is likely to be boring (or potentially eye-roll-inducing because it glosses over many of the software's finer points). The goal here is to demystify Git and take some baby steps.]

> Version control software > Git > GitHub

There are a many options for version control software, but we will use **Git**. Git was developed by Linux creator Linus Torvalds, and is an *open-source*, *distributed model* of version control. This means that each user works with repositories (files) locally on their own computer, and then changes to these files can shared with other users.

With Git, sharing between users can work in a number of ways. The easiest is through **GitHub**, a free, web-based service that hosts Git repositories and offers a variety of features for collaboration.

> Common problems that GitHub helps solve

Why use Git/Hub? It's software specifically design for version-controlling that lets you *collaboratively* view, accept, and roll back changes at any stage in a file's development, and it works for text files as well as code.

Git/Hub solves a lot of common workflow issues, including:

- **Tracking Changes**. Including who did what, when, (and why), preserved forever. No more `code_file_date_initials.do` files to sort through.

- **Reverting changes.** Control+Z lets us undo the last change we made. But this isn't great if you want to keep the 3rd, change you made, but undo the 1st and 2nd. Instead of control+Zing three times and then re-doing (and possible forgetting) change number 3, *Git let's you keep the changes you want and undo the changes you don't, no matter the order* (assuming you've committed each change individually, more on that later).
- **Experimenting.** Commonly, we "save as" to preserve original code while experimenting in a new document, which can create problems if the two eventually need to be combined. For large projects or those with many versions, this system also tends to populate your project directory with a graveyard of obsolete documents. Chances are you will hesitate to delete old versions in case they contain something important, but in reality you will probably never look at them again. *Git solves this problem by letting you create a "branch" of code off of a "master" file, and then merge the two together when needed, flagging any conflicts for resolution.*
- **Collaborating.** Researchers who collaborate on code and manuscripts often use a software like Dropbox to share files and sync versions. Although this is better than email attachments, it doesn't solve the problem of simultaneous work, and the tedium of resolving conflicting changes. Multiple authors also multiply potential issues with experimentation and tracking and reverting changes. *GitHub helps facilitate collaboration by providing a platform for multiple users to propose/accept/discuss/reject each other's changes in realtime, ensuring that "master" documents are preserved.*

> How do I use GitHub?

While it's possible to use *only* the **GitHub website**, this requires the desire/ability to work in your files directly through the online interface. If you have a constant internet connection and work only in non-code text files, this may be do-able. But mostly it's a pain.

A much easier option is to download the free **GitHub Desktop** client for Windows and Mac (for Linux, you have to use the command line, which you can also use on Macs and PCs). This lets you work on files locally and then sync them with the remote repositories hosted in GitHub online.

Advanced users can also take full advantage of Git features by using the **command line**. There are limitations to what the Desktop client can do, and sometimes it's buggy, so the command line can be liberating.

> How to think about Git

Git was designed for developers to manage software development (initially Linux) projects. This means that unless you have a background in computer science, Git/Hub will probably seem unintuitive and irritating at the beginning. Sorry (but you are not alone!).

One way to think about Git is as a helpful (creepy?) **Big Brother:** You point it toward a set of files (a "repository") and then it never stops watching them, tracking every change*.

*Assuming you're working in a type of file that Git can read—e.g. text files and most code files—it shows you line by line what changes have been made were. However, it's not very helpful with PDFs, Word docs, Excel files, etc. Because it can't read inside these formats, it will only tell you if a document of this type has been added, saved, or deleted within the repository.

[Another way to think about Git: Harrison Massey of hastac, describes how core Git processes are analogous to the analog writing process.]

> GitHub is NOT ...

When you first login to GitHub online or open GitHub Desktop, the interface can seem somewhat counterintuitive, because it *looks* like software we are used to working with but functions very differently.

It will be easier to learn GitHub if you internalize the fact that it is NOT:

- **Dropbox/GoogleDrive.** Git tracks changes in files and let's you sync/track/view/edit those changes online and with collaborators. It's primary function is NOT to be a hard drive in the cloud (although repository files are stored and accessed via the GitHub server).
- **File Manager.** GitHub desktop may *look* sort of like Mac Finder or PC Explorer, but its function is not to let you find, click on, or open files. Git's unit of operation is *changes within those files*. Think of it more as Big Brother's dashboard for changes in your document, and the whole thing makes a lot more sense.

> (The most) basic vocabulary

Git has a very specific and very precise vocabulary for different features, processes, and functions. Before we start, we'll learn two of the most important terms, and then pick up a few more as we go along:

- **Repository.** A set of files (in a folder) that you have told Git to track, along with its associated `.git` files (these may be invisible on your computer).
  - A *local repository* is the copy on your computer that you see when using GitHub Desktop.
  - A *remote repository* is the copy online that you can view through GitHub.com, and with which you sync your local repository through GitHub Desktop.
- **Commit.** A labeled change or group of changes within a repository.
  - Git tracks every change you make within a repository that you're watching, silently, in the background (creepy, no?)
  - Then, when you open GitHub desktop (or go online, or use the command line), it shows you every single change that has been made, line by line
  - You then decide how you want to group and label these changes (e.g., "added README file"), and then "commit" these changes to the official record
  - If you don't like a particular commit later on, you can *"revert"* (i.e., undo) it

> 10 Baby Steps in Git—Prep

Before we get into Git, do the following:

- **Make sure you have a good text editor.** Notepad or TextEdit will work (if you set TextEdit to `.txt` and not `.rtf`). Or get a more powerful editor like Atom [which is the app used to write these notes in Markdown and convert them to PDF!].
- **Create an account at GitHub.com.** This will give you free unlimited *public* repositories, but unless you want the whole world to be able to see your projects, click "request a discount" at GitHub edu to get free *private* repositories.
- **Download and install GitHub Desktop.** Then open and log in using your GitHub account.

> 1. Create a NEW repository

Open up GitHub Desktop and log-in, if you haven't already.

Click on `+` and then `create` to make a new repository with a name and location of your choice. This creates a new folder in a directory of your choosing that will be empty except for some hidden files (e.g., a `.git` directory)

Within the app, you should now see the repository you've created on the left. It will have a computer icon next to it, indicating that it's a *local repository*, i.e., only on your computer and not yet synched to GitHub.

> 2. Add a text file to your repository

Now, **leave GitHub Desktop** and go to your text editor. Then:

- Create a new text file on your computer called `README`, and *save it in your repository location*
- This should be a plain text file (`.txt`) or Markdown file (`.md`), NOT a rich text format file (`.rtf`).

> 3. Commit this change in GitHub Desktop

Head back over to GitHub Desktop. The window should now look something like the picture on the slide, with a document highlighted in blue called "README".

The right-hand pane is grey because there have been no changes *inside* the README file; the only change to the repository was adding it.

Let's commit (i.e., officially record) this change by adding a short summary, optional description, and clicking `Commit to master` near the bottom of the window.

> 4. Add text to README and commit changes

Go back to your **text file**:

- Add some words
- Save the text file

Then go back to **GitHub Desktop** and you'll see something new in the window:

- In the right-hand pane, there's the text you added highlighted in green ("+green" means added, "-red" means deleted)
- Essentially, this pane is telling you the *contents* of the changes within the document highlighted in blue

Now, let's **commit this change**, adding a summary and clicking `commit to master` as before.

> 5. Edit README text and commit changes

One more time, go back to your **text editor**:

- Edit your previous text
- Add some new lines of text
- Save

Back in the **Desktop app right-pane**, you will now see:

- Edited or deleted lines in red
- Added lines in green

Even if you only changed one character in the line of your previous text, you'll notice that the whole line was "subtracted" and then "re-added" with the change. This is because Git's unit of operation is the *paragraph*, not the word or character. This can be annoying for long blocks of text, but is OK for lines of code.

> 6. Undo the last commit

If you're unhappy with your LAST commit (i.e., you disliked how it was grouped or labeled), you can **click the** `Undo` **button** at the bottom of the list of uncommitted changes.

Now, these changes will appear again in the **"uncommitted changes"** tab for you to regroup or relabel.

However this DOES NOT actually undo the changes *within* your text file ... for that, you need to **revert**.

> 7. Revert a previous commit

Navigate to the "History" tab within the GitHub Desktop window (top middle). There, you can see all the previous commits you've made, both in list form and as graphical nodes on a branch at the top of the window.

Click on one of the commits in the list, the click the **gear wheel** in the right-hand pane, and select `Revert this Commit` .

- When you *return to the text file on your computer*, you'll see that the last change has been undone
- If you *go back to GitHub*, you'll see that revert has now been recorded as a new commit (so you can revert your revert if you change your mind again).

> 8. Publish repository to your online account

So far, we've been working in a *local repository*—one that that you created on your computer. This may be useful for you, but to collaborate on projects you'll need to publish the repository to the web (i.e., make a *remote repository*).

To do this, **navigate to the top-right of the GitHub Desktop** window and **click** `Publish` . This is also what you'll need to click every time you want to sync your local and remote repositories, but after initial publishing it will (more logically) be called `Sync` .

> 9. View your repository & changes online

When you login to **GitHub online**, you'll see the new repository and file you've added.

- In blue, you'll see the summary name you gave to the last commit
- If you click on the file itself, you will see the text, and have the option to view the full history of commits.

Because the demo README file was written in Markdown, you'll notice that it renders as HTML on the GitHub site—pretty cool.

> 10. Edit the file online & sync with local repository

Clicking on the file also gives you the option of editing it within the browser:

- Make some edits
- Then commit

If you go back to the **Desktop app** you'll see a new commit there; if you go to your **text file**, you will see the changes there as well.

> What's next?

That was super brief and basic. If you want to delve further into Git/Hub, Google and experiment with the following things:

- **Forking online repositories**—Duplicates *someone else's* shared repository and lets you use/change/build on it without affecting their original work. Note that when you do this it will only available to you online, unless you then clone the repository.
- **Cloning online repositories**—Copies an online repository (e.g., one you have forked) onto your local hard drive so you can work offline.
- **Branching a repository**—Lets you (and collaborators) experiment with changes that can be merged into the "master" version of the files.
- **Initiating a pull request**—Submits your commits to be merged into a forked/branched repository, which can then be accepted/rejected by collaborators.

[Dynamic Documents]

> Not covered here, but ...

Another step you can take toward reproducible research—that we don't fully cover here—is to write using **dynamic docs**, which integrate code *into your manuscript.*

Essentially what this means is that, instead of copying-and-pasting results from software output (annoying and error-prone), the results are generated live within your manuscript. Reproducing a paper just requires one or two clicks.

This feature is available via **R Markdown**, **Stata Markdoc** and **Stata texdoc**.

Solutions III. Dissemination

> Steps

You've completed your analysis and written your manuscript, congratulations! Now you're ready to share your research, submitting it to journals, etc. Before you do, you should prep your files for replication to make sure everything works, and then (or as part of publication) submit to an online repository so others can see and replicate your work, and potentially include it in a meta-analysis.

6. Prepare for Replication

> Why do we care if our code is reproducible?

Hopefully by this point, you don't need convincing that replication is important. As a reminder, however, replication is both **necessary for scientific advancement** and will make **you a more credible researcher**.

Preparing code for replication can also be a **great way to catch potentially embarrassing mistakes** in your analysis, before it goes into the public sphere. Ensuring that your code and data are straightforward and easy to use will also help **protect you** against misinterpretations by replicators who may label your study as "non-reproducible" simply because they cannot decipher what you did.

> Replication files should ...

In order to **maximize reproducibility**, replication files should (ideally) be:

- **Be complete but parsimonious**—Your files should contain all data and code needed to run the full analysis, but NOT lots of extra material that may confuse the user or cause misinterpretation. Using solid *file management* and *version control* systems to begin with will help ensure that you don't have lots of extraneous, previous versions of files floating around.
- **Run and reproduce results with one click**—The analysis should run with one (or a minimum number of) clicks. If you've streamlined your code as suggested in the *file management* section, you should already be on your way toward this.
- **Be readable and interpretable by humans**—In order for people to use your code, they must be able to interpret it, otherwise a "failure to replicate" could simply be due to the replicator not being able to follow your precise steps. Again, if you've followed the *literate programming* steps above, this should require less work.
- **Protect personal information**—Finally, any data that is made public must protect the personal information of human subjects. This is something you can deal with during data analysis, but if you haven't yet, now is the time to make

sure PII is protected.

As with many other topics covered here, there is no single, perfect way to organize or prepare files for replication. Do what works for you (as long as it meets the above criteria)!

> 5 Steps for Prepping Files

- **Set-up**—Create a new file structure just for replication files
- **Initial replication**—Before changing anything, make sure that your results duplicate with the original code
- **De-identify**—Tweak code and censor data files as needed to make sure PII is protected.
- **Edit**—Format and streamline the code so it is legible, and document everything
- **Final replication**—Make sure it all works again after your tinkering

> 1. Set Up

Create a NEW, clearly organized folder structure for replication that you add to selectively. This will help ensure that your **replication files are complete and parsimonious** and **protect your original files**.

> Create

Within your project directory, **create a new, empty replication folder* (e.g., "`replication_files/`"). This is can be the same PDEL template described under the *file management* tips, including separate folders for:

- code/
- data_clean/
- data_raw/
- output/
- extra

Also **add a README file**, used to document contents, sources, software/system versions, and other info necessary for replication/comprehension by new users.

> 2. Initial Replication

Next, **copy (don't move!)** the data and code files you need to run the analysis (and nothing else) into the appropriate replication folder and **try to replicate your results** without changing anything except necessary file paths.

Again, the point here is to **make sure your code actually runs as-is**, before doing anything else. Also, like importing selected files to a new computer rather than simply migrating all existing files and settings, this can help **get rid of some deadweight** and unnecessary files, helping you identify what is really needed.

> A. Check Analysis

When doing the initial replication, PDEL suggests you **start with the analysis code** (i.e., the last code files first), using whichever data files your merging code generated the last time you used it.

Working backwards through your code will help you better isolate the origin of any problems.

> B. Check Data Clean/Merge

Once you've fixed any bugs in the analysis code, then follow the same process for the **code used to merge or manipulate data**, copying any new scripts into the `code` folder, and any original datasets into the `data_raw` folder.

> 3. De-Identifying Individual-Level Data

You have now populated your replication file structure and debugged to make sure everything works. Next, take the necessary steps to make sure that any data and code you make public **protects personally identifying information (PII)**.

You need to do this for both **ethical reasons** and to comply with **IRB, legal, and funder requirements**.

> What does "de-identifying" mean?

Data can contain information that allows individuals to be identified in one of two ways:

- **Direct identifiers** are variables explicitly linked to subjects, such as *name, email, address, ID number, phone number, etc*. This is what people typically think of as PII.
- However, **indirect identifiers** can also be problematic. They are variables that, in combination, could be used to identify individuals, such as *gender, dates (birth, program admission, etc.), geographic location (village, GPS), unusual occupations or education, etc.*

> Example of Indirect Identifiers

For example, say that your data is a survey of teachers, and contains variables such as **gender**, **grade-level taught**, and **age.**

You may *think* that this does not constitute PII. However, if there is only ONE *female, third-grade teacher aged 40-49*, **she is not anonymous in your data.**

This is particular problem if your survey contains other sensitive information that could be embarrassing or detrimental to her career, such as opinions about school administration, political views, salary, etc.

> The Problem

You might think that it's very unlikely that someone from the local village where you did the survey will find your data online and parse it to find information about their third-grade teacher. You may be right, but that doesn't mean you don't need to take steps to protect your subjects.

Attempting to identify individuals from public data is something of a sport in certain circles, and there have been high-profile cases of individuals being identified from public data by academics, reporters, and others.

A systematic review of some of these so-called "re-identification attacks" by **El Emam et al** shows how easy this is in some cases, even where researchers have taken steps to de-identify data [see table for highlights].

> Dealing with Direct Identifiers

Dealing with direct identifiers—e.g., name, address, mobile number, ID number—is relatively easy in that they should be censored and **never made public**.

- The simplest solution is to **remove them** from the dataset (i.e., drop these variables). This can be tricky if your analysis or merging relies on them, which we discuss later.
- A second solution is to **pseudonymize** the data, using an algorithm to replace these identifiers with new values that can't be used for re-identification. This can be useful if you need to link individuals across datasets (e.g., rather than matching "Nadia Jones" in Minneapolis across two datasets, assign her to patient ID #2849082 in both datasets)

[*See decision tree on next slide*.]

> What is sufficient de-identification for indirect identifiers?

Dealing with *indirect* identifiers is more complicated. After **looking at your data to determine identifying combinations of information**:

- **Determine Risk:** Think about the level of risk of an individual being identified from your data. This should be determined by an estimate of the probability of being re-identified AND the sensitivity of the data (e.g., some political opinions, familiy history, medical information, etc.). Both the probability of being re-identified and sensitivity are likely to be context dependent.
- **Set a "k-anonymous" threshold appropriate to the risk level** A k-anon threshold is the level at which each record cannot be distinguished from at least $k-1$ other individuals who also appear in the dataset. So, a k=3 level means that any 1 individual cannot be distinguished from 2 others. [*See example on next slide*]
- **Select appropriate method(s) of de-identification:** Based on your k-anon level, figure out the best method of de-identification. This could include aggregating data, removing certain variables or observations, reducing information/detail, and/or adding random noise or values. [*See decision tree in two slides*]

> Trade-off: Anonymity vs. Usefulness

Unsurprisingly, there are trade-offs in the solutions described on the last slide. In roughly increasing order of usefulness/decreasing order of anonymity, this includes:

- **Aggregating**—When you aggregate data from an individual to a higher level, you obviously lose ability to replicate any individual-level analysis. You can still make your (aggregated) data and (individual-level) code public, but no one will be able to verify or replicate your results.
- **Removing variables**—Deleting specific variables that are used in your models may also compromise full replication.
- **Remove observations**—This protects anonymity, but can bias replication results if deleted observations are systematically different on observables/unobservables.
- **Reducing information contained in variables**—For example, changing DOB to an age-range may still allow full analysis, although replication results may be slightly precise.
- **Adding random noise/values**—As with recoding variables to reduce information, this will decrease the precision of replications.

> Good Practices

Although there are no hard-and-fast rules to de-identification—other than not making direct identifiers public!—here are a few good practices when it comes to prepping and sharing de-identified data for replication:

- **Include all code** even if it manipulates/analyzes *identified* data that you can't share, AS LONG AS the code itself doesn't compromise anonymity
  - e.g., censor code that sets the seed for a random draw to generate pseudonymous ID numbers
- If identifiers *aren't* used for analysis, **de-identify early in merging/cleaning process** so that you can share as much data as possible. This may mean re-working some of your code.
- **Store original data with PII securely**—If you're using Dropbox, see PDEL GitHub wiki for tips on sharing in a way that protects PII data, and make sure to follow your IRB requirements!

> 4. Edit and Organize Files for Clarity

Now we have working files that are de-identified; the next step is to clean and annotate them to improve usability. Again, the goal here is to make files **legible** to others in terms of structure and content.

> Basic steps

Most of these steps **reiterate the literate programming tips** discussed above, regarding file names, structure, and streamlining and annotating code.

In addition, you'll want to make sure that everything is **well documented** using comments as well as completing the README file.

> Document File and Folder Content

- Make sure you have a **complete** README file, that describes the contents of replication folders.
- If necessary, include a supplemental **codebook** in the `extra/` folder
- Also make sure to **document packages and software versions** (in README and/or code), which can be a huge stumbling block for replication

> 5. Final Replication

Almost done! Now that you have made sure everything works, de-identified data, and edited and documented your files, now is the time to perform one **final run-through** of your code to make sure nothing has broken.

In general it's a good idea to **shutdown your stats programs** in order to **clear the memory** before rerunning the entire process. Alternately or in addition, you can ask a (very generous) friend or an RA to do this for you, which may catch additional errors due to different OS's, configurations, not-installed packages that you forgot to document, etc.

Once any remaining bugs are fixed, you are good to share.

7. Share Data and Code

> About Sharing Data and Code

Before, during, or after submitting your manuscript, upload your replication files to an online repository. Although uploading files to a personal website is a good first step, storing them a **public repository** has some additional benefits:

- Repositories usually last longer than personal website and are thus more future proof
- They are also more searchable, and a better way to publicize your work and make sure it's accessible to a larger audience

Some people are shy about sharing data and code online early in a project, for fear that their data or ideas may be **stolen and used by others**. While this can be a risk,

- Repositories often let you **embargo materials** for a period of time, while still getting points for transparency (something a personal website won't let you do)
- You can always **publish only what is necessary** for replication, keeping other variables (e.g., unused survey Qs) to yourself for later use
- Furthermore, the **biggest risk isn't having your data/ideas stolen, it's having your research ignored!** (King 1995)

The only data—other than identified data—that you may not be able to share is anything **proprietary**. In this case, share code and supplementary data files where possible.

[*See where to share on next slide*]

8. Meta-Analysis

> About Meta-Analysis

Once your files have been made public, other researchers may try to replicate your results.

In addition, it's possible that your paper could end up as part of a **meta-analysis**—a statistical analysis of a group of studies that derives a pooled estimate of the effect of a treatment. Oftentimes, meta-analyses are part of a "systematic review" of literature on a given treatment and outcome (e.g., the effect of providing free bed nets on mortality rates from Malaria).

Meta-analyses are useful because, as we know, any estimate from an individual study may be biased or significant by chance—**pooling these results together can help average out this noise**. However, note that because only published results are generally included in meta-analyses, they are still subject to publication bias.

> One Study = One Data Point

Even though may have thousands of observations in our research design, the end result of typical quantitative work is one (or a handful) of point estimates. Each of these represents **a single observation in the universe of results from related studies**.

These results be due to **random chance, bias, or various decisions** you made as a researcher during study design and analysis (RDFs).

> Even with the same data, results may vary ...

To illustrate the dangers of relying on *one* study, and the benefits of replication and meta analysis, an OSF-registered project gave the same dataset to 29 research teams and asked them to estimate the effect of race on being given a red card in soccer matches.

As this **chart** shows, both the effect size and significance of the estimates vary, with some researchers finding no effect and others finding that dark-skinned players were between 1 and 3 times more likely to receive a red card than light-skinned players.

> Basic Steps

A full discussion of meta-analysis methods is beyond the scope of this presentation. However, the basic steps begin with developing a **"protocol"** (similar to a PAP) that specifies:

- Inclusion and exclusion criteria for studies
- Measurement of key outcomes (e.g., how to deal with discrete vs. continuous data, etc)
- Models to be used for "meta-regression" (e.g., RE, FE, etc.)

This protocol is then typically registered with one of a meta-analysis repository [see two slides down], before you use it to complete the meta-analysis.

> Funnel Plots

What does a meta-analysis look like? Once common output is the **funnel plot**, which is essentially a scatter plot of study effect sizes vs. the precision of the study (e.g., SE of treatment effect).

The chart on this slide is from a useful guide to funnel plots published by the BMJ and available here.

> Who Does Meta-Analysis?

If you're interested in learning more about meta-analysis, or even doing one yourself, check out organizations like the **Cochrane** and **Campbell groups** (particularly for psych and medical trials), **3ie** (for development), and others.

Extra

> Solutions at the Institutional/Disciplinary Level

This presentation has focused on individual-level solutions to the credibility problem in the social sciences. However, there are lots of other disciplinary and department-wide solutions that we can contribute to as researchers. This includes:

- **Design-based publication** (also known as "registered reports"). Here, journal articles are peer reviewed based on the merit of their theory and research design, *before* results are analyzed or known. This can help with publication bias and reduce incentives for p-hacking.
- The discipline can incentivize transparency, replication, meta-analysis, through **prizes and awards** by organizations like BITSS and OSF.
- Journal editors can adopt and enforce new **norms and standards** for data sharing and reproducibility.
- Scholars can take advantage of many opportunities for **training** to improve their transparency toolkits, such as those offered by BITSS.
- Academic departments and universities can change their **tenure requirements** to reward data sharing and reproducibility