Reproducible Coding Strategies and Intro to Version Control (Github)

Fernando Hoces de la Guardia
BITSS

-

Slides at https://goo.gl/aBQ3LR

American Institutes for Research, May 2018

Version Control

The Claerbout Principle

> *An article about computational science in a scientific publication is not the scholarship itself, it is merely advertising of the scholarship. The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures.*

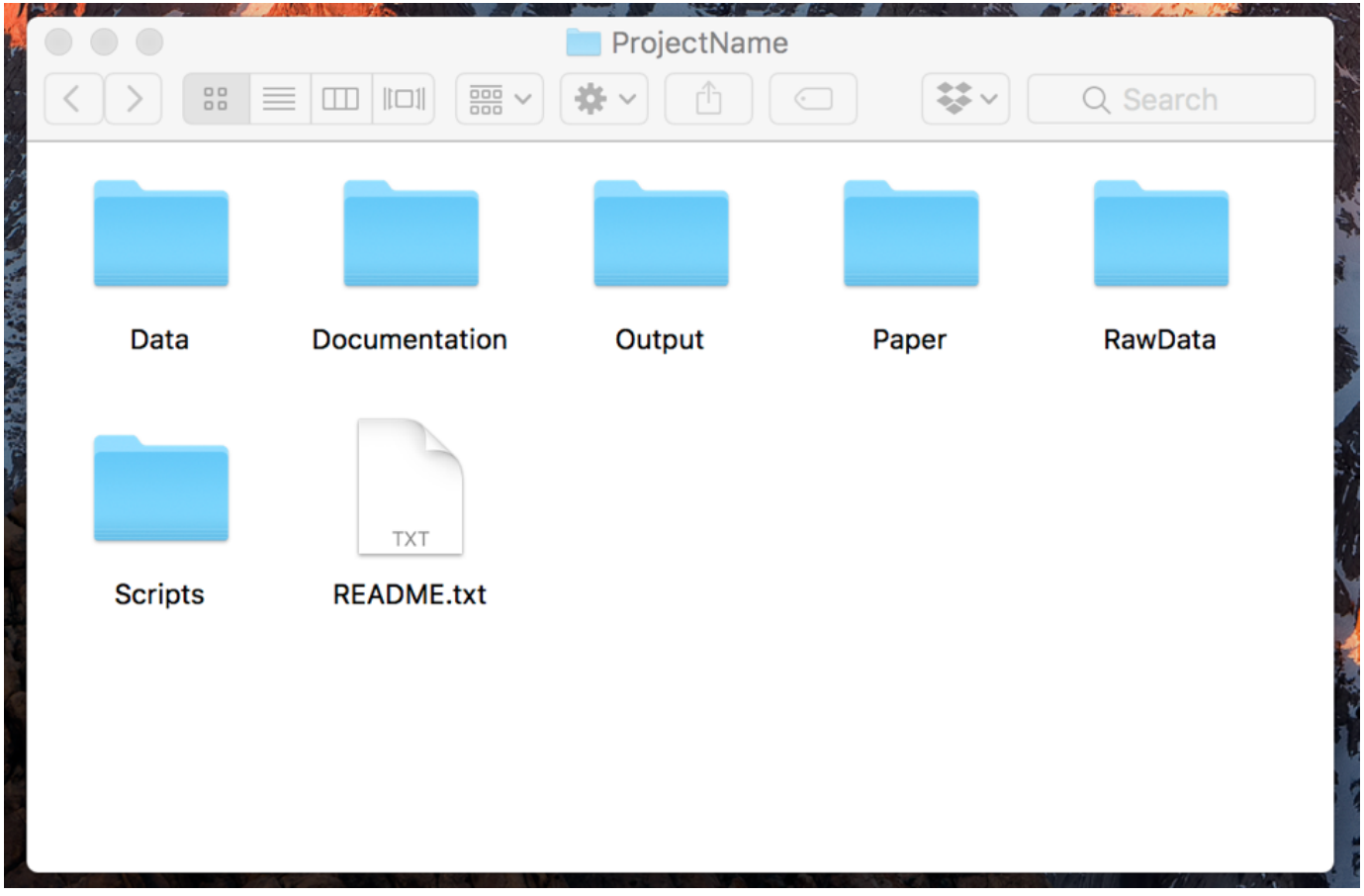
Buckheit & Donoho, 1995

Organizing Principles

1 - Use code (scripts), don't work by hand (Excel/spreadsheet, GUIs).
2 - Consider not saving statistical output, and just saving the code and raw data that generates it.
3 - Reproducibility–on your own machine across multiple runs, across machines, across researchers.

File Management & Coding Suggestions

Begin with a logical file structure



General Coding Suggestions I

Christensen, Miguel & Freese (2018)

- Make sure script files are self-contained: don't write code that only works if you run a group of other files previously in a specific order and then leave things hanging precariously.
- Include tests in your code. This can alert you if output changes.
- You can never comment your code too much. Truly explain rather than transliterating: `x=1` as "initialize the population count to 1" or "set x equal to 1."
- Indent your code.

General Coding Suggestions II

- Once posted, any changes at all require a new file name. Better: use version control.
- Separate your data cleaning and analysis files. Don't make any new variables that need saving (or will be used by multiple analysis files) in an analysis file. It is far better to only create a variable once so you know that it is identical when used in different analysis files.

General Coding Suggestions III

- Name variables informatively: pick a side for indicator variables `dead"` (`or`living") instead of "status". (gender, race, etc.)
- Don't leave clutter around-delete temporary or unnecessary intermediate objects.
- You can use a prefix such as `x_` or `temp_` so you know which files or variables can easily be deleted later. Stata also has the `tempfile` and `tempvar` functionality.
- Every variable should have a label. (If allowed for by the program.)
- Use relative directory paths (such as `./Data` not `C:/Users/garret/Documents/Project/Data`)

Version Control

The Primary Goal of Version Control (for us)

The Goal: keep track of any potentially meaningfull modification to your code.

Strategy 1:

1 - Agree on a naming convention with you co-authors (eg: YYYYMMDDfilename_INITALS).
2 - Begin working from the last saved version (eg: `20180325demo_FH.do`).
3 - At the end of the day, save on a new version (eg: `20180327demo_FH.do`).

Pros: Easy adoption.

Cons: Error prone, hard to document, lots of files for each document.

Strategy 2:

1 - Name your file `filename` (ideally `01_filename`)
2 - Take a snapshot of your work every time you complete relevant change (day, hour or minutes).
3 - Update your entire working folder to the cloud.

Pros: Error proof, seemless documentation, one file per document, track differences across all versions, meant to work with the cloud.

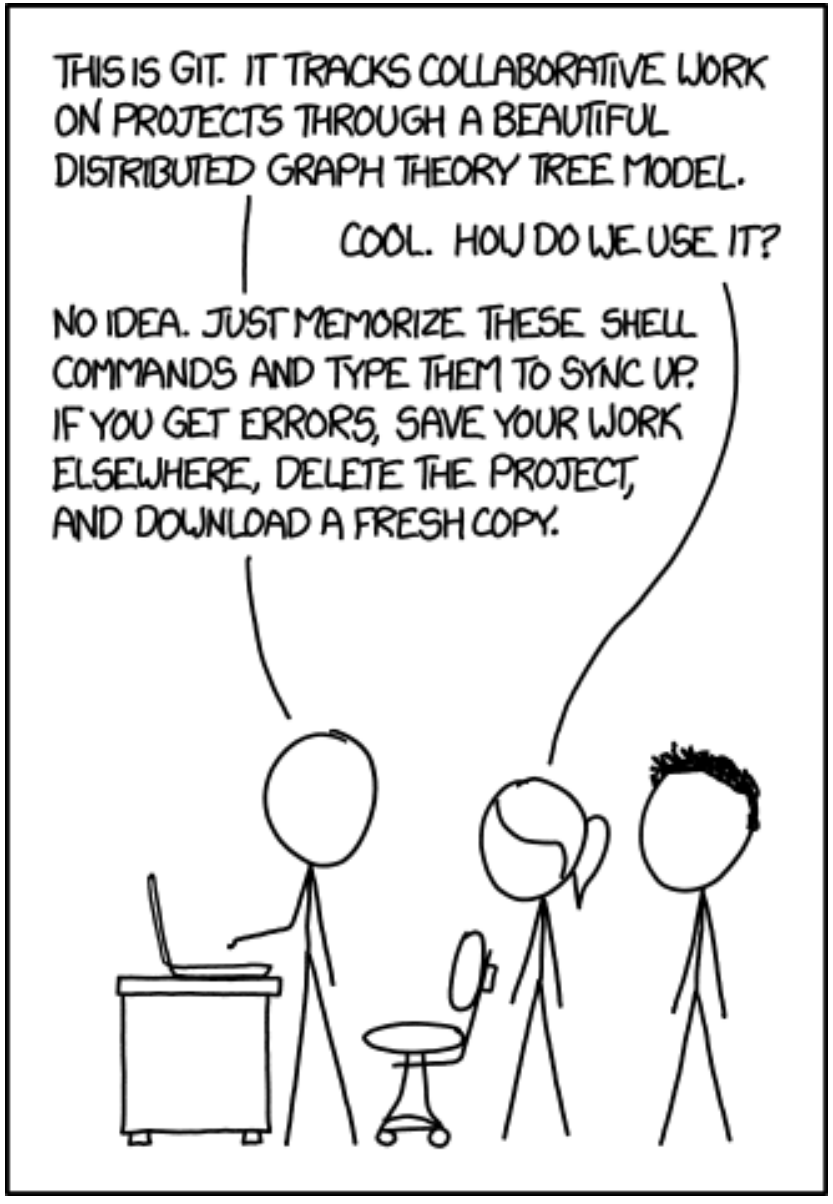
Cons: Harder adoption.

We want to avoid this situation:

Other reasons to use git

- To access a whole new world of knowledge!
- Great tool for collaboration.
- Easier to test all sorts of ideas/models.

Managing expectations



Tips to ease the adoption: concepts [MAKE HANDOUT].

- **Git** is the software that does all the magic. **Github** is an implementation of Git that is easier to use, provides free (public) cloud service, and tools for collaboration.
- A repository (**repo**) is a master folder that contains all your work.
- Whenever you take a snapshot of your *saved* work, you **commit**
- When you make changes to your files in your computer, you are working **locally**, whenever you make changes to the files in the cloud you are working **remotelly**.
- Whenever you want to update your *remote repo*, you **push**. If you want to update you *local repo*, you **pull**.
- When you copy a repo from another person into your online github account, you **fork** it.
- When you start a repo from github.com (just created or forked), and want to download it for the first time to your local computer, you **clone** it.

Short Demo: We Start in the Cloud

1- Create github.com account and sign in.
2- Let's look at some **repos**.
3- First way to access content: download.
4- What if you want to have your own copy of the repo? Fork it!.
5- Now create your own repo. Initiate readme and make some edits.

Short Demo: We move to our local computer

6- Clone the it. Explore the files and location.
7- Create new files, edit. And commit. Edit again, and commit again.
8- Push. Edit on github.com, and pull. 9- Simulate conflict and start from a fresh copy!

Want to learn more:

- Great 20 min intro to Git by Alice Bartlett
- Great 2hr tutorial to Github by Jenny Bryan (git ninja)
- Documentation from Matthew Gentzkow Jesse Shapiro
- Come to 3-Day trainning (RT2) in Seattle (in September, repo from last year)!