Git for Non-Programmers

Fernando Hoces de la Guardia
BITSS

-

Slides at https://github.com/BITSS/RT2_DC_2019

Washington DC, September 2019

Reproducible Workflow

Version Control

Demos

Reproducible Workflow

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

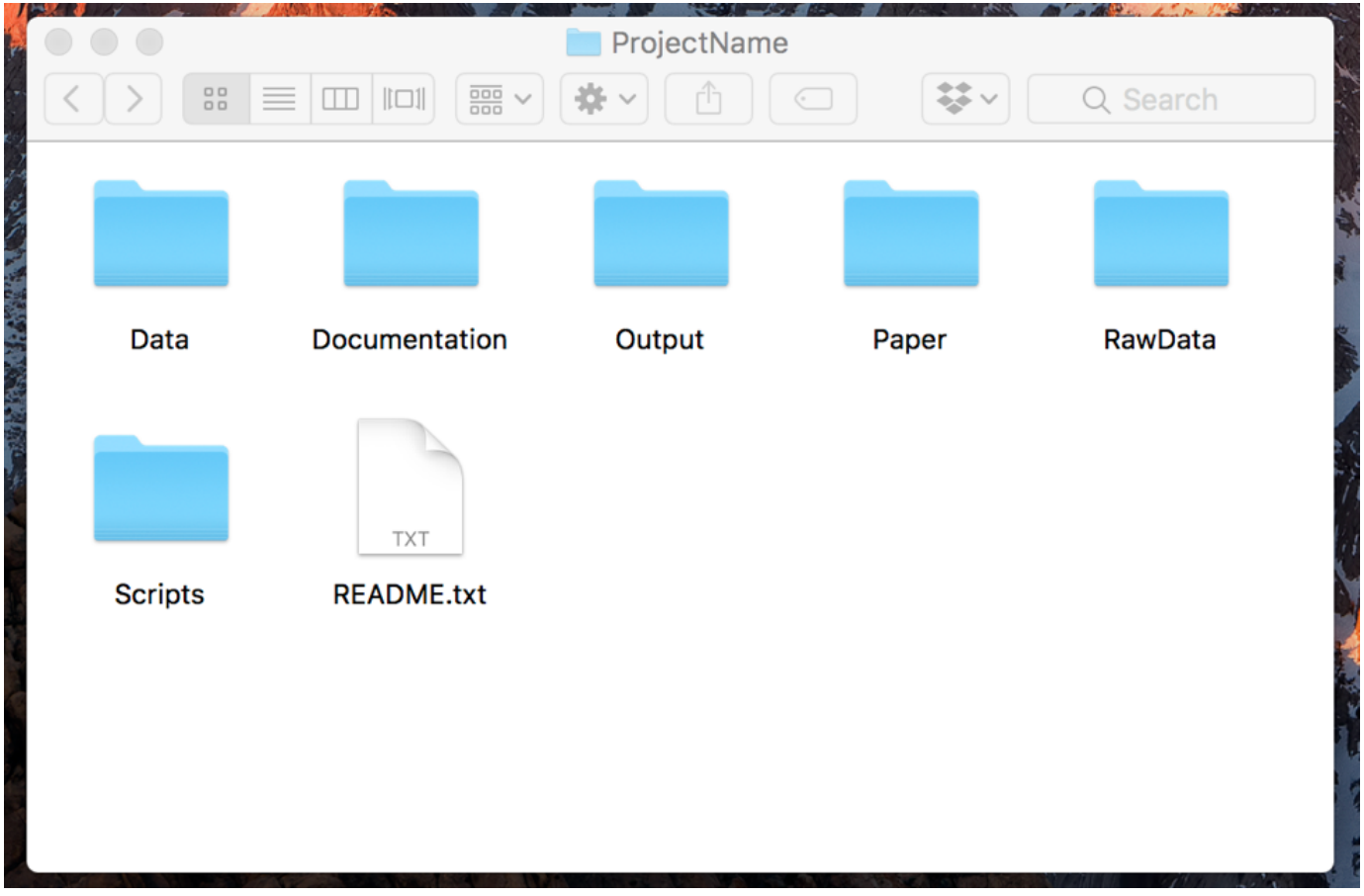
Organizing Principles

Christensen, Miguel & Freese (2019)

1 - Use code (scripts), don't work by hand (Excel/spreadsheet).
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



Version Control

Git/Github for Version Control

- Git and Github are tools to track the complete history of your files.
- They are very popular among programmers, but not so much among non-programmers.
- Why? I believe it has to do with GUIs.

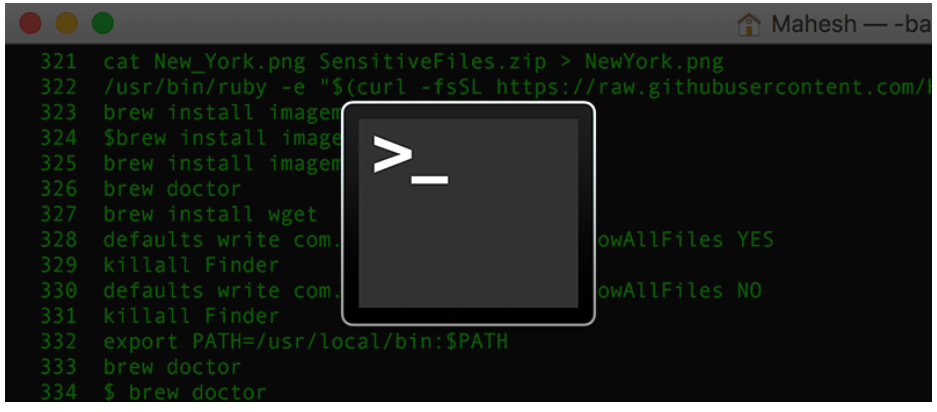
What is a GUI and why the bad reputation

Graphical **U**ser **I**nterface

- For most of us (non-programmers): *GUI* = *Software*.
- GUIs are behind the popularization of personal computers.
- Unfortunately GUIs are pretty bad at keeping a record of actions taken (bad for reproducibility).

What is not a GUI?

- Any software that is run in the command line (aka terminal, shell, bash, etc).



- Git was designed to run in the command line.
- Today we will learn Git **without** the command line.

What is Git 1/2

- Git is a software designed to track the **entire** history of the code of a project.
- Designed originally for software development, it has gained important traction in the research community.
- Main appeal: facilitates full reproducibility and collaboration.
- Git is mainly meant to work as a non-GUI (in the command line) software.
  However: most of the key features can be used through a GUI.

What is Git 2/2

- By code git understands any type of plain text file (`myfile.R`, `myfile.do`, `.tex/.md/.txt/.csv/.etc`).
- This types of files can be understood as "human readable" as machine and human see the same fie.
- Files that are "non-human readable" are called binary files (`myfile.docx`, `myfile.xlxs`, `.pdf/.exe/.dta/.etc`).
- Git can also detect changes in binary files, but it cannot show those changes.

What is Github

- Github is a company that provides two services (that we care of):
  - A web hosting service for all our files track with git (public free/private $ or free if academic).
  - A GUI software (Desktop App) that provides user friendly access to git.
- Others hosting ss include: Bitbucket, GitLab, Gitkraken, etc.
- Other GUIs include: SourceTree, Gitkraken, Atom, RStudio.

The Primary Goal of Version Control (for us)

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

The Primary Goal of Version Control (for us)

The Goal: keep track of any potentially meaningful modification to your code.
Secondary Goal: Learn how to collaborate with others using Github.

The Primary Goal of Version Control (for us)

The Goal: keep track of any potentially meaningful modification to your code.
Secondary Goal: Learn how to collaborate with others using Github.

Bonus track: get you excited about using open source statistical software (R, Python, Julia, etc)

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:

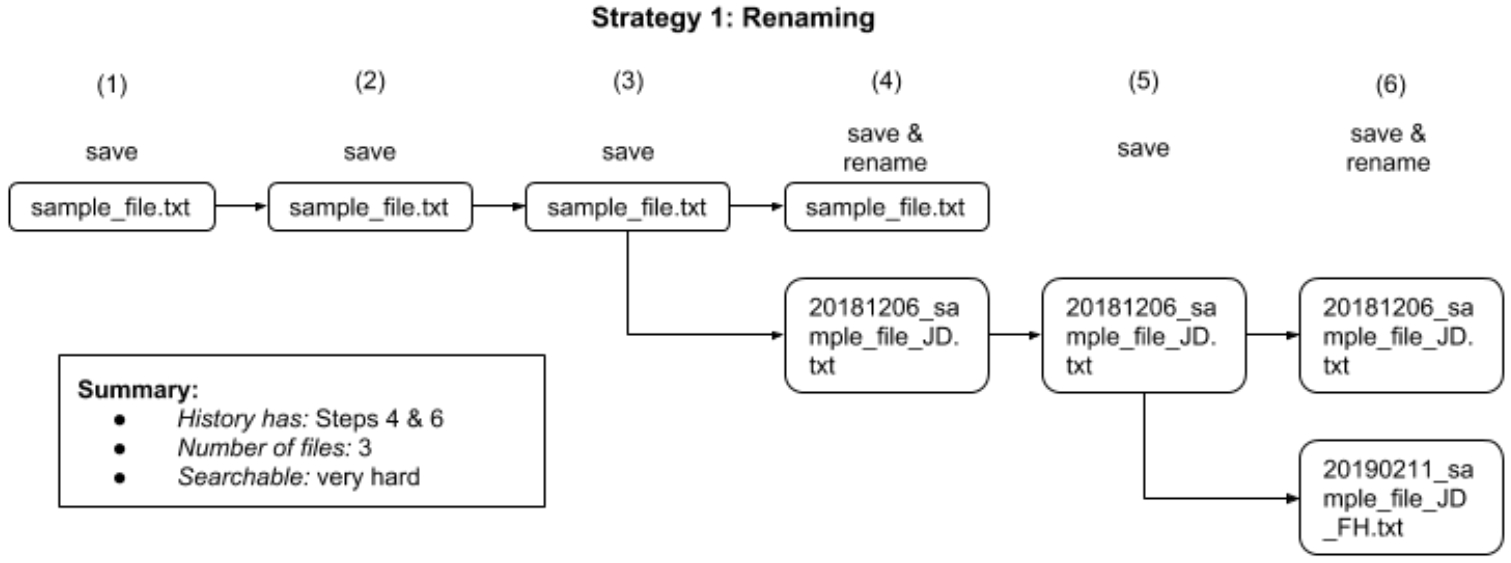
1 - Name your file `filename` (ideally `01_filename`)
2 - Take a snapshot of your work every time you complete relevant change (day, hour or minutes).
3 - Update your entire working folder to the cloud.

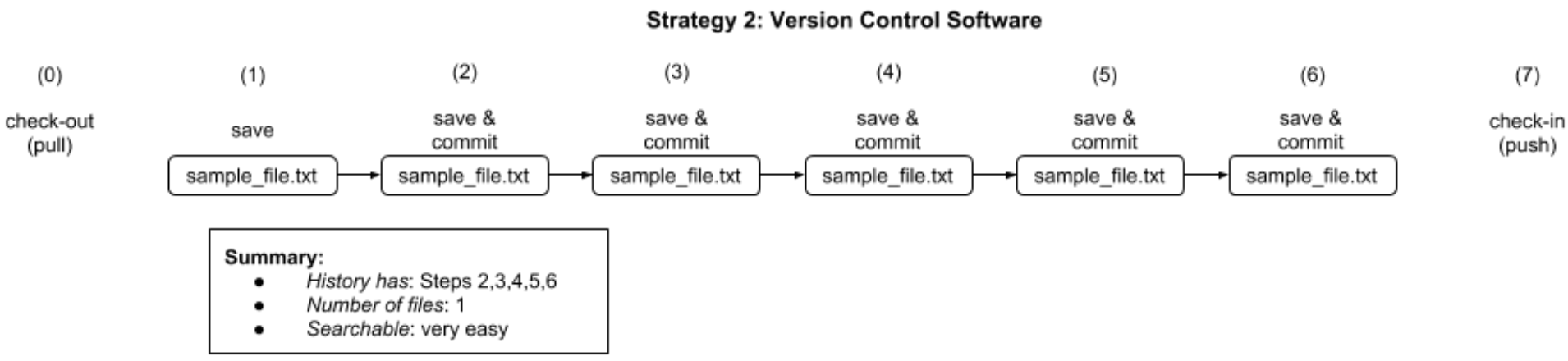
Pros: Error proof, seamless documentation, one file per document, track differences across all versions, meant to work with the cloud.

Cons: Harder adoption.

We want to avoid this situation:

Comparison of Workflows

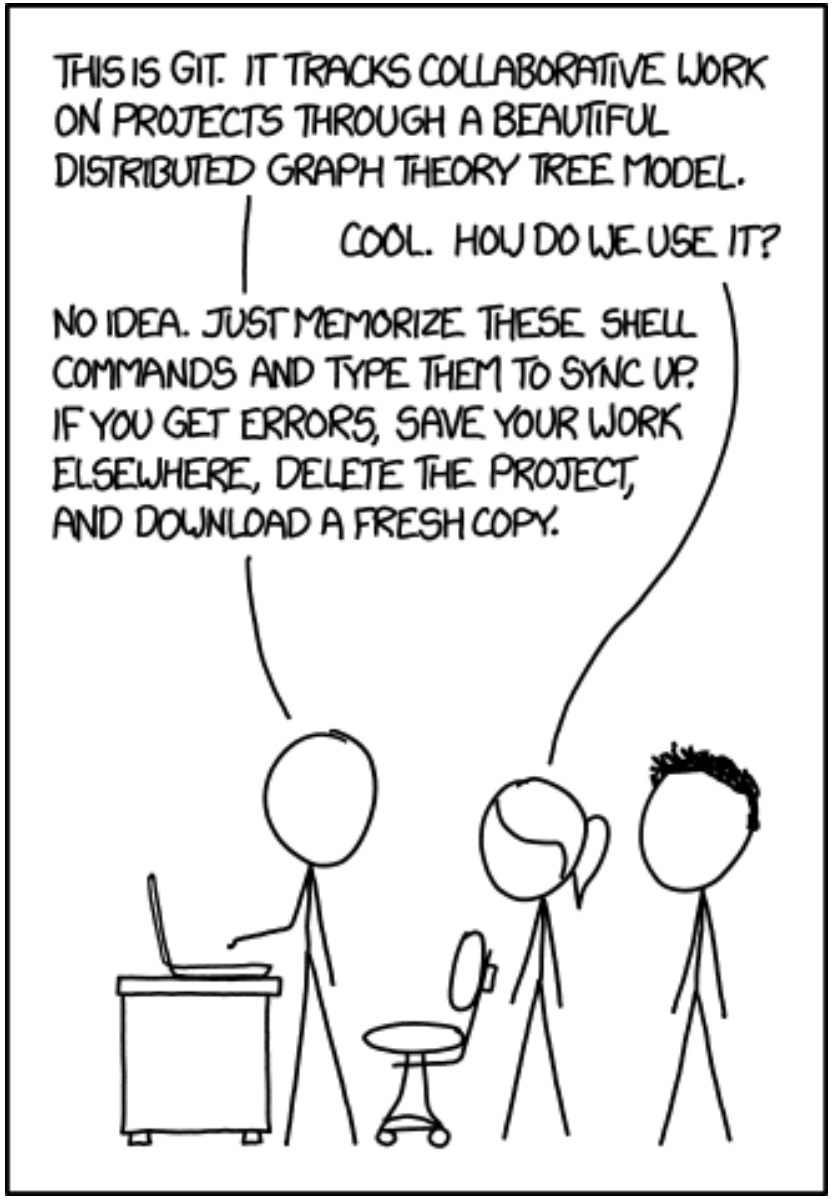




Other reasons to use git

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

Managing expectations



Demos

Three Demos:

1 - **Simple but instructive.**

2 - Repeat but with a slightly more fun example. Collaborate.

3 - Repeat with a real-life example.

Demo #1: 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.

Demo #1: 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- For this tutorial, best way to access previous version: explore in github.com and download.

Three Demos:

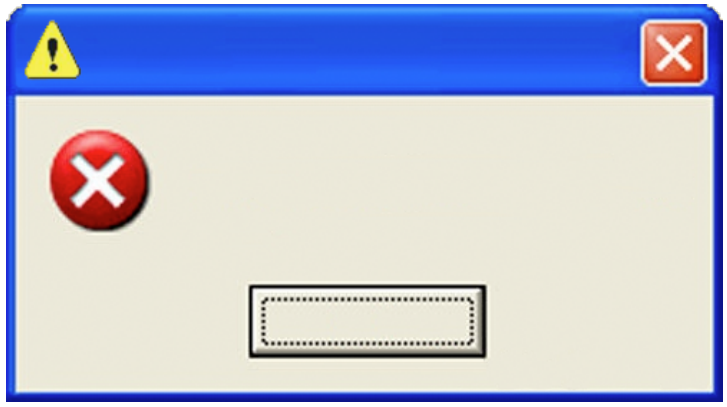
1 - Simple but instructive.
Review: def repo, github.com, download, clone, destination folder, fork, create repo, commit, push, pull, delete, search repo, download old version.
2 - **Repeat but with a slightly more fun example. Collaborate.**

3 - Repeat with a real-life example.

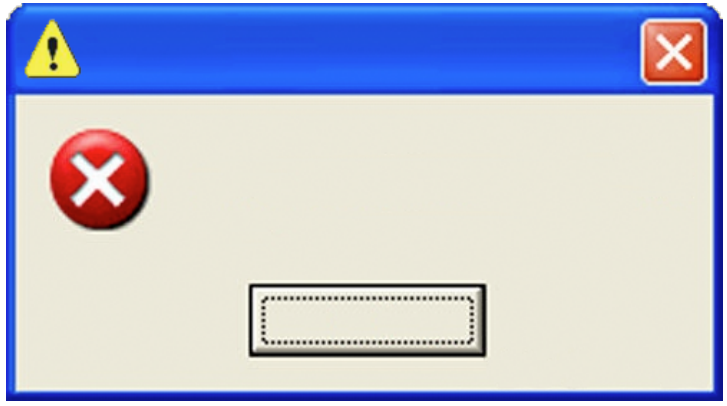
Demo #2: Branches and collaboration (we wil be here a while)

1- Create a branch from previous repo.
2- Add new content (do not replace), commit a few times, and go back and forth to the main branch. 3- Go back to main branch (master), observe file, merge.
4- Look at the history of the main branch.
5- Repeat 1-3 but now replace instead of adding content.

Demo #2: Fatal Error!



Demo #2: Fatal Error!



Burn it and start with a fresh copy!

Demo #2: Branches and collaboration

6- Fork repo `github.com/BITSS/test2`, and clone it into your machine.
7- Edit fields of name, and birth date.
8- Save, commit and push.
9- Create your first Pull Request.
10- Let's see if I can manage all those pull requests very quickly.

Talk about two modes of collaboration

- One owner, many pull requests.
  - Easier to control, requires constant updating.
- Many owners, all can push.
  - **Very** important to pull every at the beginning and at before each push.

10- Create a new repo, invite a collaborator, edit, commit, push/pull. 10.1 - One of you adds . . .
10.2 - The other clones, commits and pushes. 10.3 - The other pulls, commits and pushes. (different lines) 10.4 - Repeats 10.2 - 10.3 but in same lines. 10.5 - Repeat now but with sharedbranches.

how do you distinguish between raw and analytic data

Demo #3: Look inside a real-life project (and collaborate!)

1- Find the following repo: `github.com/BITSS/opa-wealthtax`.
2- Fork it and clone it.
3- Open it in your computer: `nber_trends.Rproj` (needs RStudio), look around and execute up to the section (`Verifying gender [WORKSHOP SECTION]`).
4- Find elasticities, fill in csv, document, submit.

Demo #X: Look inside a half-way project (and collaborate!)

Description:

- Half baked project, forgotten from a few years.
- Exploratory analysis of publication trends in NBER working paper series. Back then inspired by a paper from DellaVigna and Card.
- Now there is more literature around this: Chari and Goldsmith-Pinkham.
- I will use github to share my work with you, do a little exercise, and invite you to collaborate.

Demo #X: Look inside a half-way project (and collaborate!)

1- Find the following repo: `github.com/fhoces/nber_trends`.
2- Fork it and clone it.
3- Open it in your computer: `nber_trends.Rproj` (needs RStudio), look around and execute up to the section (`Verifying gender [WORKSHOP SECTION]`).
4- Generate random number like this: `num1 = sample(20000, 1)`.
5- Look the name and (imputed) gender of the author in row `num1`. This is done by typing:
`temp3[num1,c("name", "gender")]` in the console.
6- Create the following line at the end of the script:
`verification <- cbind(temp3[num1,c("name", "gender")], "rownum" = num1, "correct" = 1)`.
7- Save, commit, push and create a pull request.
8- Feel free to look around create more contributions if you like. Happy to co-author.

Three Demos:

1 - Simple but instructive.
Review: def repo, github.com, download, clone, destination folder, fork, create repo, commit, push, pull, delete & restart, search repo, download old version.
2 - Repeat but with a slightly more fun example. Collaborate.
Review: All of the above, plus: branch, merge, resolve conflicts, collaborate: same proj, fork model+PR .
3 - Repeat with a real-life example.
Review: All of the above, plus: how does a real-life example looks like.

Now go and explore!

Some good habits:
- Commit often ($<$1hr)
- Always pull before you start a new session of work. Also good to pull before pushing.
- Think of your remote as the most important set of files. Get used to deleting things in your local machine.

Want to learn more:

- Great 20 min intro to Git by Alice Bartlett
- Great 2hr tutorial to Github by Jenny Bryan (git ninja)
- Jenny Bryan's Happy Git; Documentation from Matthew Gentzkow and Jesse Shapiro; Karthik Ram's paper on Git for Research
- Software Carpentry's step-by-step tutorial (command line).