GUI Git

Fernando Hoces de la Guardia
BITSS

-

Slides at
https://github.com/BITSS/RT2_LOSANGELES_2018

Los Angeles, September 2018

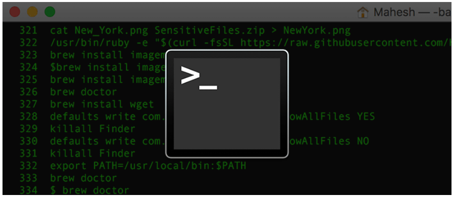
What is a GUI and why the bad reputaion

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

- For most of us (non-programmers): *GUI* = *Software*
- GUIs are behind the popularization of personal computers.
- Unfortunatley 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)



```
321  cat New_York.png SensitiveFiles.zip > NewYork.png
322  /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/
323  brew install imagem
324  $brew install image
325  brew install imagem
326  brew doctor
327  brew install wget
328  defaults write com.                    owAllFiles YES
329  killall Finder
330  defaults write com.                    owAllFiles NO
331  killall Finder
332  export PATH=/usr/local/bin:$PATH
333  brew doctor
334  $ brew doctor
```

- Git was designed to run in the command line

What is Git 1/2

- Git is a software that whose purpose is 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 commmand 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 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.

Coding Suggestions

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.

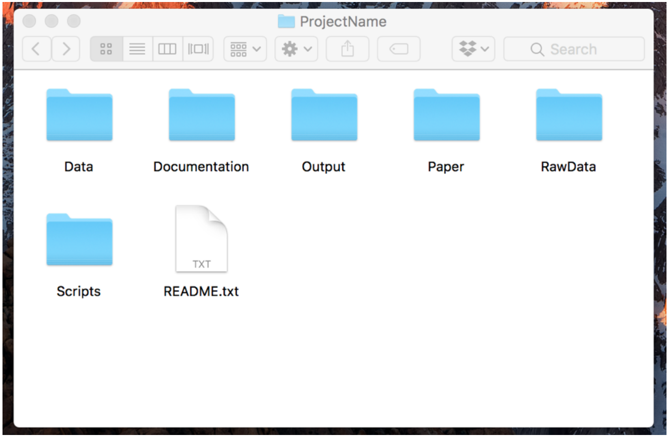
Buckheit & Donoho, 1995

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/fernando/Documents/Project/Data`)

Version Control

The Primary Goal of Version Control (for us)

The Goal: keep track of any potentially meaningful 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.

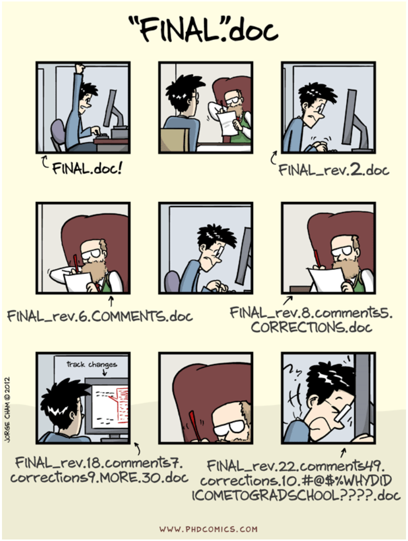
Strategy 2:

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

Cons: Harder adoption.

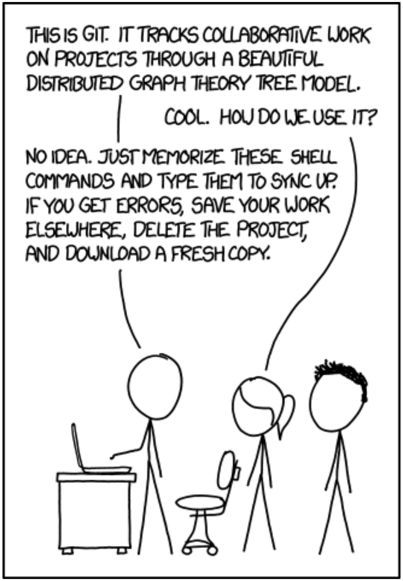
We want to avoid this situation:


"FINAL".doc
FINAL.doc!
FINAL_rev.2.doc
FINAL_rev.6.COMMENTS.doc
FINAL_rev.8.comments5.CORRECTIONS.doc
track changes
FINAL_rev.18.comments7.corrections9.MORE.30.doc
FINAL_rev.22.comments49.corrections.10.#@$%WHYDID ICOMETOGRADSCHOOL????.doc
JORGE CHAM © 2012
WWW.PHDCOMICS.COM

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


THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

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!

Three examples:

1 - Boring 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 code 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 Jesse Shapiro; Karthik Ram's paper on Git for Research