

Reproducible Research and Version Control

JSC 370: Data Science II

January 15, 2024

Replicability vs Reproducibility

Replicability

- Repeating a study by independently performing another study

Reproducibility

- Generating the exact same results when using the same data
- If we can't reproduce a study, how can we replicate it?

Reproducibility

There are several exponential growth curves happening:

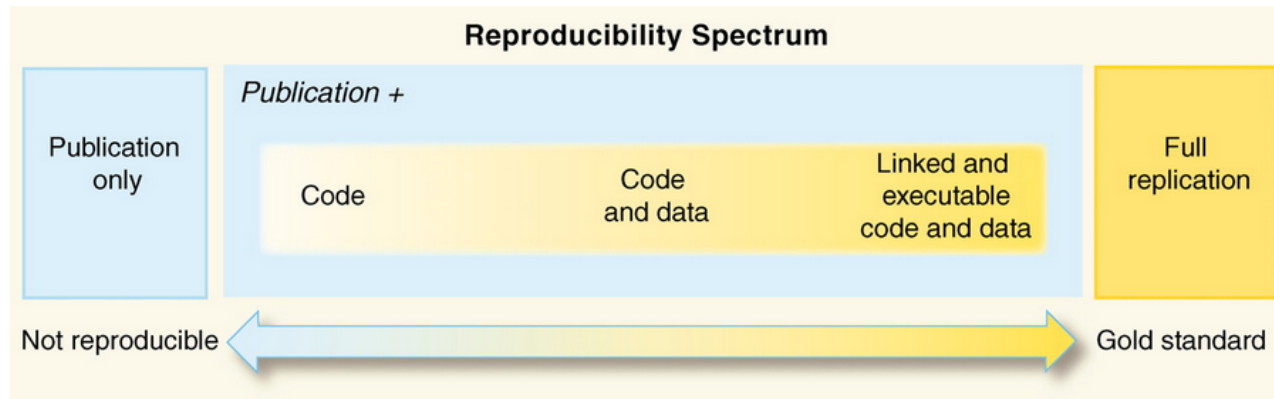
- Data production
- Data storage and transfer
- Computing power
- Complexity of methods
- Published research

Reproducibility

Barriers to doing reproducible work:

- Lack of awareness
- Tradition or common practice
- Tools and training
- Time

Reproducible Research



Reproducible Research

In computational sciences and data analysis, what is reproducibility?

- The data and code used to make a finding are available and they are presented in such a way that it is (relatively) straightforward for an independent researcher to recreate the finding.

Reproducible Research

This actually seldom happens. Consider two interesting articles by Tim Vines:

- The Availability of Research Data Declines Rapidly with Article Age “of 516 articles published between 2 and 22 years ago...the odds of a data set being extant fell by 17% per year.”
- Recommendations for utilizing and reporting population genetic analyses: the reproducibility of genetic clustering using the program structure “we reanalysed data sets gathered from papers using the software package ‘structure’... 30% of analyses were unable to reproduce the same number of population clusters.”

Reproducible Research

Scientific articles have fairly detailed methods sections, but those are typically insufficient to actually reproduce an analysis. Roger Peng and Stephanie Hicks [write](#) "Reproducibility is typically thwarted by a lack of availability of the original data and computer code."

Scientists owe it to themselves and their community to have an explicit record of all the steps in an analysis done at a computer.

Reproducible Research Do's

- Start with a good question, make sure it is focused and it is something you're interested in.
- Teach your computer to do the work from beginning to end!
- Use version control.
- Keep track of your software environment, from what is in your toolchain (software: Python, R, Tableau) to version numbers.
- Set your seed for any random number generation or sampling! This is needed when splitting up your training and test sets.

- Think about the entire pipeline.

Reproducible Research Dont's

Don't do things by hand!

- Editing spreadsheets to clean it up
 - Removing outliers
 - QA/QC
 - Validating
- Editing tables or figures
- Downloading data from a website by clicking links in a web browser

- Splitting data and moving it around
- If anything is done by hand because there is no other way, document

Reproducible Research Don't's

Don't use point and click software or other interactive software if you can avoid it.

- This type of work is not easily reproduced because there is no trace of the steps. If you have to use it, write down the steps!
- Save output. Save the data and code that generated the output, rather than the output itself.

Reproducible Research Challenges

- Data size
 - Try to build in your code tools that help with this, for example parallel processing
 - Can store in smaller chunks and write code that pulls data files automatically, combining them when needed for analysis
 - Write meta data, use tools that help with data organization

Reproducible Research Challenges

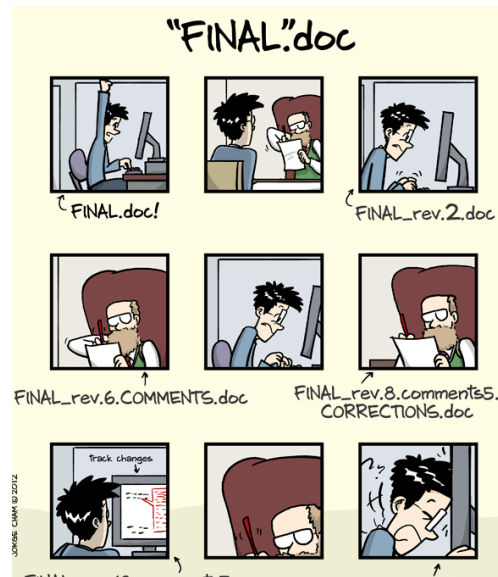
Data complexity

- Try to incorporate smaller snippets of data in your workflow to check reproducibility
- Training, validation sets
- Diagnostic visualizations

Workflow complexities

- Use readme files!!

What is version control?

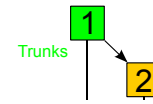


FINAL_rev.18.comments+ FINAL_rev.22.comments49.
corrections9.MORE.30.doc corrections.10.#@\$%WHYDID
ICOMETOGRADSCHOOL?????.doc

WWW.PHDCOMICS.COM

What is version control?

[I]s the management of changes to documents [...] Changes are usually



Why do we care?

Have you ever:

- Made a change to code, realised it was a mistake and wanted to revert back?
- Lost code or had a backup that was too old?
- Had to maintain multiple versions of a product?
- Wanted to see the difference between two (or more) versions of your code?

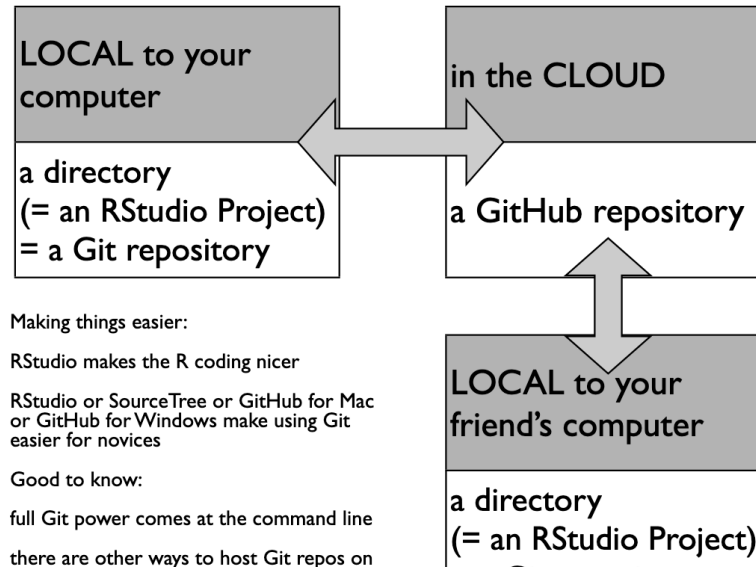
- Wanted to prove that a particular change broke or fixed a piece of code?

Why do we care? (cont'd)

- Wanted to submit a change to someone else's code?
- Wanted to share your code, or let other people work on your code?
- Wanted to see how much work is being done, and where, when and by whom?
- Wanted to experiment with a new feature without interfering with working code?

In these cases, and no doubt others, a version control system should make your life easier.

Why do we care? (cont'd)



Git: The stupid content tracker



Git logo and Linus Torvalds, creator of git

Git: The stupid content tracker

- During this class (and perhaps, the entire program) we will be using [Git](#).
- Git is used by [most developers in the world](#).
- A great reference about the tool can be found [here](#)
- More on what's stupid about git [here](#).

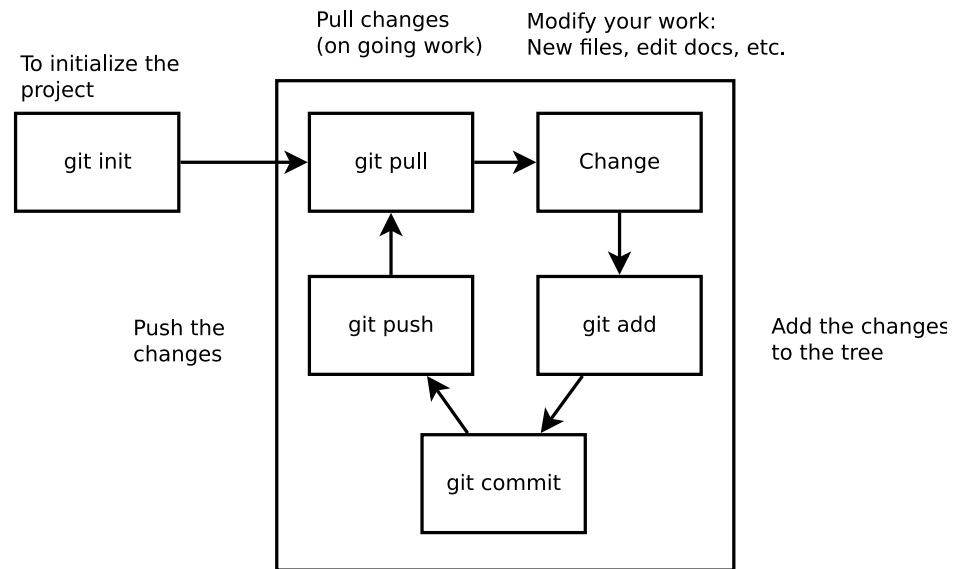
How can I use Git

There are several ways to include Git in your work-pipeline. A few are:

- Through command line
- Through one of the available Git GUIs:
 - RStudio ([link](#))
 - Git-Cola ([link](#))
 - Github Desktop ([link](#))

More alternatives [here](#).

A Common workflow



A Common workflow

1 Start the session by pulling (possible) updates: `git pull`

2 Make changes:

a) (optional) Add untracked (possibly new) files: ``git add [target file]``

b) (optional) Stage tracked files that were modified: ``git add [target file]``

c) (optional) Revert changes on a file: ``git checkout [target file]``

3 Move changes to the staging area (optional): `git add`

A Common workflow

4 Commit:

- a) If nothing pending: ``git commit -m "Your comments go here."``
- b) If modifications not staged: ``git commit -a -m "Your comments go here."``

5 Upload the commit to the remote repo: `git push`.

Note: We are assuming that you already [installed git in your system](#).

Hands-on 0: Introduce yourself

Set up your git install with `git config`, start by telling who you are

```
$ git config --global user.name "Meredith Franklin"  
$ git config --global user.email "mfranklin@email.com"
```

If you have already set up git previously, you can check your settings

```
$ git config --list
```

(to get out of the list in terminal, press q)

Try it yourself (5 minutes) (more on how to configure git [here](#))

Hands-on 1: Remote repository

We will start by working on our very first project. To do so, you are required to start using Git and Github so you can share your code with your team. For this exercise, you need to:

- a. Create an new (empty) repository on GitHub (you can try `JSC370`). Make sure to include a README.md
- b. Go to the local directory where you want to store the files for this repo.
- c. Clone the repository (in GitHub copy the repo link) `git clone https://github.com/...`.`
- d. Back in terminal, edit the README.md. You can use nano in the terminal or open in another app such c
- e. Add the edited README.md file to the tree using the `git add`` command, and check the status.
- f. Make the first commit using the `git commit`` command adding a message, e.g.

```
$ git commit -m "My first commit ever!"
```

Hands-on 1: Remote repository

You can use `git log` to see the history.

You can also use `git status` to see the list of items that might be pending in your `git` workflow.

Hands-on 1: Remote repository

The following code is fully executable (copy-pastable)

```
# (a) Creating the folder for the project (and getting in there)  
mkdir ~/JSC370  
cd ~/JSC370  
  
# (b) Initializing git, creating a file, and adding the file  
git init  
  
# (c) Creating the Readme file  
echo An empty line > README.md  
  
# (d) Adding the file to the tree  
git add README.md  
git status  
  
# (e) Committing and checkout out the history
```

```
git commit -m "My first commit ever!"  
git log
```

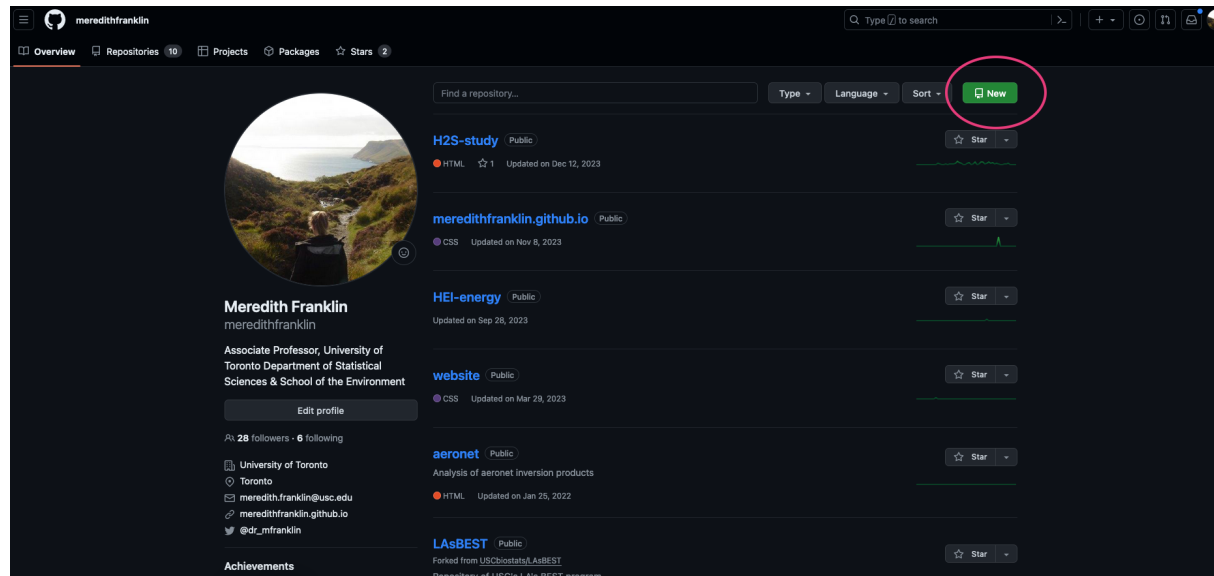
Hands-on 1: Remote repository

If you add a wrong file to the tree, you can remove files from the tree using `git rm --cached`, for example, imagine that you added the file `class-notes.docx` (which you are not supposed to track), then you can remove it using

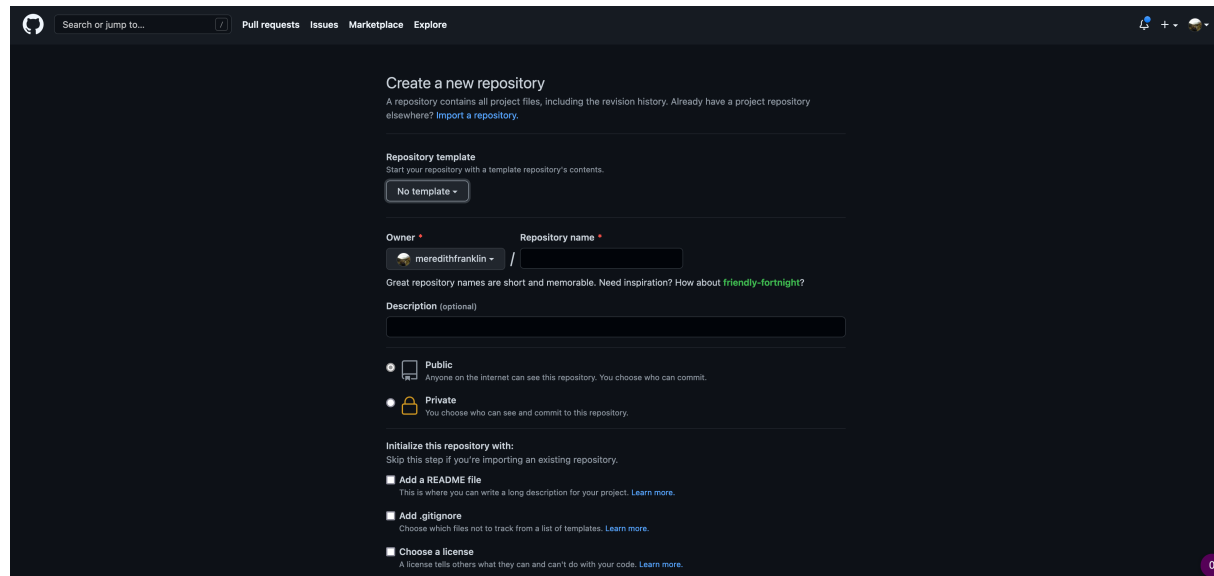
```
$ git rm --cached class-notes.docx
```

This will remove the file from the tree but not from your computer. You can go further and ask git to avoid adding docx files using the [.gitignore file](#)

Hands-on 1: Remote repository



Hands-on 1: Remote repository



The screenshot shows the GitHub interface for creating a new repository. At the top, there is a search bar and navigation links for Pull requests, Issues, Marketplace, and Explore. The main heading is "Create a new repository", followed by a sub-heading: "A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)"

Under "Repository template", there is a button labeled "No template".

The "Owner" is set to "mereditfranklin" and the "Repository name" field is empty. Below this, a tip states: "Great repository names are short and memorable. Need inspiration? How about [friendly-fortnight?](#)"

The "Description (optional)" field is also empty.

There are two radio button options for visibility: "Public" (selected) and "Private".

Under "Initialize this repository with:", there are three checkboxes: "Add a README file", "Add .gitignore", and "Choose a license".

A small purple circle with the number "0" is visible in the bottom right corner of the page.

Example for .gitignore

Example extracted directly from Pro-Git ([link](#)).

```
# ignore all .a files
*.a

# but do track lib.a, even though you're ignoring .a files above
!lib.a

# only ignore the TODO file in the current directory, not subdir/TODO
/TODO

# ignore all files in any directory named build
build/

# ignore doc/notes.txt, but not doc/server/arch.txt
doc/*.txt

# ignore all .pdf files in the doc/ directory and any of its subdirectories
doc/**/*.pdf
```


Resources

- Git's everyday commands, type `man giteveryday` in your terminal/command line. and the very nice [cheatsheet](#).
- My personal choice for nightstand book: The Pro-git book (free online) [\(link\)](#)
- Github's website of resources [\(link\)](#)
- The "Happy Git with R" book [\(link\)](#)
- Roger Peng's Mastering Software Development Book Section 3.9 Version control and Github [\(link\)](#)

- Git exercises by Wojciech Frącz and Jacek Dajda ([link](#))

- Checkout GitHub's Training YouTube Channel ([link](#))