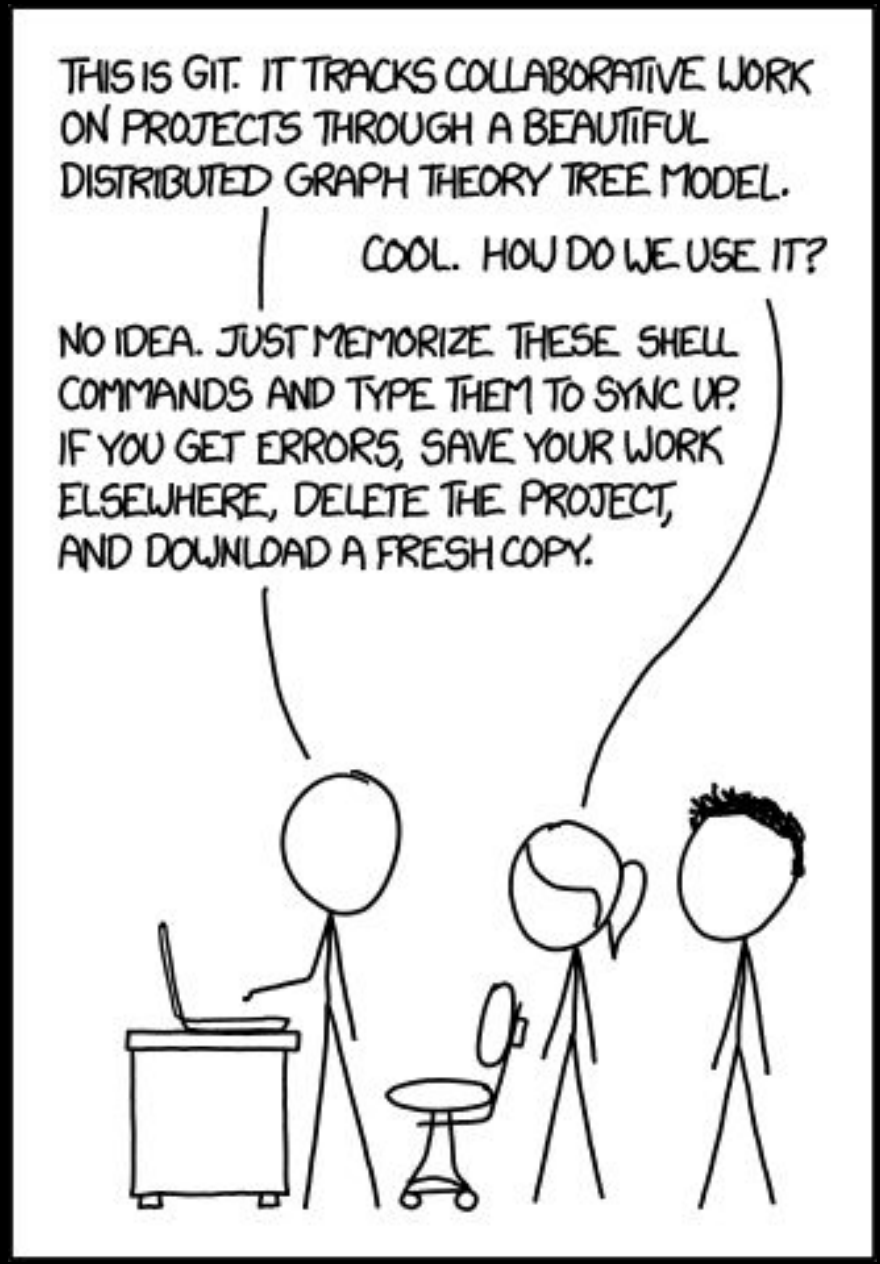
INTRO TO GITHUB

Brought to you by your NYU Stats Club

DISCLAIMER:
We're not GitHub experts…
Some days, this is just how it goes. But the sooner you start using GitHub consistently, the easier it will be!



SOME (UNSOLICITED) ADVICE

- Be consistent with file organization & naming!
- Use relative file paths
  - i.e. try to avoid things like (Users/Shannon/Docs/A3SR/stats_club/github_wksp.pptx)
  - Set your working directory wisely can reference the same file with (stats_club/github_wksp.pptx)
  - Looks nicer AND makes it much easier to share work!

GETTING SET UP

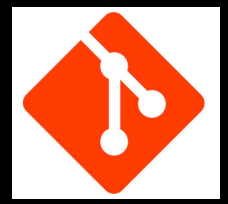
- Students get GitHub pro for free! Make an account here: https://education.github.com/discount_requests/new
- Download git
  - Mac– install Xcode from the app store
  - Windows– https://git-scm.com/download/win

GETTING SET UP(CONT.)

- Setting up the your Git username and email
  - Credentials by which you code will be saved in git on the machine as well as on GitHub
  - **Global config**– applies to ALL your repositories
  - Setting up username, email
    - git config --global user.name “Shannon Kay”
    - git config --global user.email “smk880@nyu.edu”

GIT != GITHUB

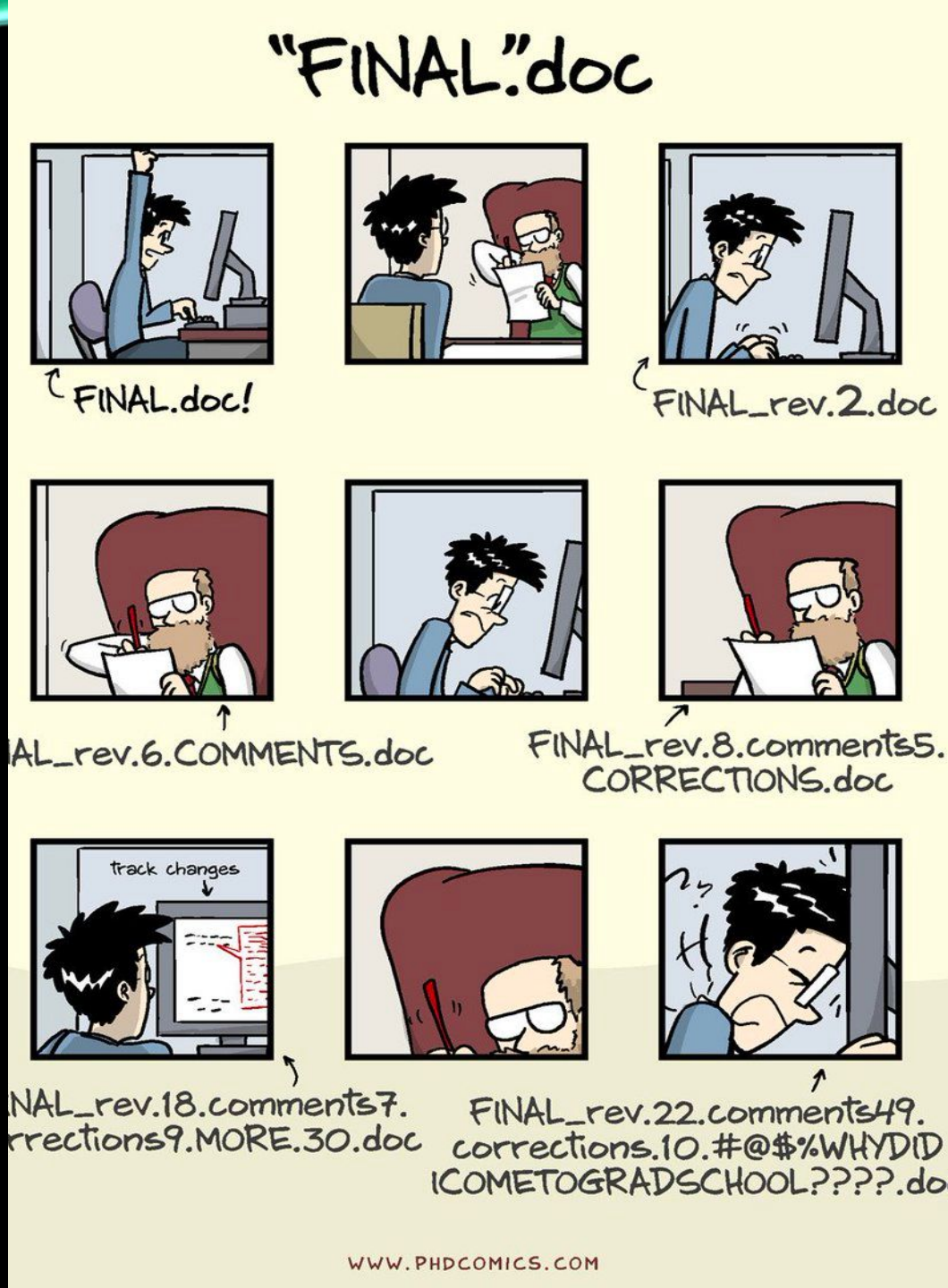
- git
  - A command line tool for managing files
  - Version control system
  - **repo** = repository = a folder/directory managed by git
  - **Commit** = a snapshot of a repo/creating a snapshot of the repo



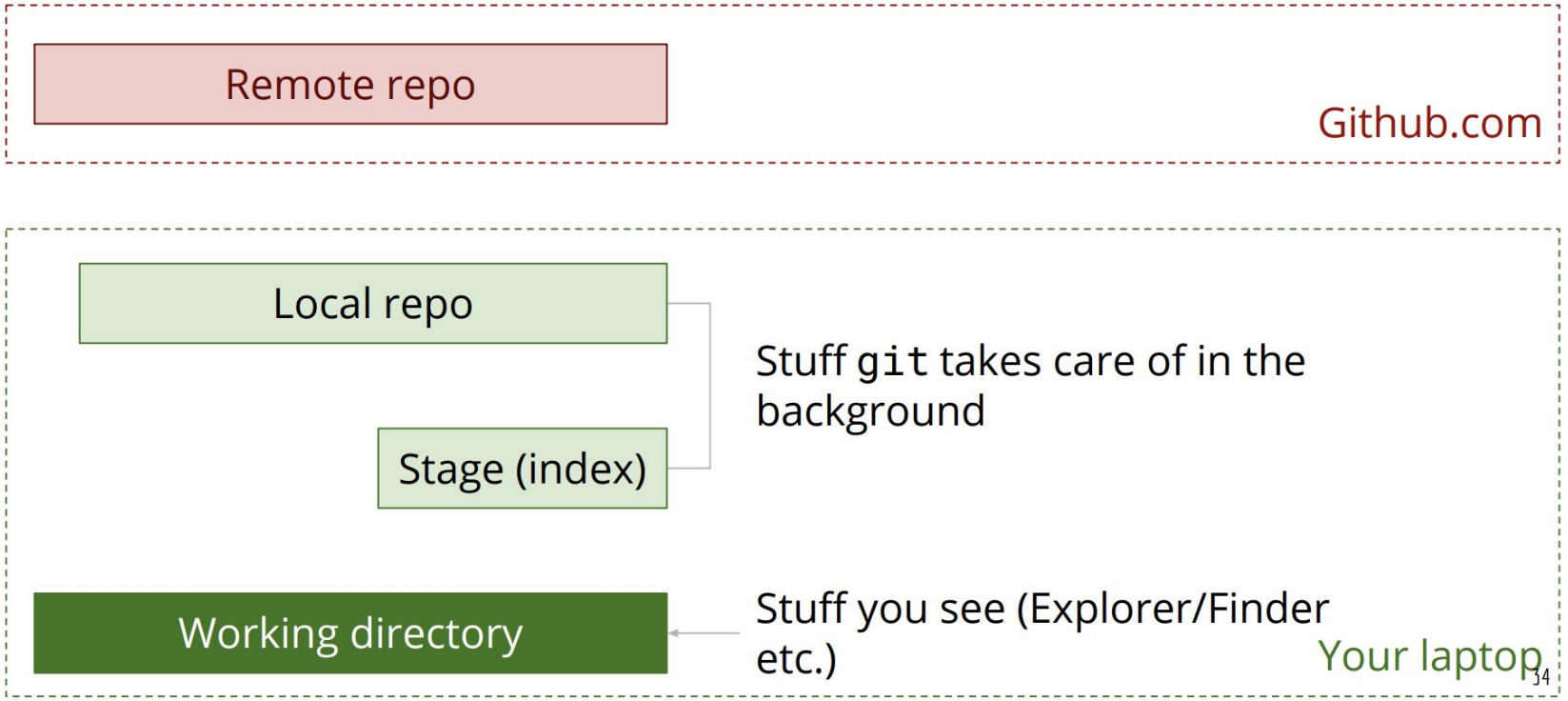
- GitHub
  - Web-hosting service that lets you collaborate & share your git-stuff, something like Dropbox (but not quite …)
  - You **push** changes from your local files to GitHub via git and **pull changes** from GitHub to your device
  - Join organizations, control access
  - Save & publish code online

WHY DO WE NEED IT??

- Helps avoid saving files like Code_v1.R, Code_v2.R, Code_final.R, Code_final_final.R, etc.
- Ability to travel back in time!
  - Access code written the past
  - Revert back changes
- Easy way to publish your work online
- To find a job
  - More and more recruiters want to see your code
  - A lot of applications require a link to your GitHub



So, how does `git` work?



BASIC COMMANDS

- **Remote add origin:**

  git remote add origin “https://github.com/USERNAME/REPOSITORY.git”
- **Clone (download) a repo from GitHub (do this in the directory you want to clone to)**

  git clone “https://github.com/dfarnand/wordcloud.git”
- **Status: check overall status of changes made:**

  git status
- **Pull: get the latest updates from the online repo:**

  git pull

BASIC COMMANDS (CONT.)

- **Add: Add files to commit**
  - git add filename (to add files individually)
  - git add . (to add all files with changes)
- **Commit: Stage files before pushing:**
  - git commit -m “a message about the changes”
    - Make the message meaningful as it could come in handy in the future!
- **Push: Update the online repo with your changes:**
  - git push
- **Log: see your recent commits:**
  - git log


COMMENT
DATE
CREATED MAIN LOOP & TIMING CONTROL
14 HOURS AGO
ENABLED CONFIG FILE PARSING
9 HOURS AGO
MISC BUGFIXES
5 HOURS AGO
CODE ADDITIONS/EDITS
4 HOURS AGO
MORE CODE
4 HOURS AGO
HERE HAVE CODE
4 HOURS AGO
AAAAAAAA
3 HOURS AGO
ADKFJSLKDFJSDKLFJ
3 HOURS AGO
MY HANDS ARE TYPING WORDS
2 HOURS AGO
HAAAAAAAAANDS
2 HOURS AGO
AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

YOUR TYPICAL USE

- 98% of the time, you will use git as follows [assuming the repo is set up, and is on github]:
  1. git pull
  2. do your work (edit your files, etc)
  3. git add the changed files
  4. git commit [or, just git commit -am “commit message”]
  5. git push
- Use git status in between, to keep track of what’s going on
- Use Google frequently...

HYPOTHETICALLY

- You can use GitHub to time travel!
- Remember how we referred to commits as snapshots of your work?
  - After a commit is made, you can go back and forth to check the state of your project. Maybe you were experimenting with some new function and realized the old one was better. No problem, you can bring back anything!
- Only works if you commit consistently during your work.
- Would use git reset
  - But we're not teaching time travel today

CONFLICTS AND RESOLUTIONS

- **Markers of conflicts:** <<<<<<<, =======, >>>>>>>
- If you see this in your code, remove the markers and retain what is needed.
    - Save, add, commit and push changes normally after that

WHAT'S .GITIGNORE?

- By default, git saves *everything* inside the folder where you initiated the repo.
- To prevent git from saving files, create a file and name it .gitignore in the folder where you ran git init.
  - In such file, you can add rules to let git know what you want it to ignore. All you need to do is specify the file name or file extension type.
  - Tells Git what files/types of files not to monitor changes in
- Some examples:
  - **Packages:** *.7z, *.dmg, *.gz, *.iso, *.jar, *.rar, *.tar,
  - ***.zip** (good to unzip these and push the content)
  - **OS generated files:** *.DS_Store
  - **R:** *.Rproj.user, *.Rhistory, *.Ruserdata, *.Rdata
  - **Python:** *.pyo, *.pyc
  - **Backup files**: *~

WHAT **NOT** TO PUT IN YOUR REPO

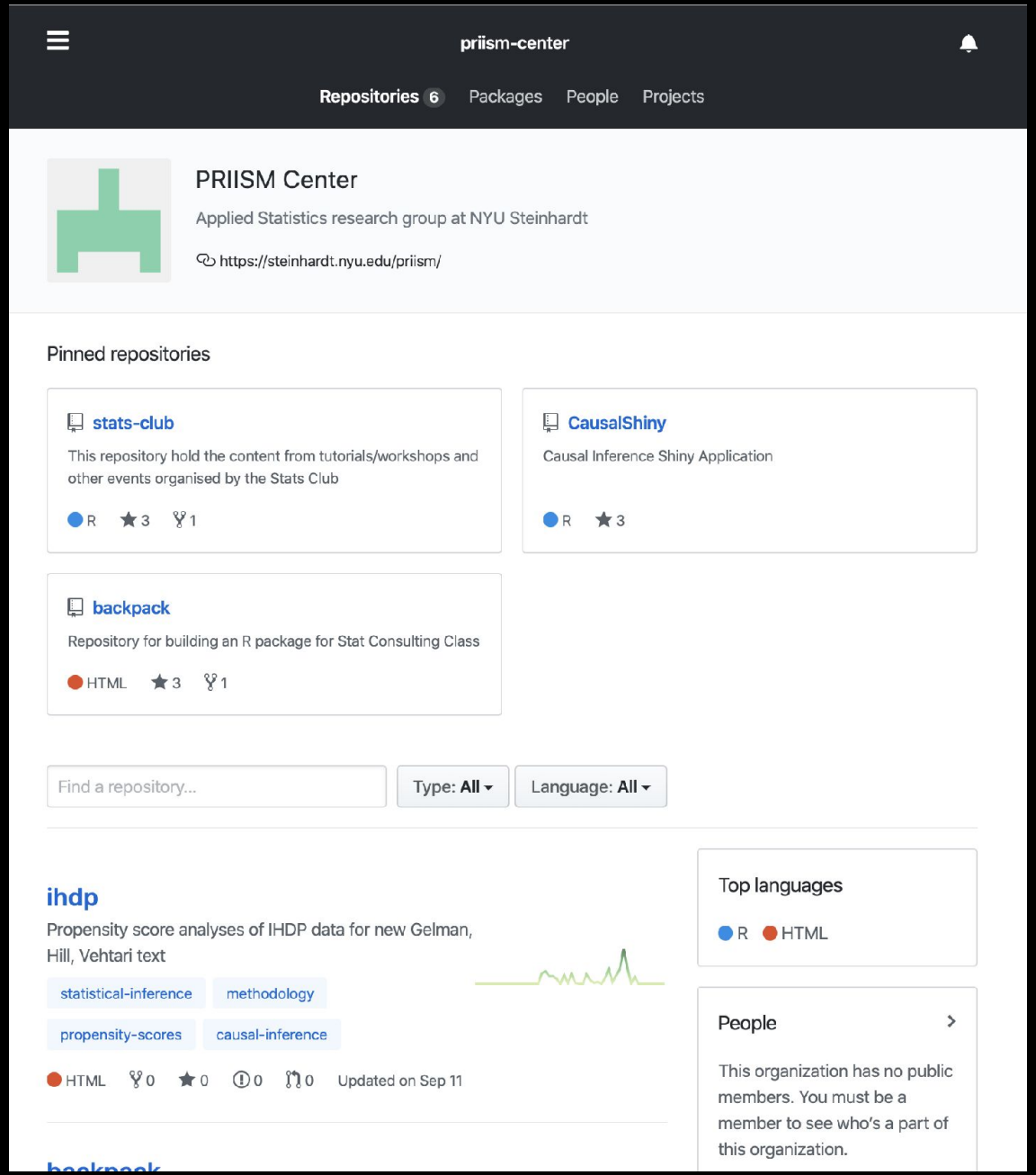
- Some files should not be kept in a git repo:
  - Large output files that are deterministically generated from other files
  - log files (e.g., .RData, .Rhistory)
  - Local settings (e.g., .Rproj.user)
- Certain files should **NEVER EVER EVER** be in a git repo
  - Sensitive data (e.g., human subject data, SSN)
  - Credentials! (e.g., AWS access keys, API tokens/secrets)

KEY POINTS TO REMEMBER

- Pull updates before you start!
- add -> commit -> pull -> push
- Remember to commit changes
- Add meaningful messages with commits
- Make use of projects in Rstudio
- Best practices
  - Create a README file that documents your repository structure
    - What are the different folders? eg: data, src, results, etc.
    - What does each folder have?
  - Check/skim through differences before commit
  - Try to follow a good project template structure as much as possible

BY THE WAY…

- PRIISM-center has a GitHub!
  - Private repos for student projects and student-A3SR faculty collaborations
  - Data and code can be kept private to the team
- Easy to publish student work in the future
- Stats Club has a repo!



RESOURCES

- **Resources for beginners**
- 15 minute tutorial to learn git - This is a must for people to get started.
- git - the simple guide - A simple guide to get to know the most important concepts.
- Pro Git - Free online textbook.
- **Resouces for becoming a git ninja**
- A successful git branching model - A model to work with git using branches. This model is widely used in the open source community.
- Learn Git Branching - Understanding what branches and rebases are, in an amazing interactive tutorial.
- Reset Demystified - A blog post on git reset which develops some useful concepts along the way.
- Understanding git for real by exploring the .git directory - A blog post on what's inside a commit.
- A git style guide, complete with branch naming, suggestions on how to handle commit messages, and more.

THANK YOU!!


I properly test my
changes and ensure they
work before I commit and push
And
Other Hilarious Jokes
You Can Tell Yourself
Volume II

SOURCES

- MDML Lecture 3- Professor Ravi Shroff
- Last year's stats club workshop (credit to Clare Clingain & Kaushik Mohan)

https://github.com/priism-center/stats-club/tree/master/GitHub-Basics