STAT 209
Troubleshooting Workshop!

March 16, 2018

Colin Reimer Dawson

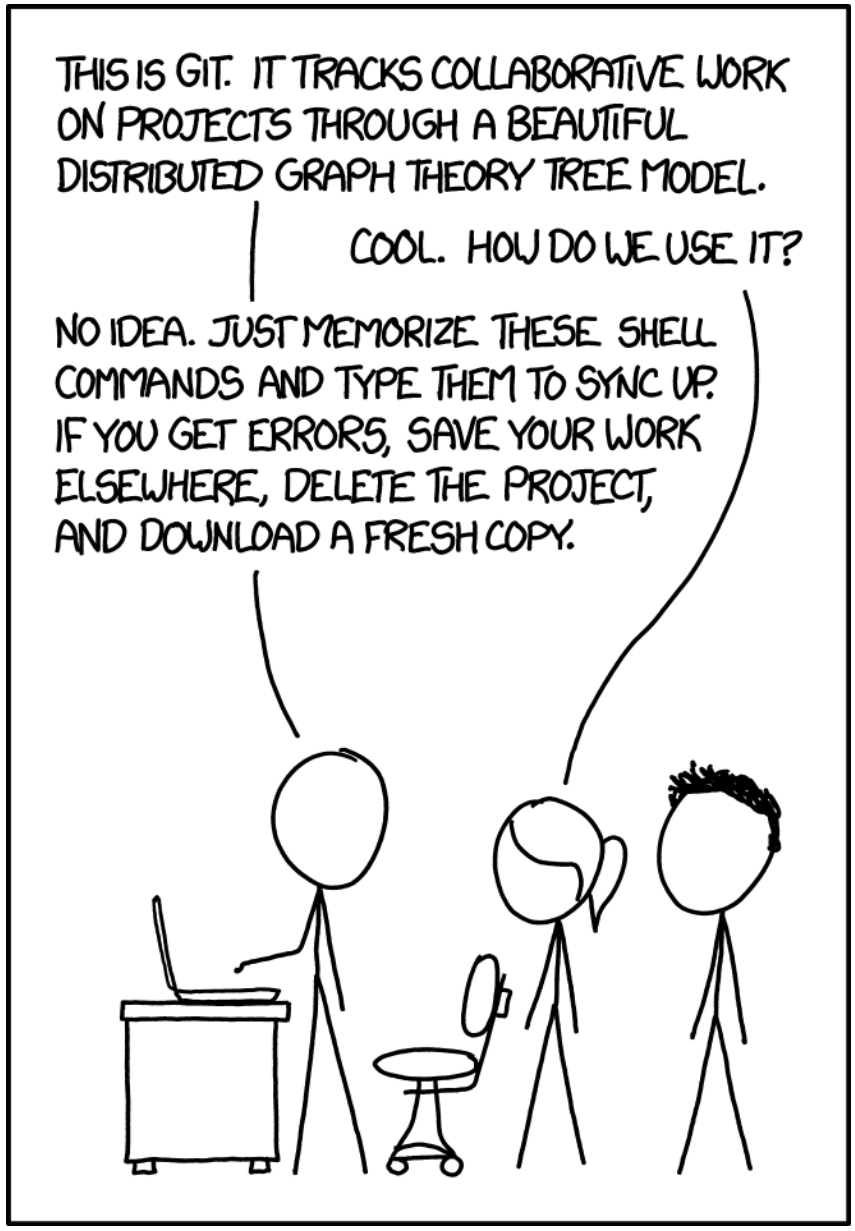
My tentative plan for today (but feel free to suggest other things)

- Making `git` and GitHub less painful
- Responding to your `#lab5 and #lab7` questions
- If there's time left, working on outstanding labs

Outline

GitHub Pain and Suffering

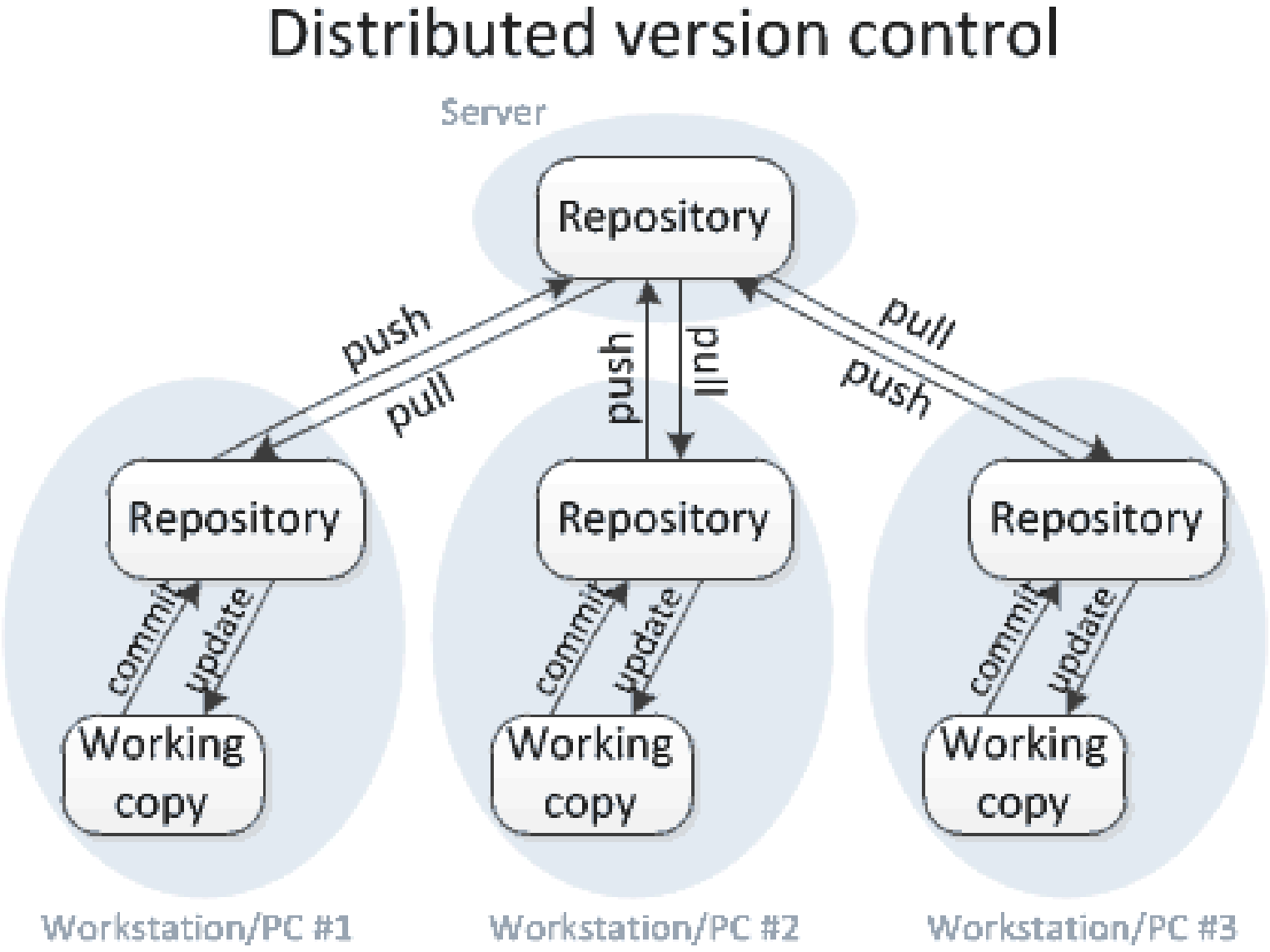
Data-Wrangling Issues


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.

Reminder: The Main Git Verbs

`clone`	Copy a remote repo to your computer
`pull`	"Download" changes from the remote repo to your local repo
`add`	Register changes to some files to be committed at the next commit
`commit`	Take a snapshot of your working directory and register the state in your local repo
`push`	"Upload" new commits from your local repo to the remote repo

Reminder: Git Structure



Recommended Workflow

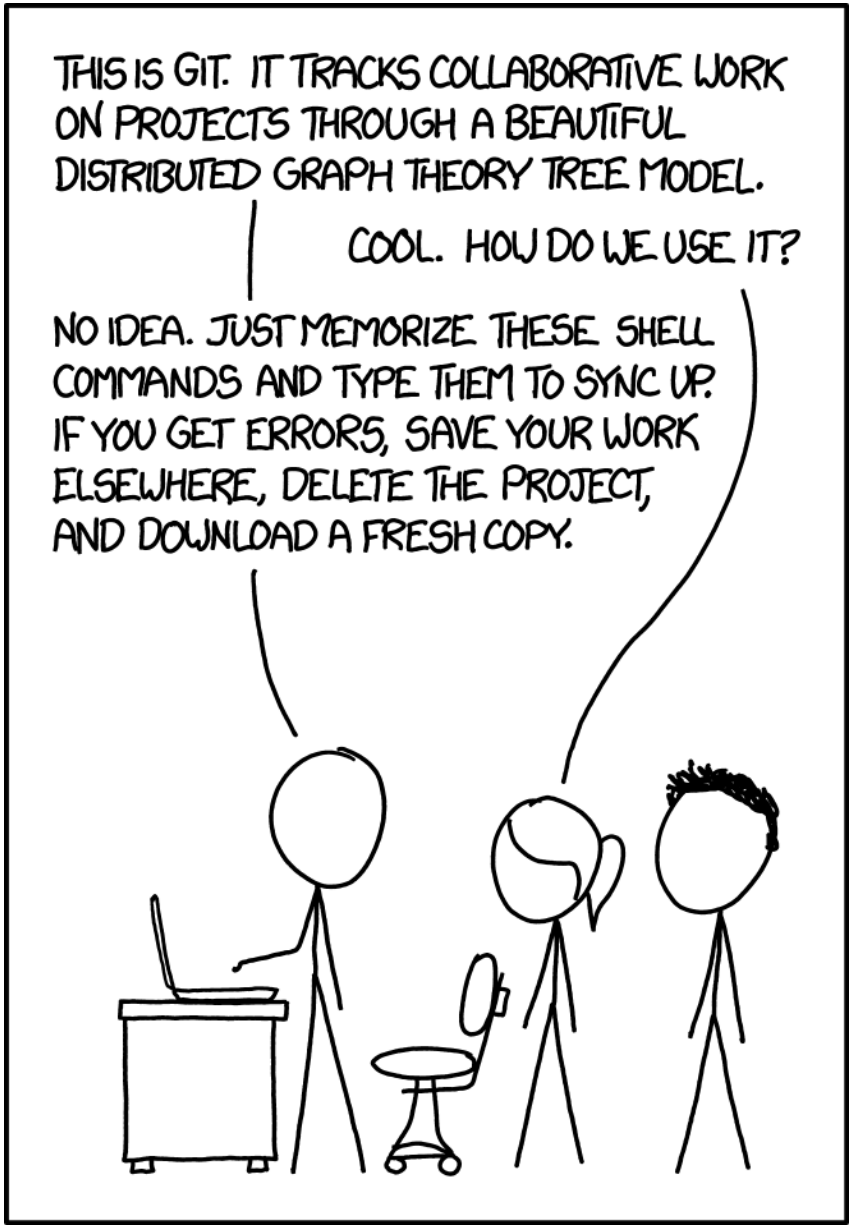
Each time you sit down to work on a project:

1. First **pull**, in case others have made changes
2. Assuming no conflicts, begin **edit**ing
3. Each time you do a "unit of work", make a **commit**, with *an informative commit message* that describes what you did
4. Now **pull** again, in case others have made changes
5. If not done, go back to step 2 (unless you need someone else to get your changes immediately, in which case **push**, then return)
6. **push** your commits

Merging Changes

- Sometimes you will get error messages that prevent you from **push**ing.
- Sometimes the problem is solved by simply following the directions that `git` gives you (e.g., `pull` first, then try `commiting` and `pushing` again
- Worst case, you can revert to whatever state the project was in at the last successful `push`

Panic Button



Panic Button

- A fallback option that's always available is to rename your project directory, and `clone` the repo from GitHub afresh
  - In RStudio, this means creating a new project from the same URL
- Then you can copy your newer versions of files from your renamed directory, `pull` again (to be safe), `commit`, and `push`

A Better Solution

- Before pressing the panic button, try to resolve the conflict
- **Don't resort to committing via the GitHub web interface!**
- Demo
- **Caution: Make sure to save everything and close your project in RStudio when done for the session. Don't accidentally save other work in the project directory**

Outline

GitHub Pain and Suffering

Data-Wrangling Issues

I still have trouble discerning what types of arguments are accepted by each function.

To some extent, this just takes practice, but key types of arguments are

1. Datasets (all, but often "piped" in, except `_joins()` which take a second one)
2. Existing variable names (`select()`, `group_by()`, `gather()`, `spread()`)
3. Logical conditions (`filter()`)
4. New variables to create as arg names, and definitions for those variables as arg values (`mutate()`, `summarize()`)
5. New variables to create as arg values (`gather()`)

> One thing that still remains unclear to me is knowing when to use the different 'join' functions (i.e using `inner_join` as opposed to `left/right_join` or `semi_join`, etc.)

Only difference between `inner_`, `left_/right_`, and `full_join` is which cases get kept

- Is data only useful if I have it from both tables? Then use `inner_`
- Are the cases I care about exactly those in one of the two datasets? Then use `left_` or `right_`
- Am I interested in cases with any data from either table? Then use `full_`

It is still a bit unclear when to use the verbs inside the main verbs, such as `mean`, `max`, and `min`, as sometimes the variable that you are changing does not always work with what you want

(Maybe) two different issues here:

1. Making sure variables are read in in the right format (may want `parse_number()` or `parse_integer()`, or `parse_date()`, used inside `mutate()`)
2. Functions that *transform* (use with `mutate()`) vs functions that *aggregate* (use with `summarize()`)

> I think the points I'm still a bit unsure of is distinguishing when to use `select()` and when to use `filter()`

The key difference:

1. `filter()` picks out *cases* / *data points* (rows)
2. `select()` picks out *variables* (columns)

I'm a little unclear as to what `summarise(N=n())` does

Yes, there are too many `n`s and `N`s floating around.

1. `n()` is a special function to use with summarize that just counts the number of rows we are summarizing over
2. `n` is also a variable name in `babynames` counting number of births
3. `N` is the name we are giving to the new column holding the number of rows computed by `n()`

While I understand how to manipulate data in R studio from the labs and the datacamp videos, I think its much harder to apply what I learned to actual data.

Absolutely. There's a hierarchy of depth of learning new things:

1. Reading/seeing an explanation
2. Guided/structured practice
3. Free-form practice
4. Explaining how to do it to someone else