A LaTeX-oriented intro to Git

the tex part is in the interactive demo not in the slides

Danielle Amethyst Brake

22 October - 26 November, 2018

ICERM Semester on Nonlinear Algebra

31 October - 1 November, 2019 with Martin Skrodzki

ICERM Semester on Illustrating Mathematics

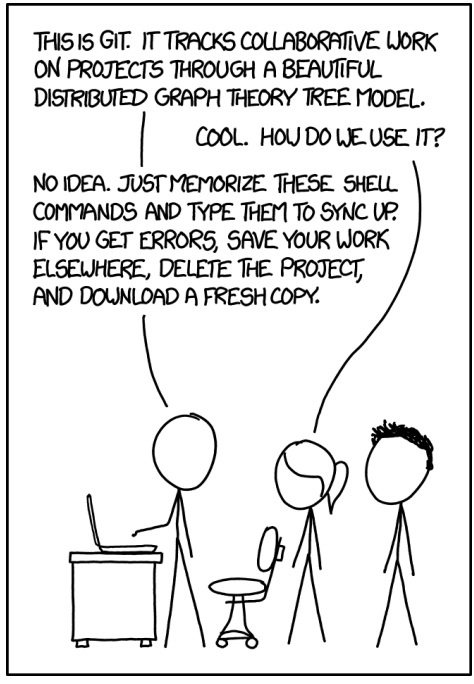
University of Wisconsin
Eau Claire

The Power of AND

Outcomes from these four sessions

- Make a repo, commit to it
- Basic skills for working with a Github hosted repo
- Pushing, pulling
- Branching, mergine
- Forking, issues, pull requests

This is *a lot*.


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.

xkcd 1597

Outline

What is git?

Definition

`git` is a command-line program that records differences in files.

and how is that different from github?

good question, my friend.

Definition

GitHub is social media site build around `git` repositories.

What are words related to Git?

Repo-focused words

- version control software
- repository / repo
- commit
- clone
- remote
- reset
- `.gitignore`

Commit-focused words

- branch
- rebase
- fast-forward
- merge
- fetch
- pull
- conflict

What's a commit?

Definition

A **commit** is a set of differences – additions and subtractions of lines in file – together with a message describing them

What's a repo?

Definition

A **repository** is a folder, together with a subfolder containing special git files.

Don't screw with these special files. They live in `.git/` inside the repo folder.

Ok, sometimes you need to adjust them...

What should I version control?

Use git for these:

- Files that can be meaningfully-diffed (think text files), and
- static non-diffable files (think images that don't change)

do NOT use git for version control for these:

- generated files (pdf's, executables)
- temporary files
- word documents (but you wouldn't use word anyway, would you?)

How do i make a repository?

1. make a new folder, or move to an existing one
2. `git init`

look around in the folder. does it look different?

How do i commit?

1. make some changes
2. `git add filename.ext`
3. `git commit`
4. write a commit message
5. save ‘file’ for commit message, done

What's a good commit message?

1. a short summary, like 3-10 words
2. some longer description

- no vague crappy messages that utterly fail to describe your changes
- a message to your future self

How do I prevent myself from adding temp files?

Be very careful to not ever add a temp file, because once it starts tracking, it's there forever.

Use `.gitignore`, a file at root level (in the repo), that describes patterns for files to ignore.

protip: `.gitignore` is cumulative in subdirectories, and you can un-ignore things too

How do I delete files?

1. Delete the file in your local clone
2. `git add deletedfile.ext`
3. `git commit`

or just

1. `git delete file` – all in one

What if I mess up?

Don't mess up. JK, but seriously, don't mess up.

You cannot delete commits from a repo.

- You **revert** them with the inverse commit.
- You can delete branches... and maybe lose work
- You can delete repos... and maybe lose work

Things I haven't talked about yet

- **Merge conflicts** – helping solve the "hot copy" problem
- GitHub
- Alternatives to GitHub
- Using Git as a time machine
- Alternatives to Git
- Branching models
- Rebasing, pulling, fast-forwarding

Practice time!

Things to practice:

1. Make some changes
2. Commit your changes (`git commit`)

Things to research:

- Deleting files (`git rm`)
- Renaming files (`git mv`)
- Writing your commit message in the same command as committing (`git commit -m "arst"`)

Outline

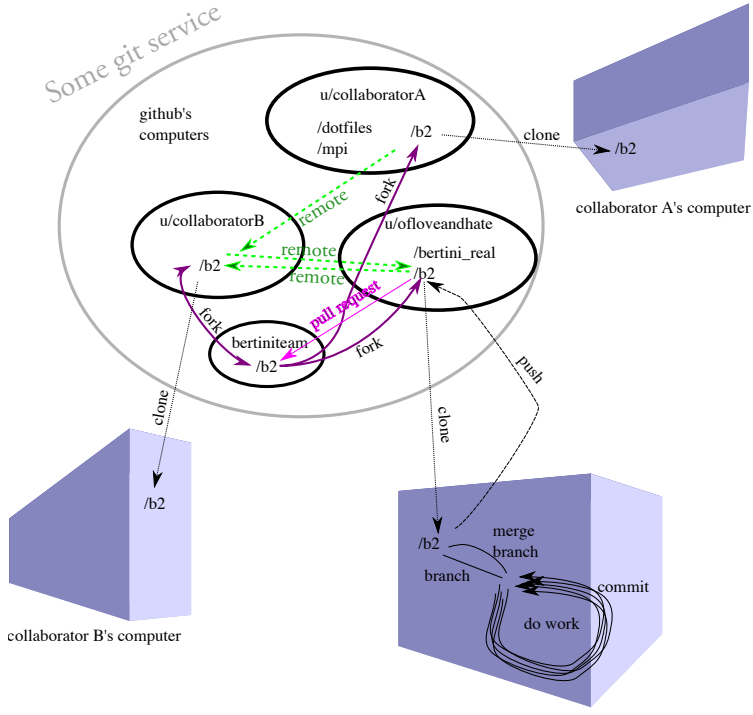
GitHub-specific things

Repo-focused words

- fork
- clone
- issue

Commit- or branch-focused words

- pull request


Some git service
github's
computers
u/collaboratorA
/dotfiles
/mpi
/b2
clone
/b2
collaborator A's computer
fork
remote
u/collaboratorB
/b2
remote
remote
u/ofloveandhate
/bertini_real
/b2
fork
pull request
bertiniteam
/b2
fork
push
clone
clone
/b2
collaborator B's computer
/b2
merge
branch
branch
commit
do work

Game plan for teaching

We'll do these things:

1. make a new repo on github
2. clone
3. commit
4. push

and give overviews for these:

- remotes
- pulling and pushin
- merges and conflicts

1. new

Let's make a git repository on Github.

⇒ Follow along on the projector

If you pay GitHub, you can make your repos private

→ Yes, I pay for that service
⇒ (no, i don't, they have free4academic going)

we won't make a repo on our local computer this time

What is a clone?

Definition

A **clone** is a repo that has been "cloned" from another with `git clone`. The source is a *remote*

- It's a copy of the original repo, in terms of commits.
- It's only an identical copy when cloned. it's expected to diverge by committing...

2. clone

on your local machine, from the terminal

1. move to a location you want the repo's folder to be
   ```
   cd /path/to/folder
   ```
2. type
   ```
   git clone https://github.com/username/reponame
   ```
3. look around
   ```
   ls
   ```

3. commit

while 1

1. make some changes
2. **add** the changed files
3. **commit**

4. get that data online

now we'll push.

Definition

to **push** is a git action, meaning to make a sequence of *commits* appear in another repo

- all existing commits in `remote` must be present to push.
- hence, pull before you push. this often makes no difference in a one-person project
- typically, you push to origin, which is where the repo came from when cloned

Why use a git server? Collaboration

there are many ways to use git to share code (focusing on github here).

here are 2 main ways:

- add all members as collaborators, all people commit directly to repo
- each person forks, and resolve via pull requests

they both have advantages and disadvantages. We get to choose.

- Collaborating will lead to merge conflicts sooner
- Forking will let us learn about forking and PR's. This will happen in session 4.

pull, fetch, merge – defn

Definition

to **pull** is a git action, meaning to fetch-and-merge from a remote

Definition

to **fetch** is a git action, to query available branches and commits

Definition

to **merge** is a git action, meaning to make a sequence of commits appear in the current repo. you get the commits from a branch, either *locally* or from a *remote*

some speech using the diagram is necessary here

merge conflict

git tracks diffs on diffable files. it's pretty smart about disjoint modifications to a file, but sometimes... it can't tell what you want to do.

- a merge conflict isn't bad
- the conflict is resolved when the conflicted file is marked as saved.
- i honestly use a visual tool for this and don't know how to do it command line.
- i totally use visual diff tools.

Outline

What if I have multiple things to work on at once?

Branches solve the problem of working on multiple facets of a project. They're separate commit-sequences that you can make converge by `merging`

- Make a new branch via `git branch newbranchname`
- Convege a branch into your current by `git merge branchtomerge`
- Get a branch from a remote via `git checkout existingbranchname`
- Update your branch list and commit status with `git fetch [remotename]`

relevant branch words

- `branch`
- `checkout`
- `merge`
- `fetch`
- `pull`
- `stash`

make a branch

do it now. either in a GUI or in the terminal

```
git branch branchname
```

you can totally make up "folders" for your branches, and branch from branches, etc.

make a commit

do it now. either in a GUI or in the terminal

make a switch back to `master` branch

do it now. either in a GUI or in the terminal

now look around at the file you edited. what do you notice?

branch again, from master

```
git branch branchname2
```

make a commit

do it now.

please, DO modify the same area of a file you edited in the other branch-commit exercise. (i have a reason you should do this)

now we'll merge in the first branch

1. switch back to `master`
2. `git merge branchname`

optional – `git branch -d branchname`

this is done at the end of a branch's life, often, when you are done with a feature (sequence of commits focused on one thing)

merge in the second branch

same process. be in branch you want the commits to end up in, and merge the other branch into it.

1. switch back to `master`
2. `git merge branchname2`

optional – `git branch -d branchname2`

if we've done it right, we'll get a merge conflict problem.

What's a merge conflict?

git records differences in files.
if a patch (diffset) is attempted to be applied on top of a set of commits that record changes in the same portions of a file, it's a **merge conflict**.

Get some tools to help

I turn to GUI tools at this point, honestly.
There are lots of tools you can use. Choose one for

- git itself. i do not recommend the GitHub app, unless you are hosting origin at GitHub.
- merging. do a search right now for tools for this. OSX users – XCode comes with FileMerge, and it's not terrible.

How to resolve a merge conflict

1. goto the repo in the git tool, and use the UI to 'launch external merge tool' on the conflicted files.
2. use the merge tool to resolve conflicts, choosing right, left, both, neither, or hand-editing.
3. save
4. test. goto 2?
5. quit the merge tool
6. return to the git tool. finish the commit to finish the merge.

things you should know

- If things get FUBAR, `git reset --hard` will, well, reset your repo to before you started the merge. just start over
- beware accentally committing partially merged files. you get diff crap left in the file, and it's annoying
- You might consider doing the merge from master into feature, then merge feature into master, to make any mistakes appear in feature.

Outline

Github features

- Forking
- Pull requests
- Issues

Fork

Definition

to **fork** is a Github action, meaning to take a clone from one user's repository, and associate it as upstream

Definition

we say A is **upstream** of B when A is a remote of B, and Github has stored that direction in the association graph

let's use that diagram again

Pull request

Definition

a **pull request** is a Github thing that requests that an owner of a repo A from which a fork B has been taken, pull from B into A.

Pull requests (PRs) facilitate discussion around blobs of commits, and the resolution of the merge.
They're helpful because, if you don't have this service, how do you let the other person know that you're ready to share your new changes / code?

notes on PRs

- Pull requests are often made from branches on the fork. In fact, when you make the PR, you choose the from/to branch.
- Sometimes it is nice to generate PRs, even if you control both repos, for generating records.

let's make a PR.

together, let's

1. fork a friend's repo
2. clone to our computer
3. branch
4. commit to that branch
5. push to origin
6. make the PR
7. resolve the PR

Issues

Definition

an **issue** is a github thread of conversation focused on one topic related to the repo.

notes on issues

- Issues are unique to the repo. Forks don't carry issues.
- Issues and PR's form a sequence starting at 1. #1 is the first issue/PR for that repo
- Use **labels** to good effect. Github pre-populates a list for you these days
- Hide irrelevant posts in the thread to help your users
- Think about having a policy on the README or on your Wiki that describes the etiquette expected in your issues page

Let's make issues

1. Find a friend's repo
2. Start an issue

This part will be done in a browser

What else would you like to talk about?

???

Thank you!