Becoming a Git Wizard

Pearl Hacks

D. Ben Knoble

UNC Chapel Hill

23 February 2019

Part I

Getting Started

Ground Rules

Introductions

Setup

Ground Rules

- ▶ Introduce yourself with name and pronouns (this will be your partner)
- ▶ Respect others' pronouns
- ▶ There are no dumb questions
- ▶ Help your neighbor when you can
- ▶ Have a good time!

About Me

- He/Him/His
- @benknoble on GitHub

My Journey with Git

▶ Pre-college: what is version control, and who cares?

My Journey with Git

- ▶ Pre-college: what is version control, and who cares?
- ▶ Blogging! GitHub?

My Journey with Git

- ▶ Pre-college: what is version control, and who cares?
- ▶ Blogging! GitHub?
- ▶ Interlude: command-line

My Journey with Git

- ▶ Pre-college: what is version control, and who cares?
- ▶ Blogging! GitHub?
- ▶ Interlude: command-line
- ▶ Hackathon! Merge conflicts?

My Journey with Git

- ▶ Pre-college: what is version control, and who cares?
- ▶ Blogging! GitHub?
- ▶ Interlude: command-line
- ▶ Hackathon! Merge conflicts?
- ▶ Class projects!

My Journey with Git

- ▶ Pre-college: what is version control, and who cares?
- ▶ Blogging! GitHub?
- ▶ Interlude: command-line
- ▶ Hackathon! Merge conflicts?
- ▶ Class projects!
- ▶ Now: wizardry (internships?)

My Journey with Git

- ▶ Pre-college: what is version control, and who cares?
- ▶ Blogging! GitHub?
- ▶ Interlude: command-line
- ▶ Hackathon! Merge conflicts?
- ▶ Class projects!
- ▶ Now: wizardry (internships?)
- ▶ Now is the *best* time in your career for this

What We Will *Not* Be Doing

Figure: xkcd/1597

- ▶ Memorizing magic commands
- ▶ Learning the (beautiful) model behind git

Required setup

1. `git`
2. GitHub account
3. GitHub Desktop [be sure to sign in with GitHub!]
4. Fork and clone the starter code

Do what?

fork Deep-copy a repository/project from another GitHub
account to your GitHub account

clone Deep-copy a repository/project from one location to
another (e.g., from GitHub to your computer)

Other notes

- ▶ GitHub Desktop is a git *client*—other programs exist that allow you to use git, too
- ▶ GitHub is not git; it hosts projects that use git

Part II

The Magic of git

First Steps with `git`

Experimentation with the Multiverse

Collaboration in a Distributed System

Advanced: Dealing with Merge Conflicts

We Don't Want To Do This
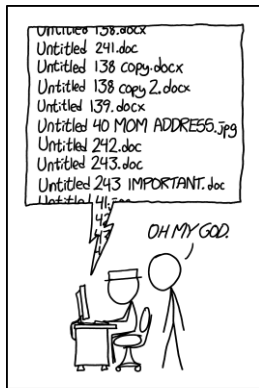
PROTIP: NEVER LOOK IN SOMEONE ELSE'S DOCUMENTS FOLDER.

Figure: xkcd/1459

- ▶ Because we are software engineers and artists
- ▶ Because there is a better way

Instead

We commit.

Definition (Commit)

A snapshot in time of a project's *content* that knows what came before it.

How?

Definition (Commit)

A snapshot in time of a project's *content* that knows what came before it.

1. Make changes
2. Add files
3. Commit
4. (Push to update a remote)

Lumos: Your First ~~Spell~~ Commit

Not the time for *commitment* problems

Exercise (Make a commit)

1. Create yourname.txt in your editor or file-browser
2. Place your name in it
3. Click commit
4. Push to your fork

Lumos: Your First ~~Spell~~ Commit
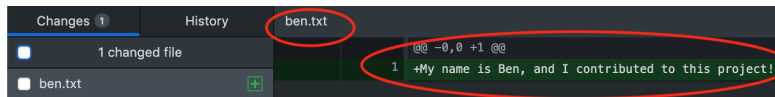
Figure: Create `ben.txt` (1: make changes)

Lumos: Your First ~~Spell~~ Commit
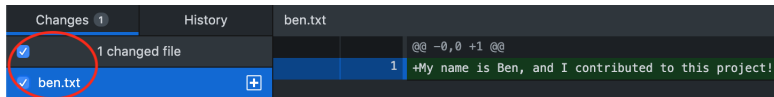
Figure: 2: Add `ben.txt`

Lumos: Your First ~~Spell~~ Commit
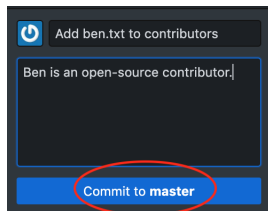
Figure: 3: Commit

Lumos: Your First ~~Spell~~ Commit
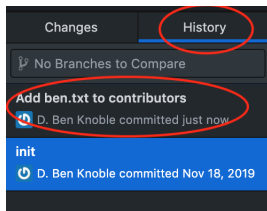
Figure: Checking out our history

4: Now we're going to push

A Note about Messages

AS A PROJECT DRAGS ON, MY GIT COMMIT
MESSAGES GET LESS AND LESS INFORMATIVE.

Figure: xkcd/1296

Our messages should summarize
the what and describe the why.
Here's a good format to follow
(click to follow link).

Trying new things

How many of you have ever. . .

- ▶ edited code?

Trying new things

How many of you have ever. . .

▶ edited code?

▶ tried to undo your edits?

Trying new things

How many of you have ever...

- ▶ edited code?
- ▶ tried to undo your edits?
- ▶ tried several approaches to a problem, and had to use undo/redo?

Remember this?
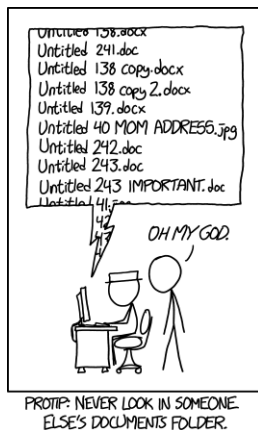
Figure: xkcd/1459

Branches

Git's version of an experiment is called a *branch*

Definition (Branch)

A pointer to a commit.

Remember that a commit knows who it's parent is, so if I have a pointer to a commit—a branch—I have an entire *branch* of history in the history tree.

Why would I do this?
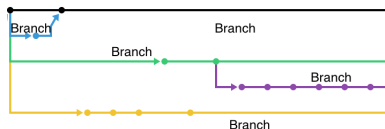
Figure: Some branches

- to separate work on feature A from feature B
- to experiment without affecting the main code
- to make pretty graphs

Potions Class & PPE: Experimenting Safely

Let's make a branch for a code experiment we want to do

Exercise (Branching)

1. Use the Branch menu to create a new branch and switch to it
2. In `lib.py`, implement add1 by changing `pass` to `return a + 1`
3. Commit (you've seen this before!)
4. Switch back to master and start a new branch
5. Implement sub1 the same way
6. Commit

Creating a branch
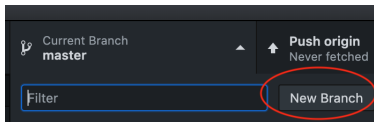
Figure: 1: Using the branch menu
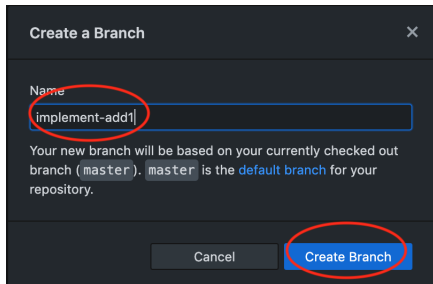

Figure: 1: Naming a branch

Implementing add1

Figure: 3: Creating the add1 commit

Implementing `sub1` on another branch
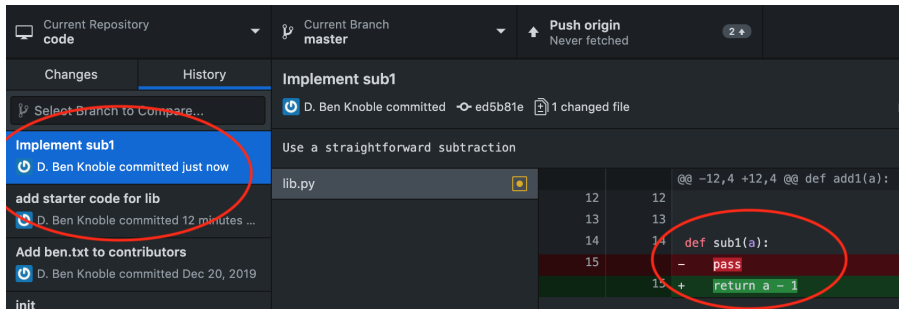
Figure: 6: Creating the `sub1` commit

Comparing the history
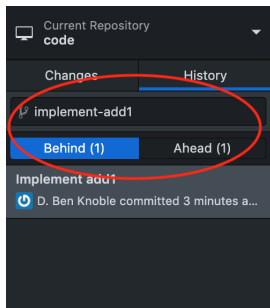
Figure: Some commits on each branch

The experiment is a success: now what?

We would ideally like to be able to *merge* the code and history in
our experiment with the code and history from our "main"
development.

Definition (Merge commit)

A commit (content snapshot) that combines history from multiple
branches.

`git` is (usually) able to figure out the best way to combine code
and history from one branch with code and history from another.
(We will take a look later at what happens when it cannot.)

Mixing Potions: Your First Merge
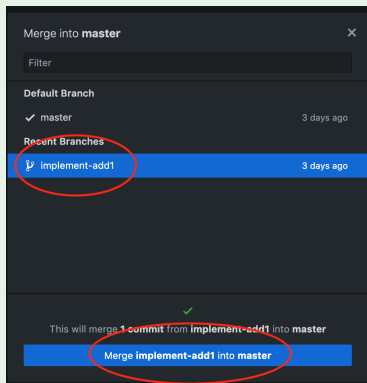
Exercise (Merging)

Figure: 1: Setting up the merge

Mixing Potions: Your First Merge

Exercise (Merging)

Figure: 2: Checking the results

Accio Code! Collaborating via Pull Requests

1. Group projects anyone?
2. Real jobs involve teams
3. In fact, *almost everything involves teams*

Conclusion

We need collaboration

Fundamentals of Distribution

Remote code

We need access to remote code. We've seen *push*, sending local history to remote history. We need *pull*, or getting remote history to local history.

Fundamentals of Distribution

Remote code

We need access to remote code. We've seen *push*, sending local history to remote history. We need *pull*, or getting remote history to local history.

History everywhere

Remember that, because `git` is distributed, there are multiple histories of the same project floating around! You might have one (or more) on each of

1. your computer
2. GitHub, GitLab, etc.
3. a partner's computer

and each one could be *different*.

Pull Requests

Definition (Pull Request)

A *request* that someone else *pull* your history into theirs.

The common process is this:

▶ You make useful commits to your copy of a project

▶ You ask other people to pull your new history into their copies

This almost always results in a merge being done: they probably added some commits in the meantime, so git has to combine the two histories.

GitHub

GitHub (and other web interfaces) provide tooling to facilitate the change-review-merge cycle.

Exercise (Creating a Pull Request)

1. (If you haven't already) Push your master branch
2. Open your fork in GitHub
3. Click "New Pull Request"
4. Change the base to your *partner's* fork
5. Investigate the UI
6. Write a description
7. Submit
8. View your partner's PR on your repo
9. Merge it

Synchronizing later is slightly more complicated. See the accompanying notes for links and explanations.
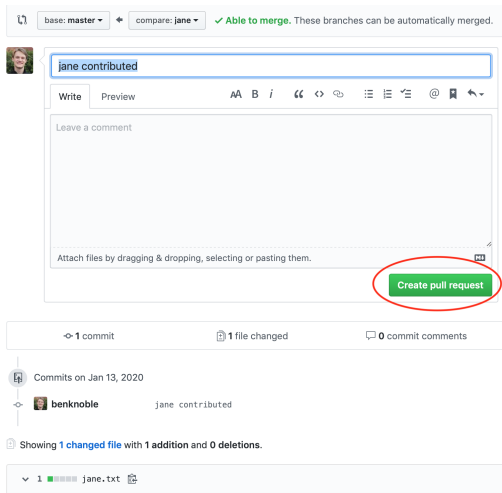
Figure: 3: Create a Pull Request UI

Figure: 8: Merging UI

Boggarts and Riddikulus: Merge Conflicts are not Scary!

Definition (Merge Conflict)

What `git` produces when it doesn't know how to merge changes together

Most common occurrence: different edits to the same line of code

The Best Practice is Doing

Exercise (Creating conflicts)

1. On a new branch, edit conflicts.yaml with your name's value
2. Commit
3. On master, repeat *with your partner's name*
4. Commit
5. Attempt a merge

Results

Figure: Joe (1/2: adding "my" name)

Figure: Jane (3/4: adding my partner's name)

Figure: 4: Attempting to merge. Notice the conflict markers? Lines like <<<HEAD, ===, >>>branchname These help you see what code came from where

Picking sides

Exercise (Resolving conflicts)

1. Decide which version to keep, and remove all the conflict markers and the other code (I picked Jane's code).
2. Commit

Most editors have settings, features, or plugins for working with merge conflicts.

Conclusion

Who Am I?

Ben Knoble

Summary

`git` is a great tool for collaboration: learn it and use it!

Resources

Linked on GitHub: shortcut

Reminders

Mentors are a great resource for help at Pearl Hacks

Upcoming Wizard Classes

TODO???