

Git Wizard 101

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
- ▶ Respect others' pronouns
- ▶ There are no dumb questions
- ▶ Help your neighbor when you can
- ▶ Have a good time!

About Me

- ▶ He/Him/His
- ▶ Senior (CS, French, Math), soon-to-be M.S. student
- ▶ [@benknoble](#) on GitHub

About Me

- ▶ He/Him/His
- ▶ Senior (CS, French, Math), soon-to-be M.S. student
- ▶ [@benknoble](#) on GitHub
- ▶ Conversationally proficient in French, D&D, puns

About Me

- ▶ He/Him/His
- ▶ Senior (CS, French, Math), soon-to-be M.S. student
- ▶ [@benknoble](#) on GitHub
- ▶ Conversationally proficient in French, D&D, puns
- ▶ And, of course, all things `git`

What We Will *Not* Be Doing



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

Figure: [xkcd/1597](#)

Required setup

1. git

Required setup

1. `git`
2. GitHub account

Required setup

1. `git`
2. GitHub account
3. GitHub Desktop [be sure to sign in with GitHub!]

Required setup

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

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

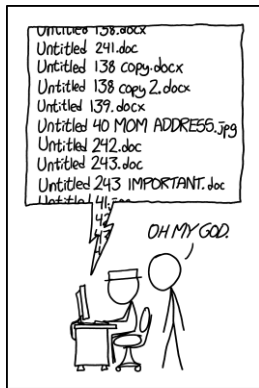
First Steps with git

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.

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

Figure: [xkcd/1459](#)

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`
2. Place your name in it
3. Click commit
4. Push to your fork

Lumos: Your First ~~Spell~~ Commit

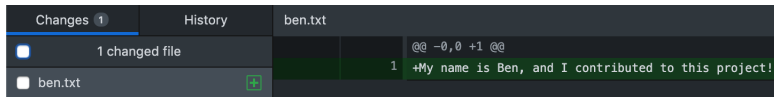


Figure: Create `ben.txt`

Lumos: Your First Spell Commit

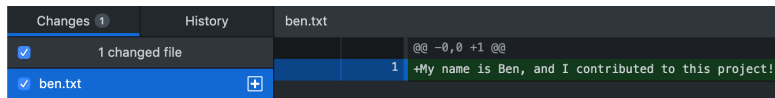


Figure: Add ben.txt

Lumos: Your First ~~Spell~~ Commit

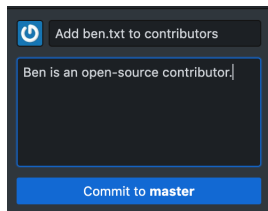


Figure: Commit

Lumos: Your First ~~Spell~~ Commit

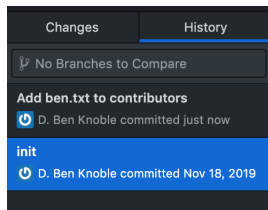


Figure: History

A Note about Messages



| | 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 |
| ○ | AAAAAAA | 3 HOURS AGO |
| ○ | ADKFJSLKDFJSOKLFJ | 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.

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

Figure: [xkcd/1296](#)

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?

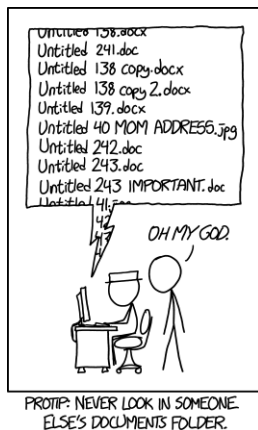


Figure: [xkcd/1459](#)

Branches

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

Definition (Branch)

A pointer to a commit.

Remember that a commit knows who its 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?

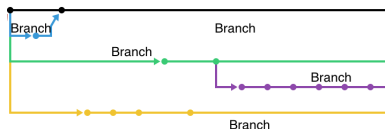


Figure: Some branches

- ▶ 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

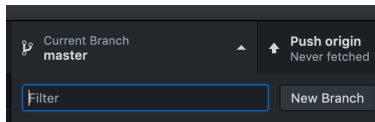


Figure: Menu

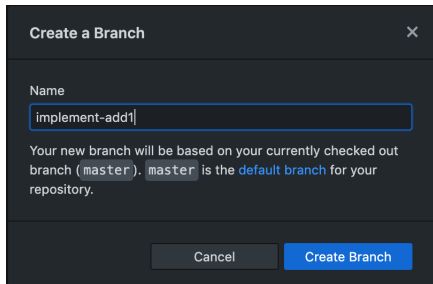


Figure: Naming a branch

Implementing add1

The screenshot displays the GitHub web interface for a repository. At the top, the 'Current Repository' is set to 'code', the 'Current Branch' is 'implement-add1', and there is a 'Publish branch' button. Below this, the 'Changes' tab is active, showing a list of commits. The commit 'Implement add1' by D. Ben Knoble is selected, showing it was committed a minute ago. The commit message is 'Use a straightforward addition this time'. The diff view shows changes to 'lib.py', with line 11 being added (green background) and line 10 being removed (red background). The code changes are as follows:

| Line | Old | New |
|------|------|--------------|
| 8 | 8 | 8 |
| 9 | 9 | 9 |
| 10 | 10 | 10 |
| 11 | - | def add1(a): |
| 11 | pass | return a + 1 |
| 12 | 12 | 12 |

Figure: The add1 commit

Implementing sub1 on another branch

The screenshot shows a Git web interface with a dark theme. At the top, there are three main sections: 'Current Repository code' with a dropdown arrow, 'Current Branch master' with a dropdown arrow, and 'Push origin' with a button labeled '2' and an upward arrow, and the text 'Never fetched'. Below these, there are two tabs: 'Changes' and 'History', with 'History' being the active tab. A search bar labeled 'Select Branch to Compare...' is visible. The main content area is titled 'Implement sub1' and shows a commit by 'D. Ben Knoble' with hash 'ed5b81e' and '1 changed file'. The commit message is 'Use a straightforward subtraction'. Below this, a diff view for 'lib.py' is shown. The diff highlights changes in lines 12 through 15. Line 12 is unchanged. Line 13 is unchanged. Line 14 is unchanged. Line 15 is changed from '-' to '+', and the text 'pass' is replaced with 'return a - 1'. The diff view also shows the context of the change, including the 'def add1(a):' function definition.

Current Repository code

Current Branch master

Push origin
Never fetched

2

Changes History

Select Branch to Compare...

Implement sub1

D. Ben Knoble committed just now

add starter code for lib

D. Ben Knoble committed 12 minutes ...

Add ben.txt to contributors

D. Ben Knoble committed Dec 20, 2019

init

Implement sub1

D. Ben Knoble committed ed5b81e 1 changed file

Use a straightforward subtraction

lib.py

| lib.py | | | |
|--------|----|--------------|--------------------------------|
| 12 | 12 | | @@ -12,4 +12,4 @@ def add1(a): |
| 13 | 13 | | |
| 14 | 14 | | def sub1(a): |
| 15 | - | pass | |
| | + | return a - 1 | |

Figure: The sub1 commit

Comparing the history

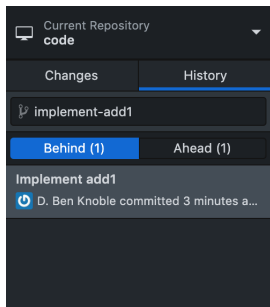


Figure: Some commits on each branch

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)

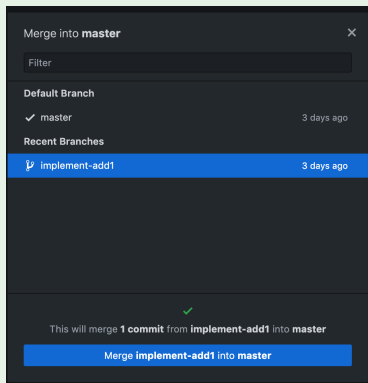


Figure: Setting up the merge

Mixing Potions: Your First Merge

Exercise (Merging)

The screenshot shows a Git GUI interface with the following components:

- Current Repository:** code
- Current Branch:** master
- Push origin:** Never fetched (4 commits)
- Changes/History:** The History tab is selected, showing a list of commits:
 - Merge branch 'implement-add1' (D. Ben Knoble committed just now)
 - Implement sub1 (D. Ben Knoble committed 3 days ago)
 - Implement add1 (D. Ben Knoble committed 3 days ago)
 - add starter code for lib (D. Ben Knoble committed 3 days ago)
- Merge branch 'implement-add1':** D. Ben Knoble committed 9f96c99, 1 changed file.
- Diff View:** A comparison of the 'lib.py' file between the two branches.

| lib.py | | | |
|--------|----|----|---------------------|
| | 8 | 8 | @@ -8,7 +8,7 @@ def |
| | 9 | 9 | |
| | 10 | 10 | def add1(a): |
| | 11 | - | pass |
| | | 11 | + return a + 1 |
| | 12 | 12 | |
| | 13 | 13 | |

Figure: 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.

A set of small navigation icons typically found in Beamer presentations, including symbols for back, forward, search, and other slide controls.



jane contributed #1

 **Open** benknoble wants to merge 1 commit into `master` from `jane` 


Conversation 0 Commits 1 Checks 0 Files changed 1


 **benknoble** commented now  


No description provided.


  jane contributed 5e5f750


Add more commits by pushing to the `jane` branch on `benknoble/git-wizard-code`.



 **Continuous integration has not been set up**
[GitHub Actions](#) and [several other apps](#) can be used to automatically catch bugs and enforce style.

 **This branch has no conflicts with the base branch**
Merging can be performed automatically.

Merge pull request  You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

 Write Preview **AA B i “ <> 🔗 ☰ ☷ ☹ @ 📌 ↶**

Leave a comment

Figure: 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

```
@@ -1,4 +1,4 @@
1 ---
2 # this is some configuration file for your proje
3 ct
4 -key: "Ben's value"
5 +key: "Joe's value"
```

Figure: Joe

```
@@ -1,4 +1,4 @@
1 ---
2 # this is some configuration file for your proje
3 ct
4 -key: "Ben's value"
5 +key: "Jane's value"
```

Figure: Jane

```
2 ---
1 # this is some configuration file for your project
0 <<<<<< HEAD
1 key: "Jane's value"
2 ||||| merged common ancestors
3 key: "Ben's value"
4 =====
5 key: "Joe's value"
6 >>>>>> conflicts
7 ---
```

Figure: Notice the conflict markers?

Usually 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???