GEOG 4/590: Geospatial Data Science

Lecture 6: Code management

Email: jryan4@uoregon.edu
Office: 163A Condon Hall
Office hours: Monday 15:00-16:00 and Tuesday 14:00-15:00

Final project ideas

- Max: Tsunami evacuation times from specific locations on the Oregon coast
- Emerson: Weather/precipitation and unsafe driving conditions
- Hana: vegetation recovery times in response to fire regimes/disturbances in Siberia
- Dalton/Timmy: Spatial statistics of Chicago crime data
- Addy: Ice breakup in the Yukon River Delta
- Ethan: School funding and education outcomes
- Anna A:

Final project ideas

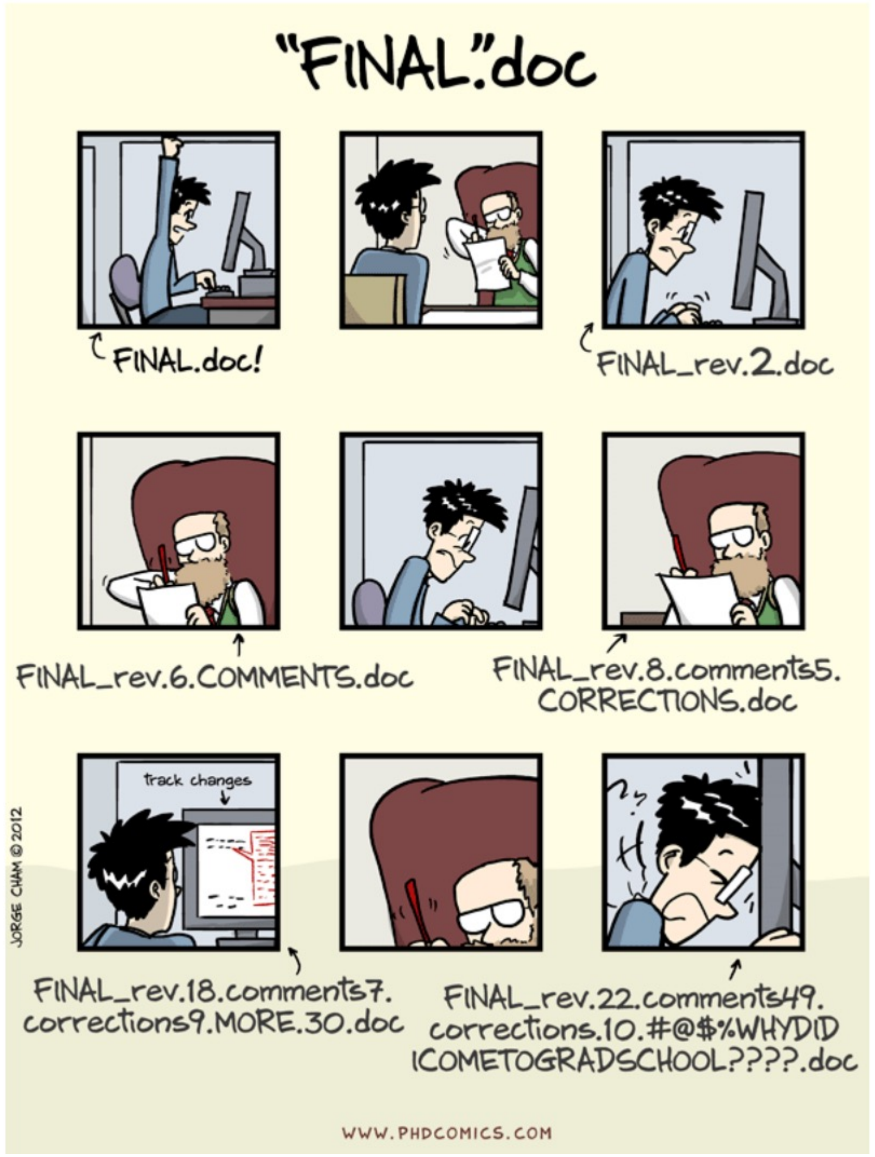
- Anna J:
- Isaac:
- Kelly:
- Sam G:
- Sam F:
- Bowie:
- Shauny:

Final project ideas

- Parker:
- Adamaryz:
- Haley:
- Lauren:
- Devlin:
- Kent:
- Jasper:

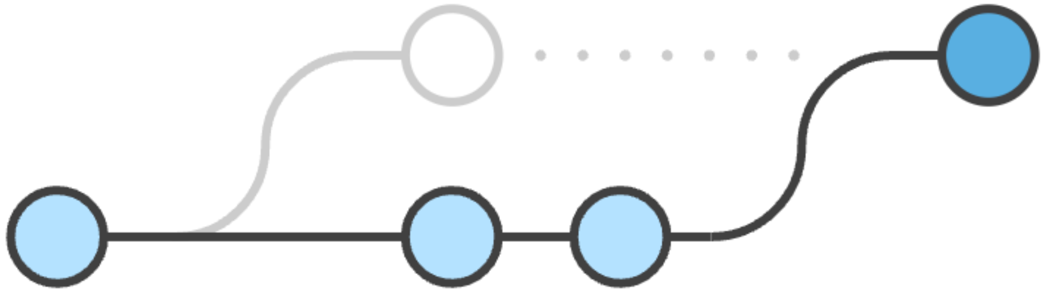
Code management

In this demo we will learn about using **version control** to collaborate on programming projects.



Code management

- **Version control systems** (VCS) start with a **base version** of the document and then **keeps track of changes** you make each step of the way
- VCS are essential for **developing software** and **carrying out projects** with a lot of code
- VCS does not care about file names, intead records **who, what, when, and why** changes were made to files



Git

- One of the most popular VCS tools in use today is called `git`
- It is a command-line tool that is installed locally
- It is free and open-source software

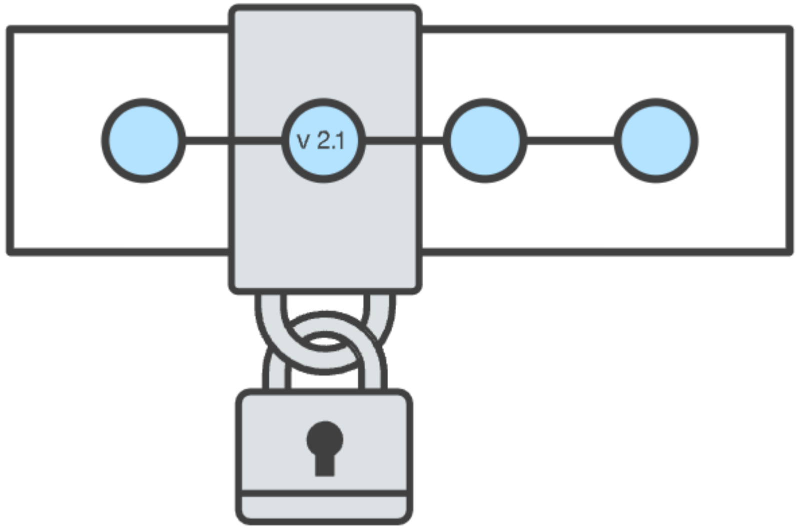
GitHub

- **GitHub** is a web-based hosting service for `git`
- Provides a graphical user interface
- Maintained by Microsoft
- There are other web-based hosting services (e.g. **GitLab** and **Bitbucket**)

Why do we use version control systems?

Security

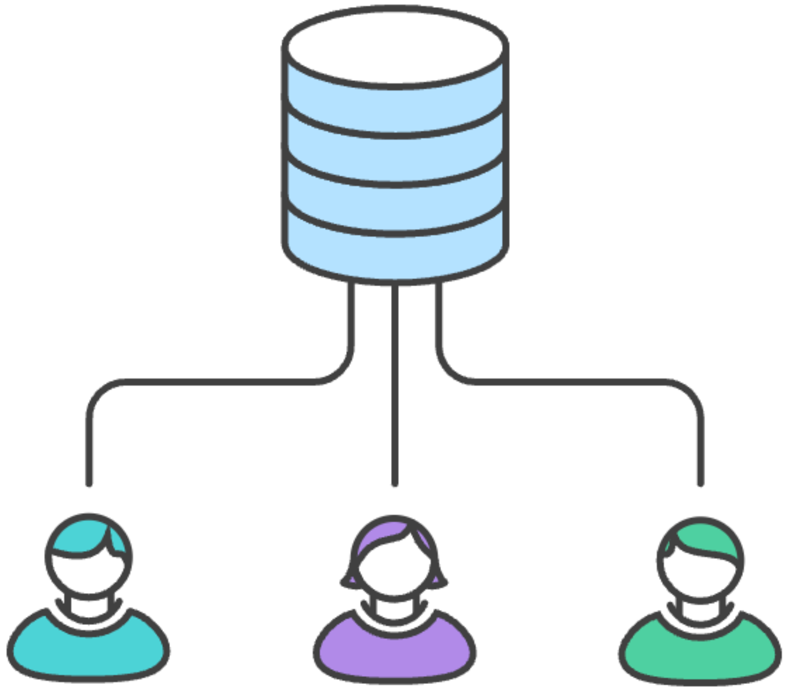
- VCS acts like an unlimited **‘undo’** thereby **protecting source code** from yourself **and** others
- e.g. catastrophe, human error, and unintended consequences



Why do we use version control systems?

Collaboration

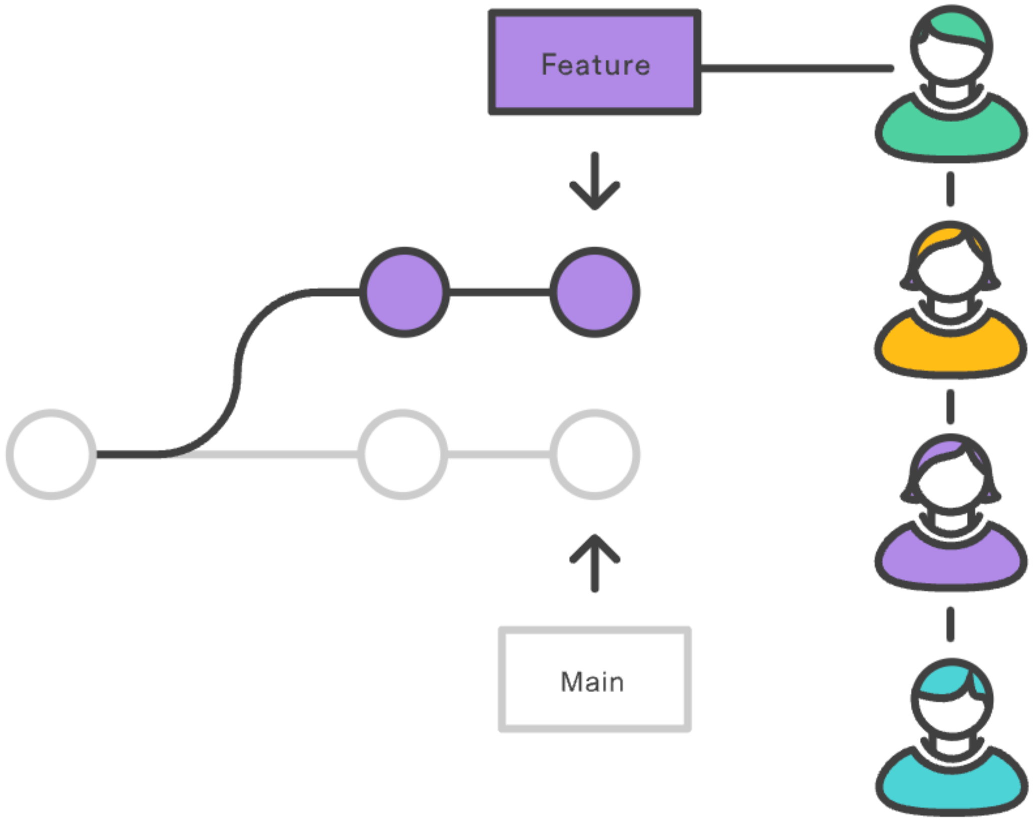
- VCS enables **many people** to work on the same project at the same time
- Teams working in parallel accelerates project development



Why do we use version control systems?

Community

- Impossible for junior developer to mess up a big project
- Since it is so robust this encourages open-source **experimentation** and **development**
- `GitHub` has really emerged as the industry standard



Drawbacks of version control

- Difficult to learn

Some basic terms

Fork

- **Copy** a repository to your GitHub.com account

Clone

- Retrieve a repository from **GitHub.com** to **local machine**

Commit

- Create a **snapshot** of the contents of your file tree

Some basic terms

Push

- **Upload** your local changes to the central repository, along with necessary commits and objects

Pull

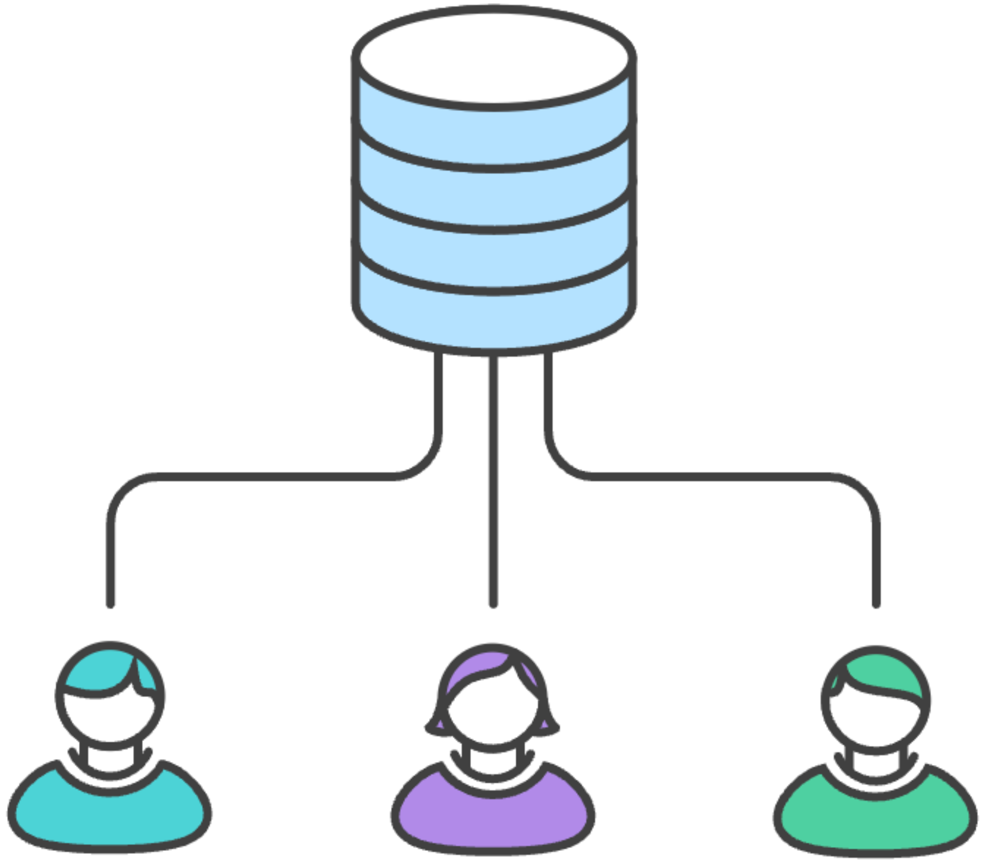
- **Fetch** the contents of the central repository and immediately **merge** to your local copy

Collaborating with GitHub

- Centralized workflow
- Feature branch workflow
- Forking workflow
- Others (e.g. Gitflow workflow)

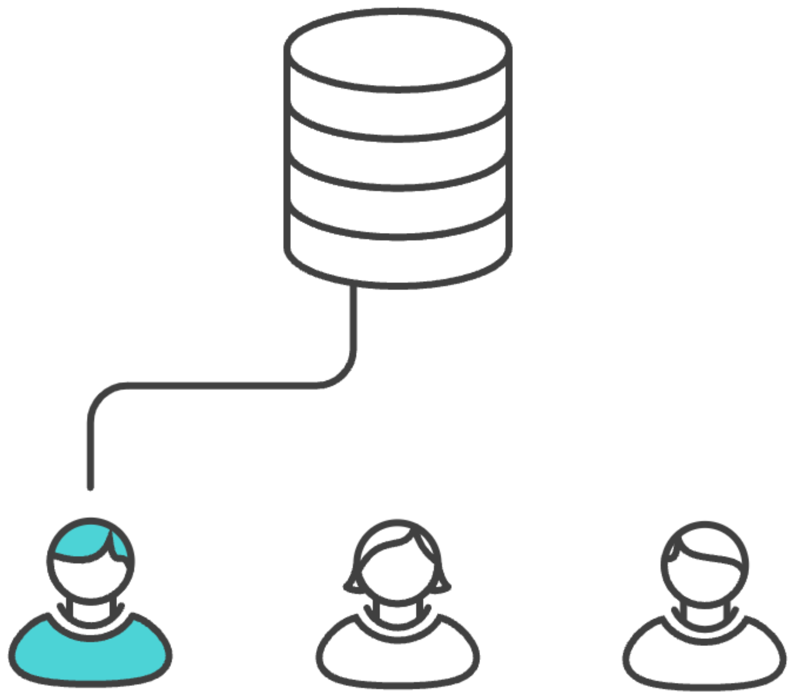
Centralized workflow

- All team members **clone** a **single, central repository** to their local machine



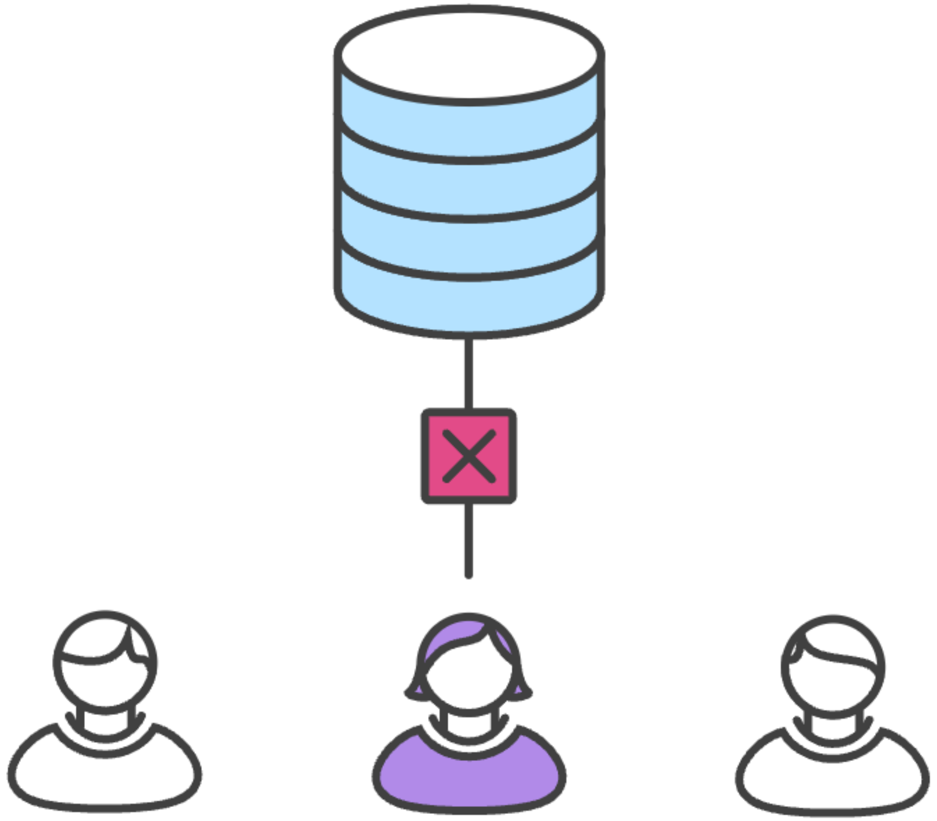
Centralized workflow

- One team member makes changes (e.g. add, modify, delete) to files on their local machine
- Periodically, they should **commit** these changes (i.e. take a snapshot) with a short message saying what they did
- When they are finished working, they can **push** their changes back to the central repository



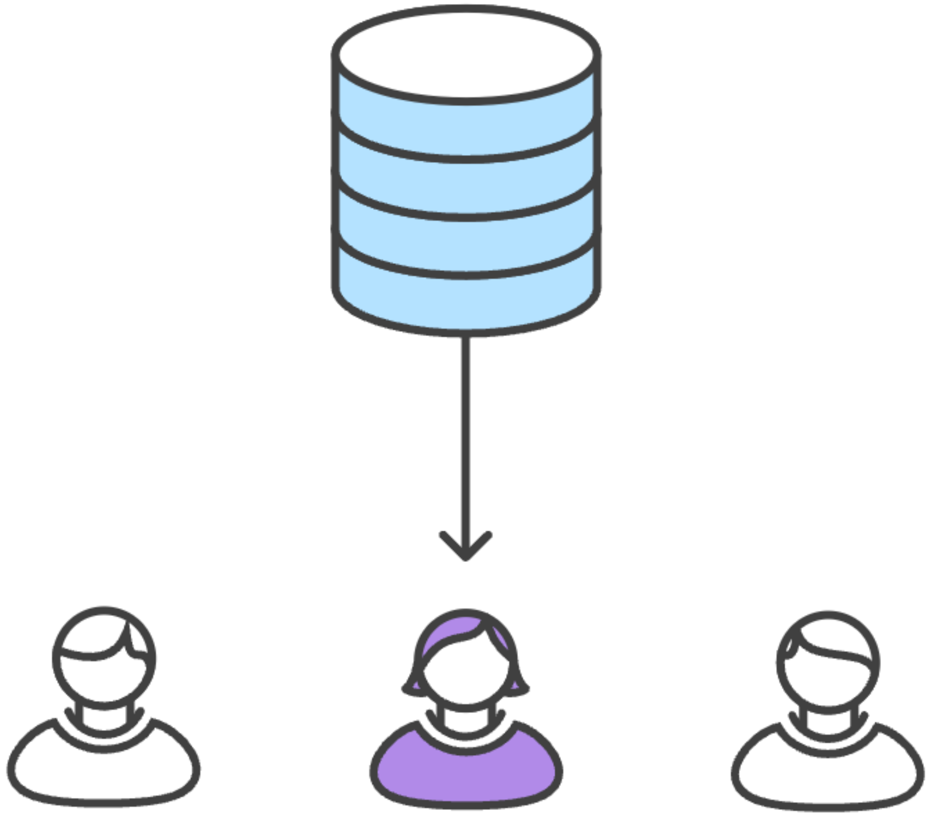
Centralized workflow

- But now when **another team member** (who has also been working on the project) tries to **push** their changes, Git will **refuse the request** because the their local history has **diverged** from the central repository



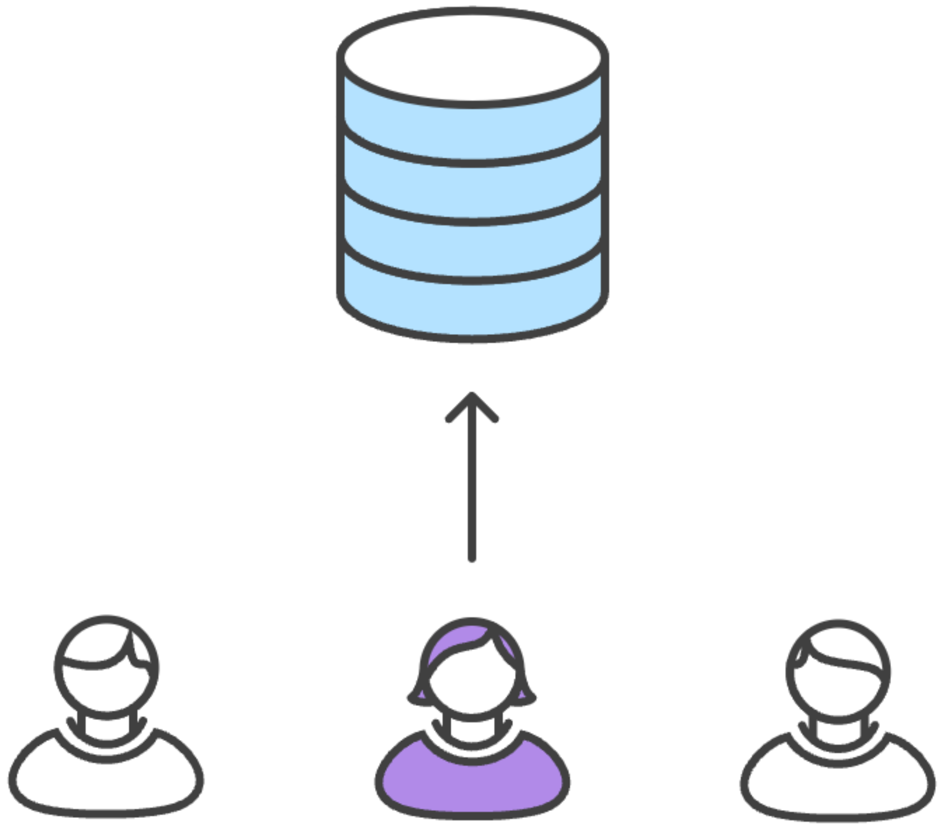
Centralized workflow

- The team member must first **pull** the most recent changes in the central reposistory into their local repository



Centralized workflow

- Team member then resolves any conflicts between their local version and the central repository.
- Once finished, team member can then **commit** and **push** their changes to the central repo



Centralized workflow

Advantages

- Simplest workflow
- Works well for small teams

Disadvantages

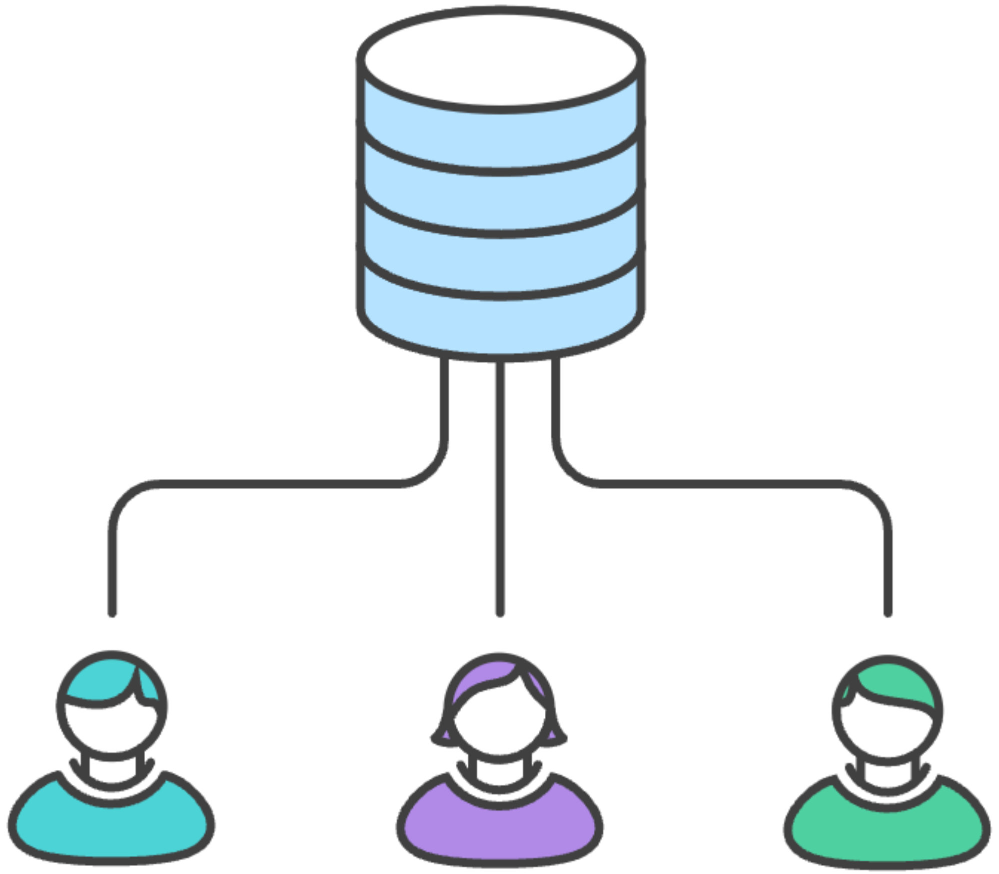
- If someone breaks the central repo, it breaks for everyone
- Potential for a lot of conflicts
- One solution is to avoid working on the same files
- But this does not scale well as teams increase in size

Feature branch workflow

- The logical extension of the centralized workflow is to use **branches**
- In this workflow, all feature development takes place in a dedicated branch instead of the main branch
- This means that main branch never contains broken code - a huge advantage for continuous integration environments

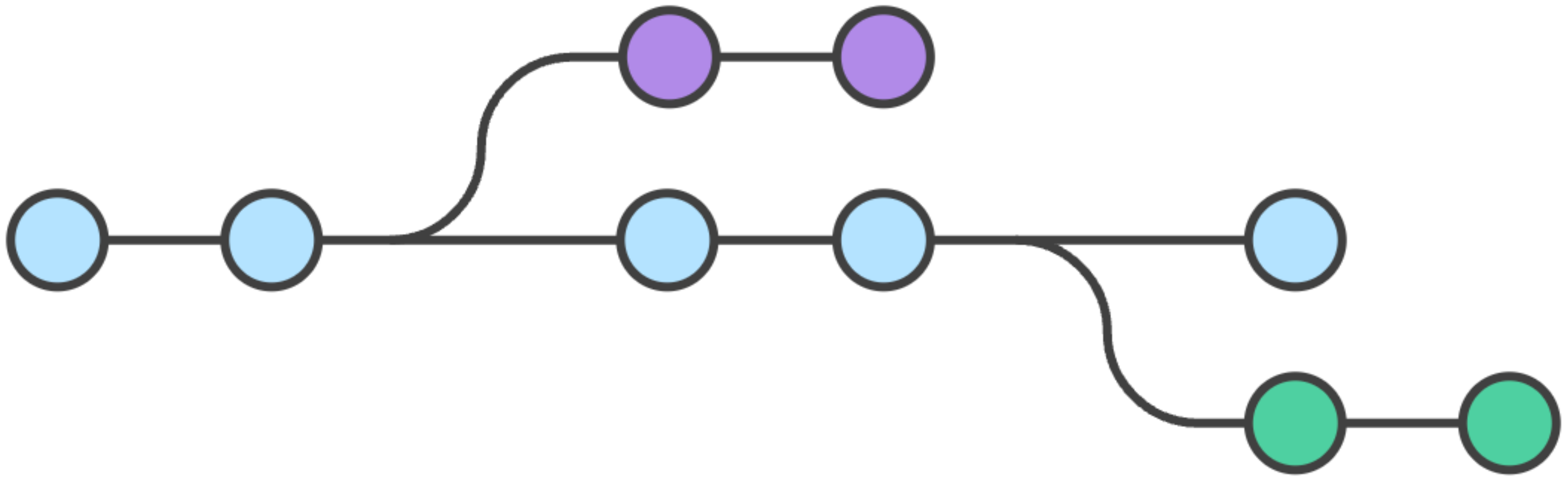
Feature branch workflow

- All team members **clone** a **single, central repository** to their local machine



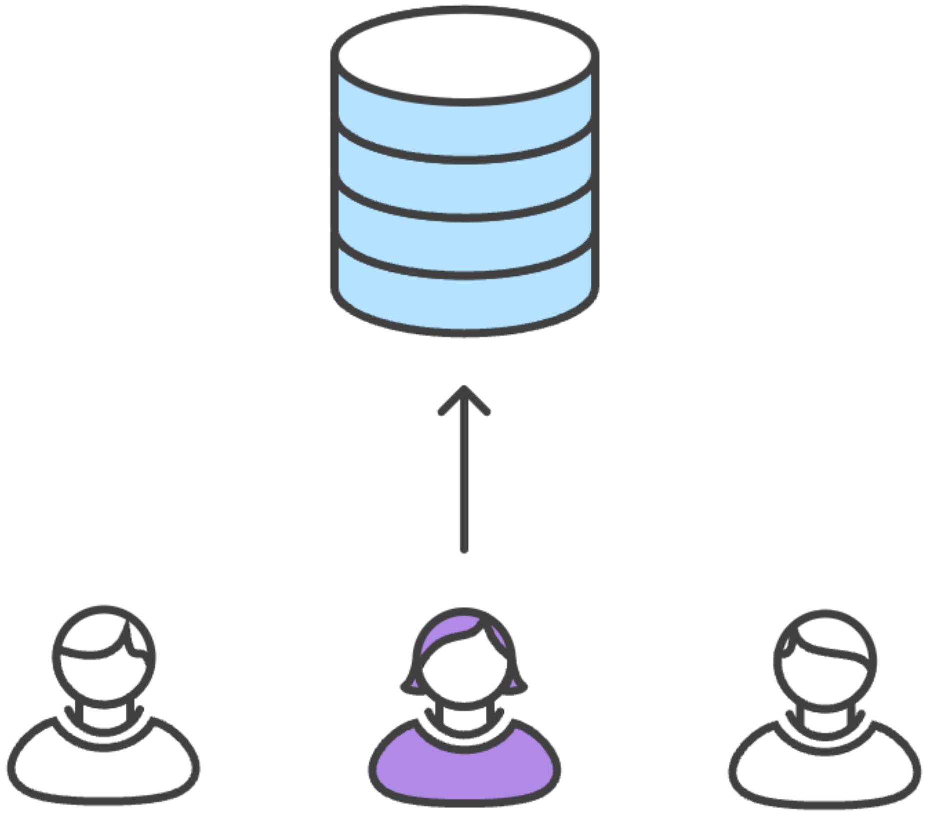
Feature branch workflow

- Team members immediately create a new branch to make their changes



Feature branch workflow

- When team members finish their changes, they **push** their branch to the central repository. The central repository will now contain multiple branches.
- Therefore, unlike the centralized workflow, this **push** will never cause conflicts

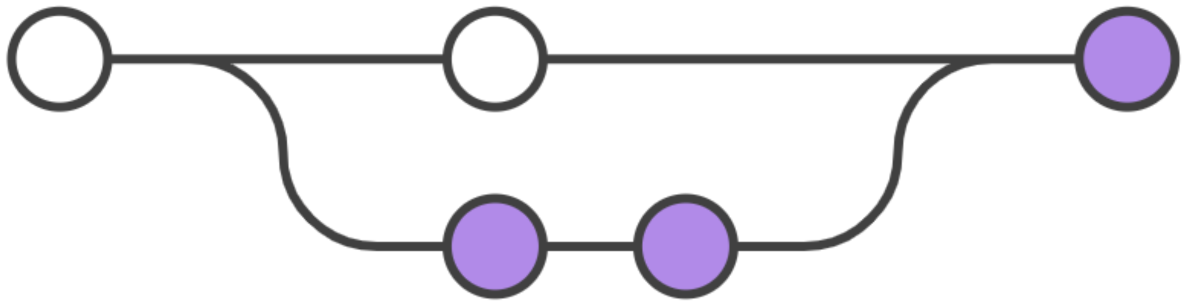


Feature branch workflow

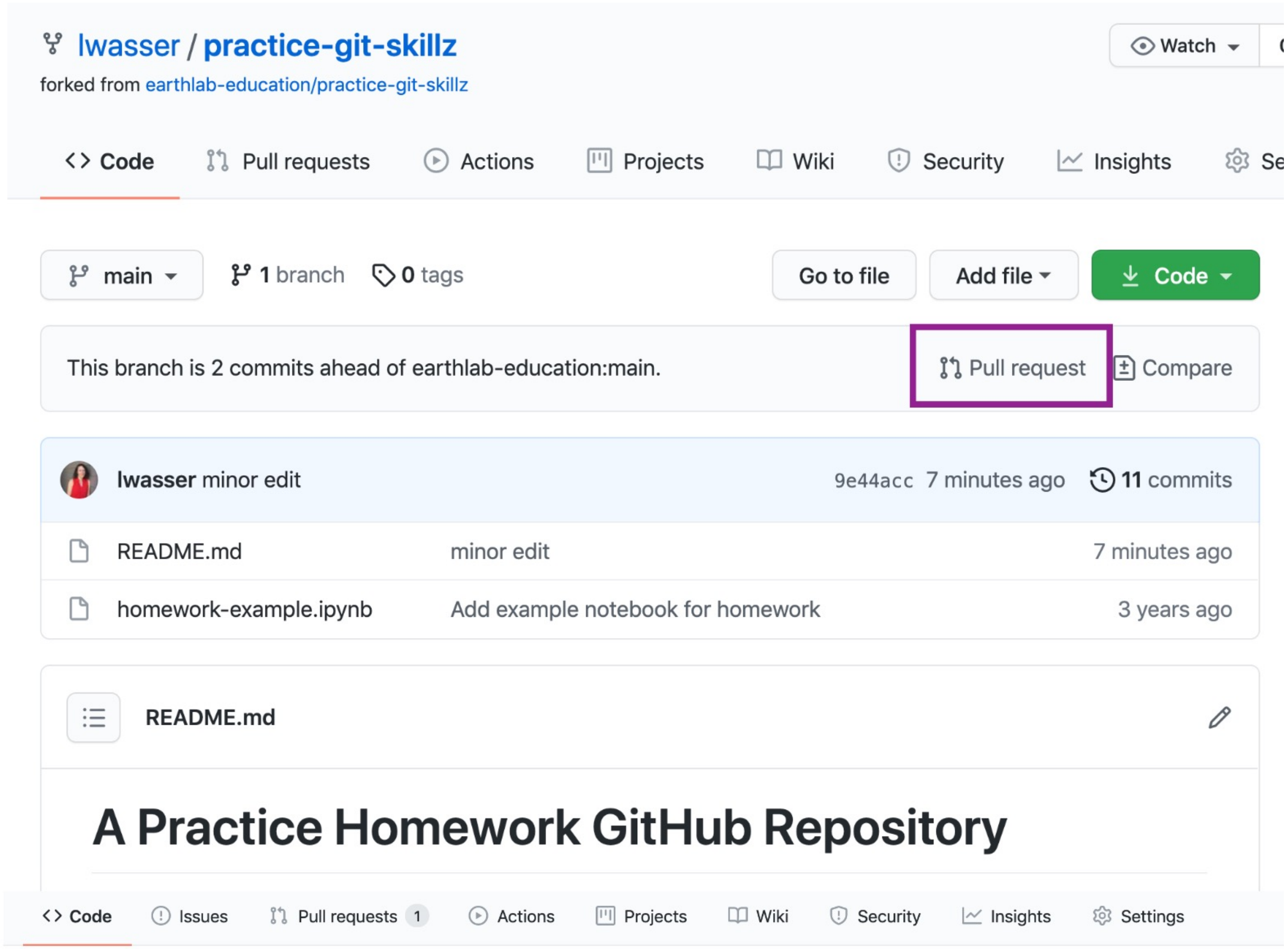
- Team members then submit a **pull request** on GitHub.com asking to **merge** their new feature (or branch) into the main codebase, all team members will be notified automatically

Feature branch workflow

- Team leader **reviews** pull request, discusses any changes with team members
- Once everything looks good, team leader merges new feature into main codebase
- Team member can then delete their branch



Feature branch workflow



lwasser / practice-git-skillz

forked from earthlab-education/practice-git-skillz

This branch is 2 commits ahead of earthlab-education:main.

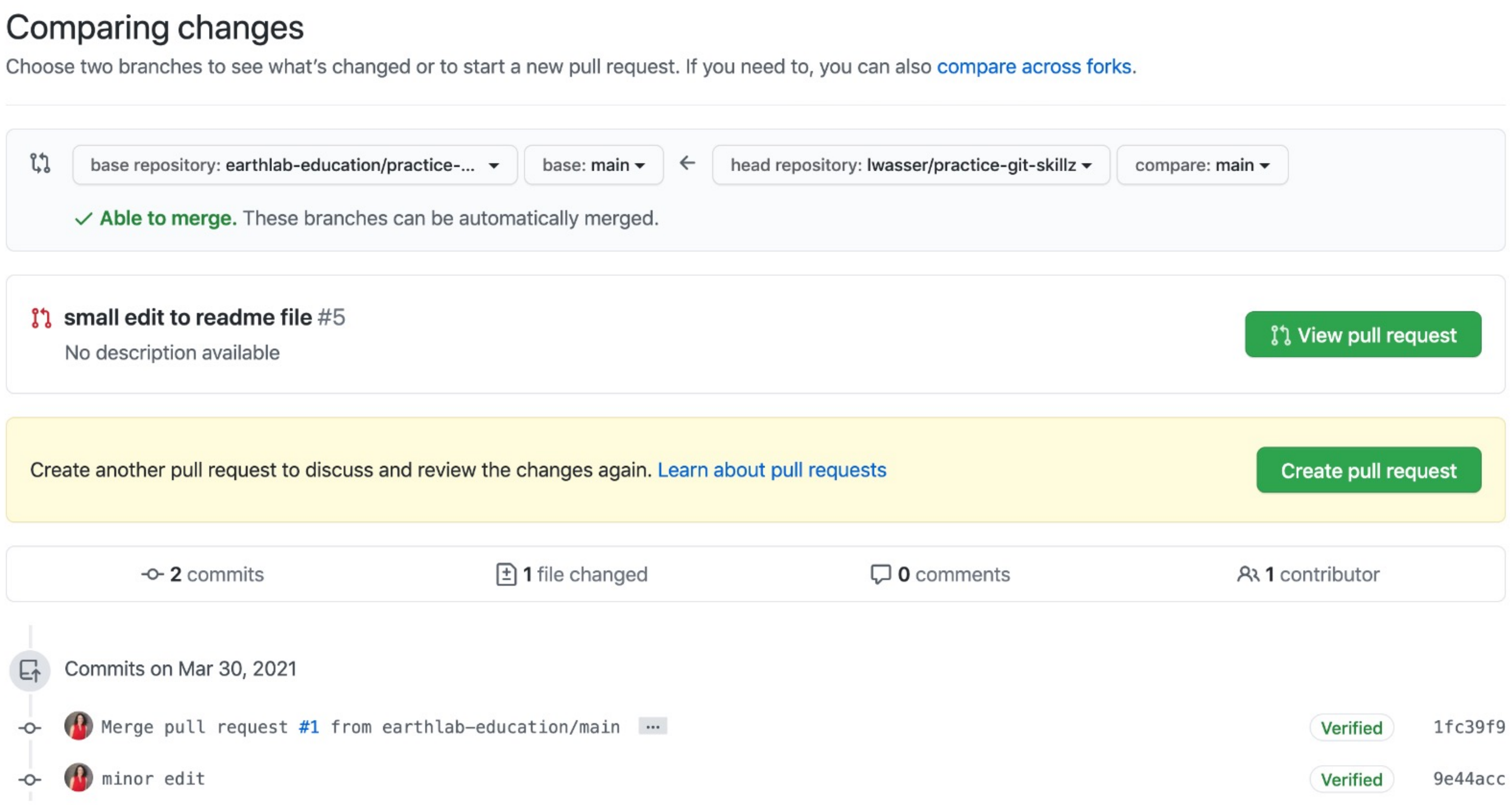
lwasser minor edit 9e44acc 7 minutes ago 11 commits

README.md	minor edit	7 minutes ago
homework-example.ipynb	Add example notebook for homework	3 years ago

README.md

A Practice Homework GitHub Repository

Feature branch workflow



Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also compare across forks.

base repository: **earthlab-education/practice-...** base: **main** ← head repository: **lwasser/practice-git-skillz** compare: **main**

✓ **Able to merge.** These branches can be automatically merged.

small edit to readme file #5

No description available

View pull request

Create another pull request to discuss and review the changes again. Learn about pull requests

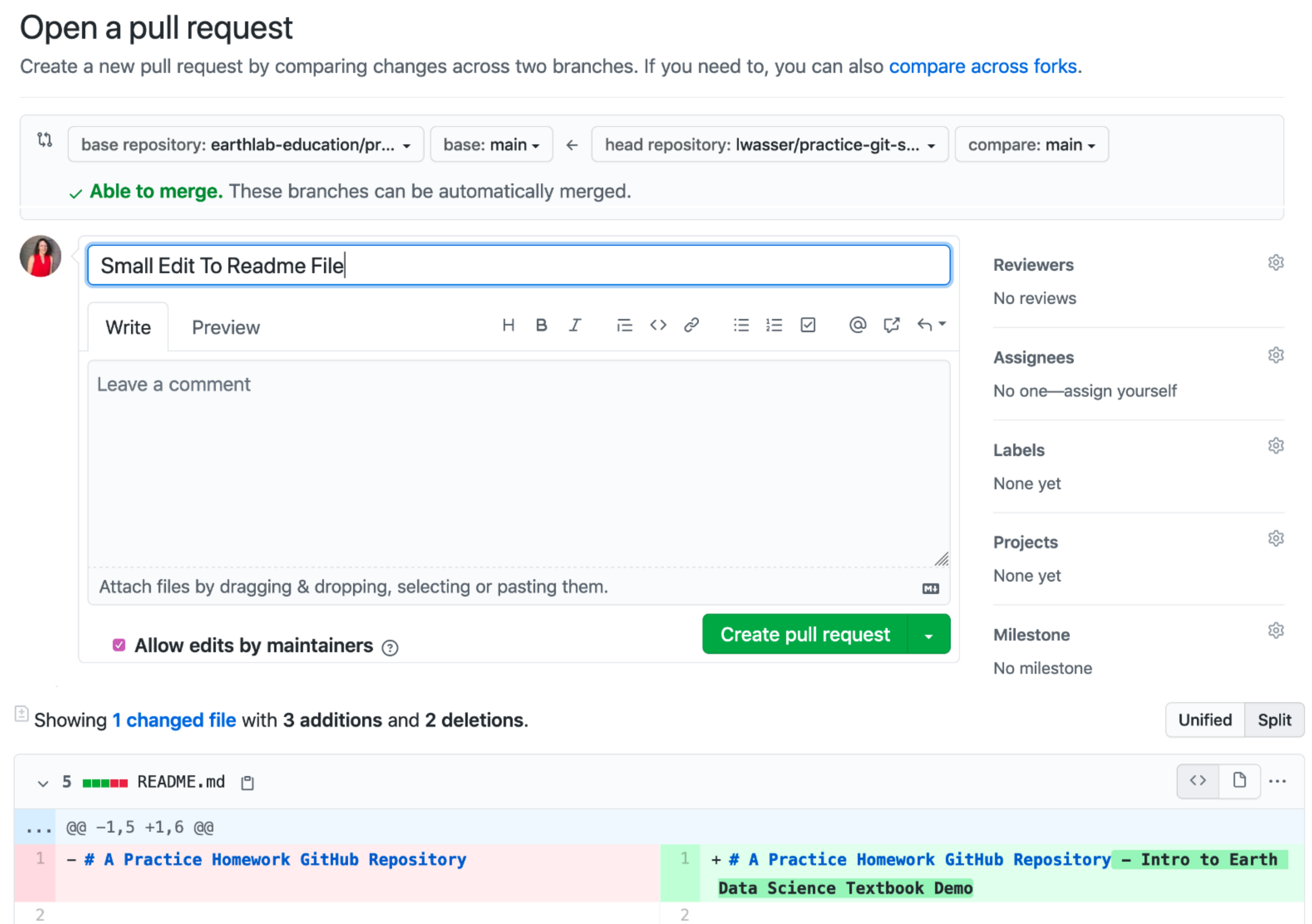
Create pull request

2 commits **1** file changed **0** comments **1** contributor

Commits on Mar 30, 2021

```
Merge pull request #1 from earthlab-education/main ...    Verified  1fc39f9
minor edit                                                Verified  9e44acc
```

Feature branch workflow



Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks.

base repository: earthlab-education/pr... ▾ base: main ▾ ← head repository: lwasser/practice-git-s... ▾ compare: main ▾

✓ **Able to merge.** These branches can be automatically merged.

Small Edit To Readme File

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

☑ **Allow edits by maintainers**

Showing 1 changed file with **3 additions** and **2 deletions**.

```
5 README.md
@@ -1,5 +1,6 @@
1 - # A Practice Homework GitHub Repository
1 + # A Practice Homework GitHub Repository - Intro to Earth Data Science Textbook Demo
2                                           2
```

Advantages of feature branch workflow

- Promotes collaboration with team members through **pull requests** and **merge reviews**
- Teams can work in parallel on same files so good approach for larger teams
- Main branch never contains broken code
- Guiding framework for other, more complex worflows

Advantages of feature branch workflow

- Instead of using a single, central repository, forking workflows give every team member their **own central repository**
- Team members can tinker with their forked repository as they wish without disturbing anyone else
- When ready they can **push** to their private central repository and file **pull requests** if they think their changes are ready to be integrated to main codebase

Advantages of feature branch workflow

- Provides a little more **power** to the team leader because they are the only person that can push to the official repository
- Allows the team leader to **accept/reject commits** from any developer without giving them write access to the main codebase
- Often used for large open-source projects

Good practices

Agree on a workflow

- It is important that teams establish shared patterns of collaboration
- If a team doesn’t agree on a shared workflow it can lead to inefficient communication when it comes time to merge branches

Good practices

Agree on a workflow

- It is important that teams establish shared patterns of collaboration
- If a team doesn’t agree on a shared workflow it can lead to inefficient communication when it comes time to merge branches

Commit often

- Commits are **easy to make** and provide opportunities to **revert** or **undo** work
- They should be made **frequently** to capture updates to a code base

Good practices

Agree on a workflow

- It is important that teams establish shared patterns of collaboration
- If a team doesn’t agree on a shared workflow it can lead to inefficient communication when it comes time to merge branches

Commit often

- Commits are **easy to make** and provide opportunities to **revert** or **undo** work
- They should be made **frequently** to capture updates to a code base

Ensure you’re working from latest version

- VCS enables rapid updates from multiple developers
- It’s easy to have a local copy of the codebase fall behind the global copy
- Make sure to `git pull` or `fetch` the latest code before you start working on project

Good practices

Make detailed notes

- It is important to leave descriptive explanatory commit log messages. These commit log messages should explain the “why” and “what” that encompass the commits content.
- These log messages become the canonical history of the project’s development and leave a trail for future contributors to review.

Good practices

Make detailed notes

- It is important to leave descriptive explanatory commit log messages. These commit log messages should explain the “why” and “what” that encompass the commits content.
- These log messages become the canonical history of the project’s development and leave a trail for future contributors to review.

Use branches

- Branches enable multiple developers to work in parallel on **separate lines** of development
- Branches should be used **frequently** as they are quick and inexpensive.
- When development on a branch is complete it should be **merged** into the main line of development and then **deleted**

There are two ways to use `git`, the command-line and **GitHub Desktop**. Most students prefer to use the desktop version to begin with but we’d be happy to provide guidance on the command-line version during labs.

Next time: Data access

Email: jryan4@uoregon.edu
Office: 163A Condon Hall
Office hours: Monday 15:00-16:00 and Tuesday 14:00-15:00