Introduction to Version Control with git and github

Bryan Scott

LSSTC Data Science Fellowship Program Pre-Orientation

August 31, 2023

Some git Resources

Pro Git book available at git-scm.com/docs

and

Beginning Git and GitHub: A Comprehensive Guide to Version Control, Project Management, and Teamwork for the New Developer by Mariot Tsitoara

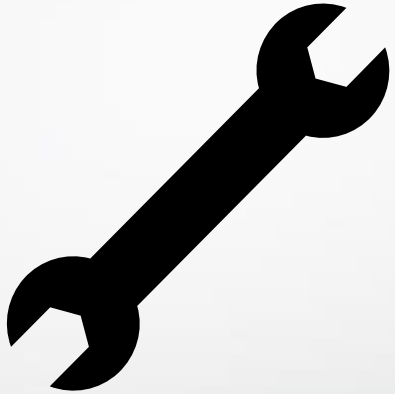
What makes learning git difficult?

What makes learning git difficult?

Let's be real – development workflows…

Brilliant idea!

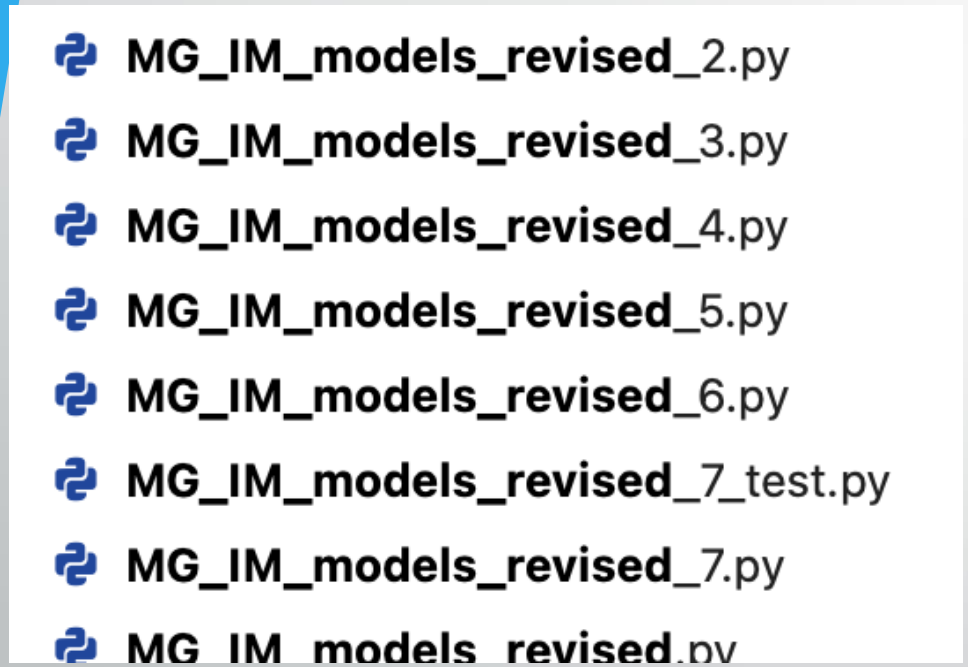
Prototyping



Iterative Fixes

Oh no!

Problems in typical development workflows…



Paper_draft_final.tex
Paper_draft_final_revised.tex
Paper_draft_final_revised_final.tex
Paper_draft_final_revised_final_submission.tex
Paper_draft_final_revised_final_submission_with_comments.tex
Paper_draft_final_revised_final_submission_with_comments_and_responses_draft.tex
(and so on….)

What is version control?

Version control is a *system* for creating a *reproducible* record of changes to a *project*.

The key idea is that there is only **one** project. *No matter* how big or complicated that project is.

Git is a *distributed* version control system – there is no authoritative master repository and each copy is as valid as any other.

The usefulness of software repositories for different kinds of developers...

For "small projects", tracking changes with git (or another VCS) and making regular commits makes your work more reproducible and gives you a "global undo" if you introduce a catastrophic error into your code.

For "large projects", a distributed VCS can allow multiple developers to work independently and synchronously. Changes are merged later (and conflicts resolved appropriately).

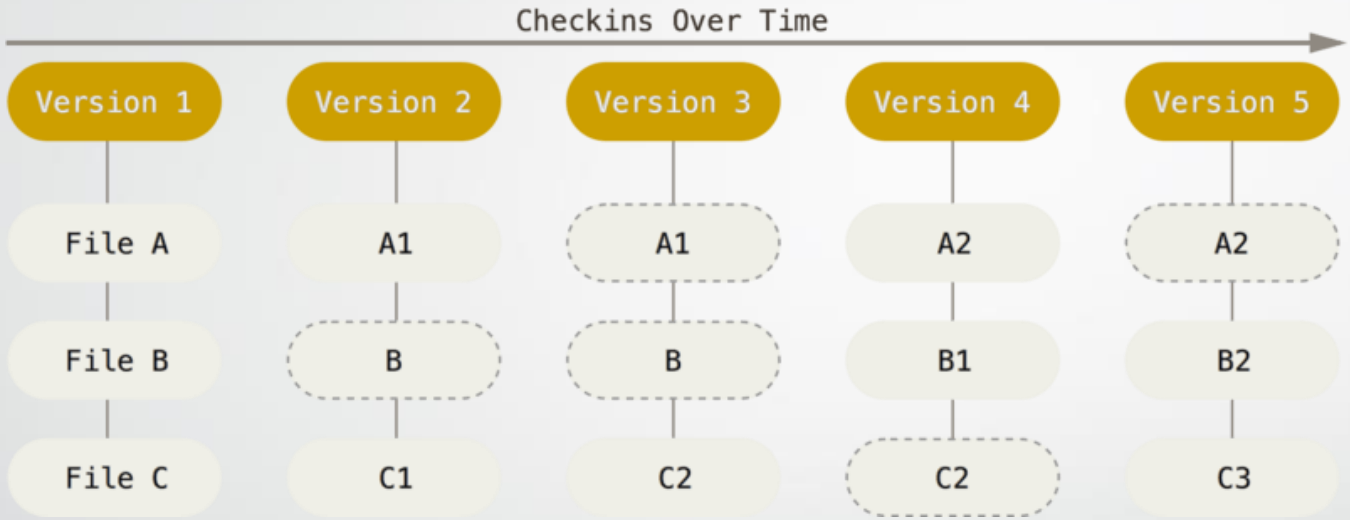
Introduction to git

Distributed Version Control

The Distributed Version Control Model (Git)

- A software repository is a place for storing software and metadata about it.
- In a distributed version control system, each developer has a copy of the repository – while it may be convenient to choose a "central" copy as a way of aiding collaboration – there is no authoritative version of the repository.
- Git is an example of a distributed version control system.

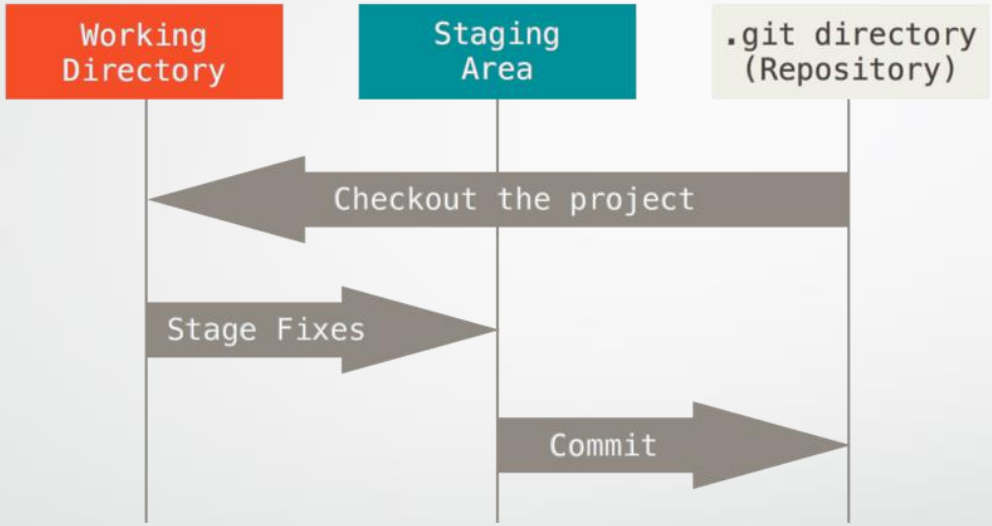
How does Git work?



Git stores snapshots of the project over time. If a new file is added or an old file changed, the new version will be stored in the next snapshot.

But if a file is unchanged, git simply links to the version stored in the previous snapshot.

Git States

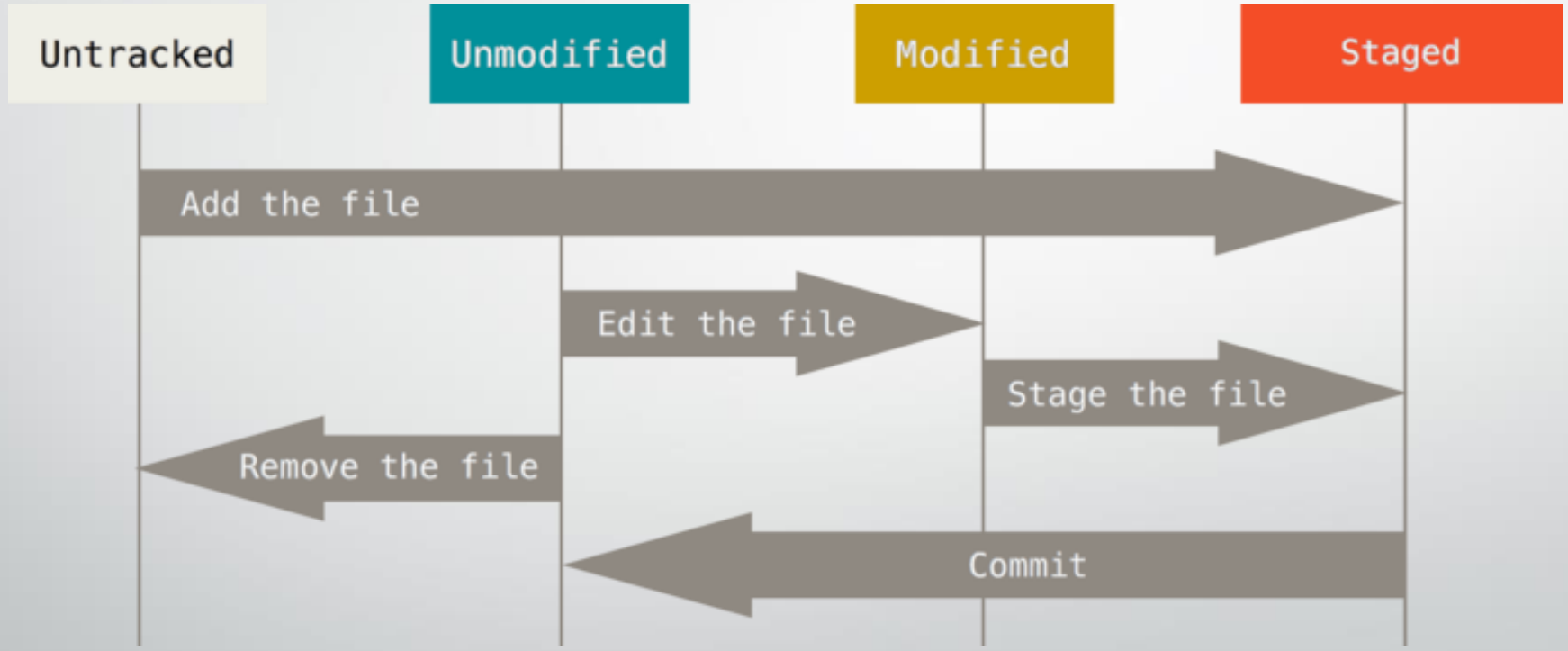


Files can be in one of three states. *Modified, staged, and committed.* Files move between the three stages as you work. After editing a file it is *modified.* You then *stage* the file which tells git to include it in the next snapshot. You then *commit* the changes (save a snapshot of the project).

Git Usage – Changes to the Repository

- Files in your project can be in one of two states – *tracked* or *untracked*
- Git tracks all files that were in the last snapshot (*commit*) or that have been created and staged with the git *add* command.
- To add changed files to the repository, you need to stage them with *add* and then snapshot with *commit*

Git File Cycle



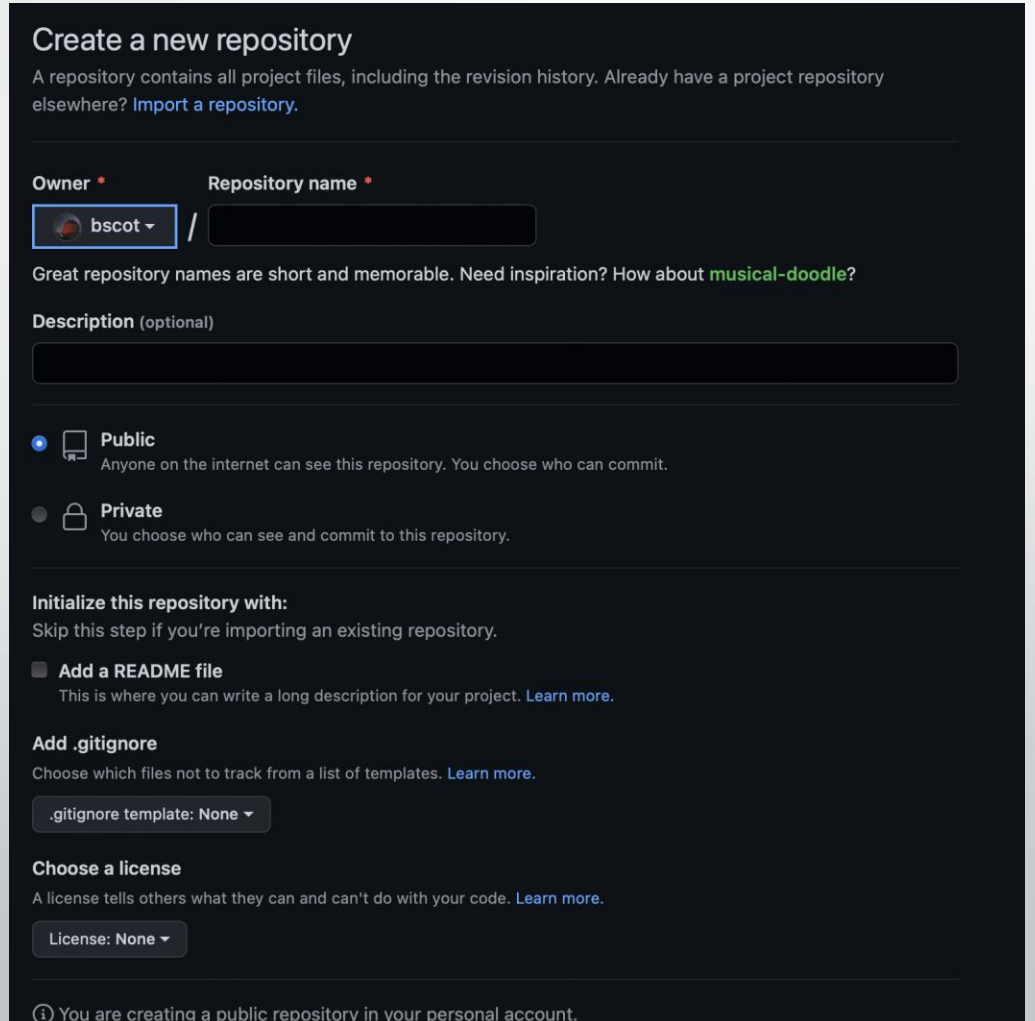
Introduction to Github

+ branches, merges, and more

Git as a distributed version control system

- Remember, git is a distributed version control system. Up until now, we have considered only a local copy of the software repository on our machine.
- But there's nothing special about our copy – we can have n copies of our project that are shared across many developers. We'll need to figure out how to manage n copies (not easy!), but the power of being able to share and collaborate is obvious!
- It would help if we had a convenient place to store a copy that we'll all work from – this is where **github** or **gitlab** come in.

Create a remote repository on github



Two ways to create a Repository

- We can create a local repository by navigating to a directory and typing

```
git init
```

- Or we can clone a repository from remote by typing

```
git clone [link to remote]
```

Linking the local and remote repositories

- Once we have a remote repository, we can link the local to it with git *remote* add [name] [link]. By convention, origin or upstream are used as the name of the remote.

```
Git remote add origin [link to your fork of the DSFP repo]
```

- You will sometimes see this command – this sends your code to the remote repository

```
git push origin main
```

- Where origin is the [remote name] and [main] is the branch we want to push.

  Caution: the command git push origin *master* was the default until recently. Main has replaced it as the preferred/default name for your production branch.

Git branches

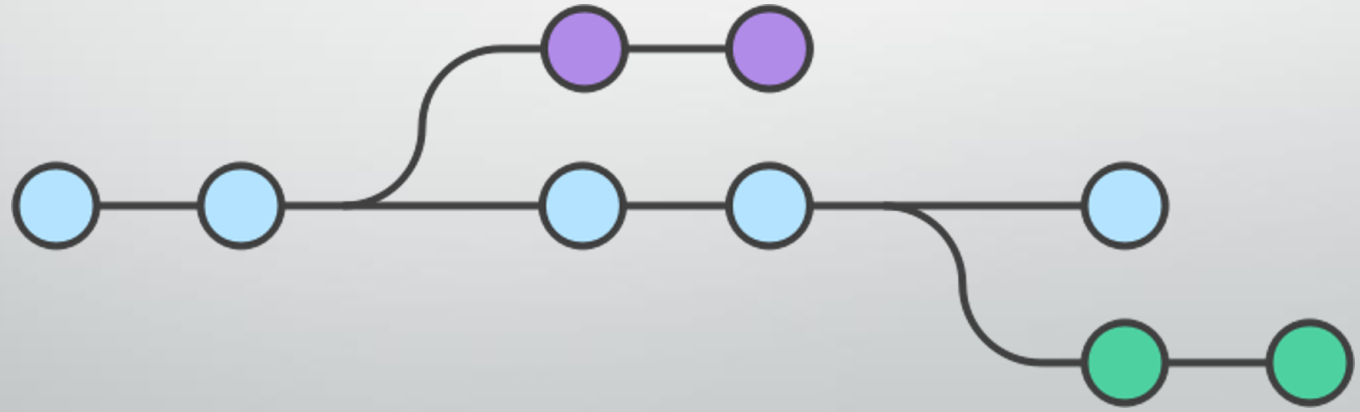
- Let's say we want to add a new feature to our code. We don't want to impact our previous version with bugs that the new feature introduces.

- We could copy the project and work on a whole new copy, perhaps, project_revised.py would be a new file in the copy. Or we could branch the project – recognizing we're still working on the same project, just adding something to it.

To branch our project

- The branch command is

```
git branch [branch_name]
```



Giving branch the –d flag will delete the referenced branch

Pushing branches to remote – Pull Requests

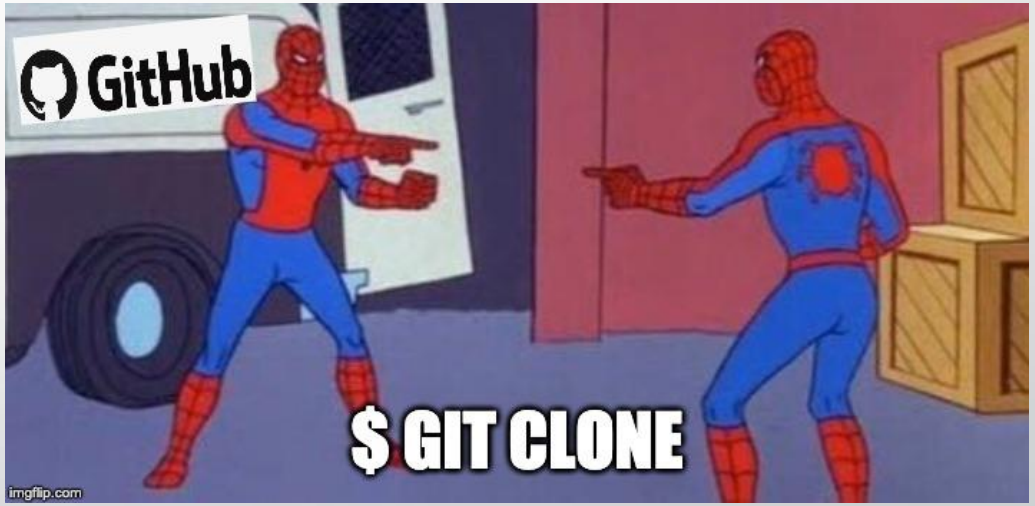
- The output of git pushing a branch to the remote repository will either perform a merge automatically or you'll need to create a pull request.

- If there are conflicts, it is sometimes possible to resolve the conflicts through the github GUI. Typically you will need to resolve it locally and push the new version.

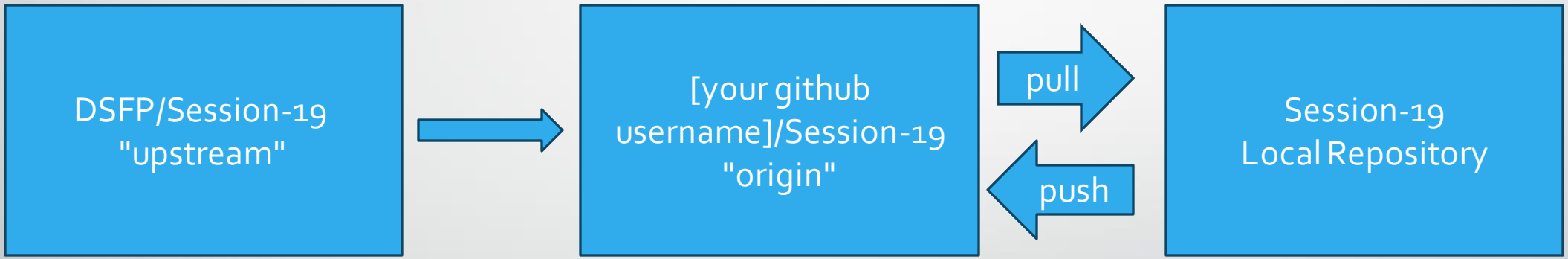
The terminology here is a little weird and confused me for a long time. Push and pull are just opposite actions based on my perspective – am I copying into my branch or copying out of my branch. Remember: git is distributed, so relationships between repos are symmetric.

Putting this into practice

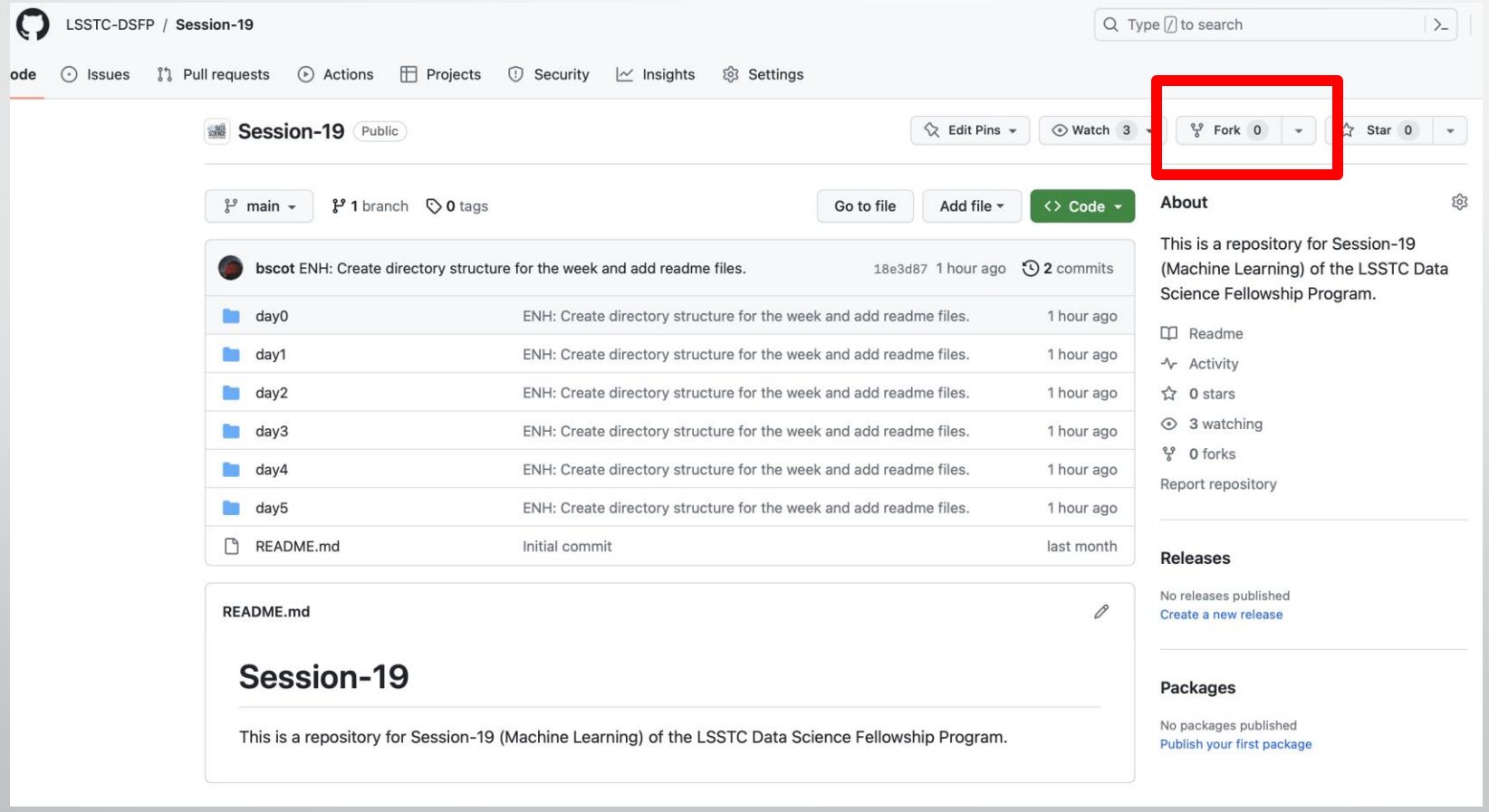
We will now go through forking and cloning the DSFP Repository to create a local version on your machine.



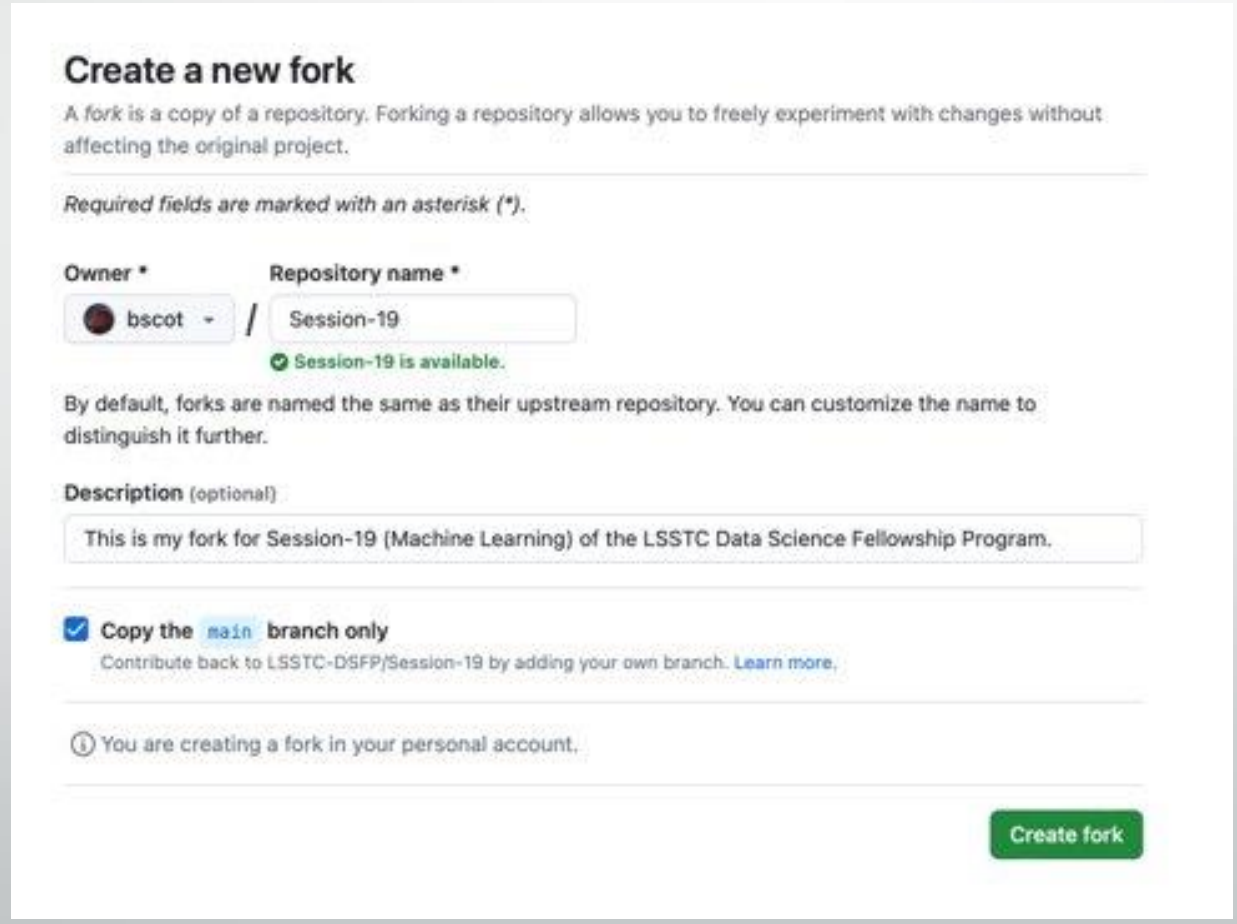
What we want to do



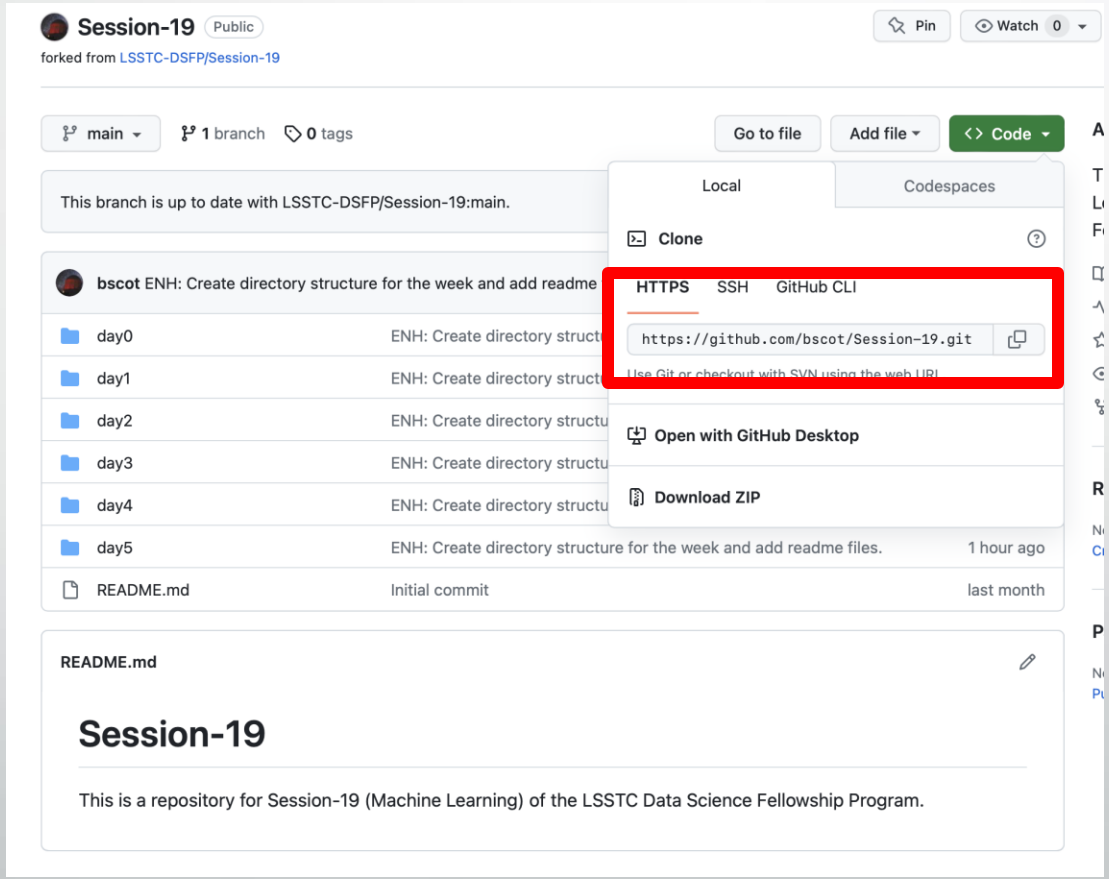
Fork the Session-19 Repo



Fork the Session-19 Repo



Now clone the repository locally



Now clone the repository locally

git clone [link you copied]

```
(base) bryan@Bryans-MacBook-Pro-8 Documents % git clone https://github.com/bscot
/Session-19.git
Cloning into 'Session-19'...
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 7 (delta 0), reused 4 (delta 0), pack-reused 0
Receiving objects: 100% (7/7), done.
(base) bryan@Bryans-MacBook-Pro-8 Documents %
```

Finally, let's link this to the 'official' Session-19 Repo

git remote add upstream https://github.com/LSSTC-DSFP/Session-19.git

```
(base) bryan@Bryans-MacBook-Pro-8 Session-19 % git remote
origin
upstream
```

git remote pull upstream main

```
(base) bryan@Bryans-MacBook-Pro-8 Session-19 % git pull upstream main
From https://github.com/LSSTC-DSFP/Session-19
 * branch            main       -> FETCH_HEAD
Already up to date.
```

Using git in the DSFP

Before each day, pull from the upstream repository to get the day's materials (and any other new material from the previous day)

- You can either pull from upstream directly ("git pull upstream main") [you'll then need to push to origin ("git push origin main") to bring everything up to date.
- Or use the github GUI to bring your origin repository up to date, then pull origin. [preferred]

At the end of each day (or more frequently), push your changes/saved work to origin

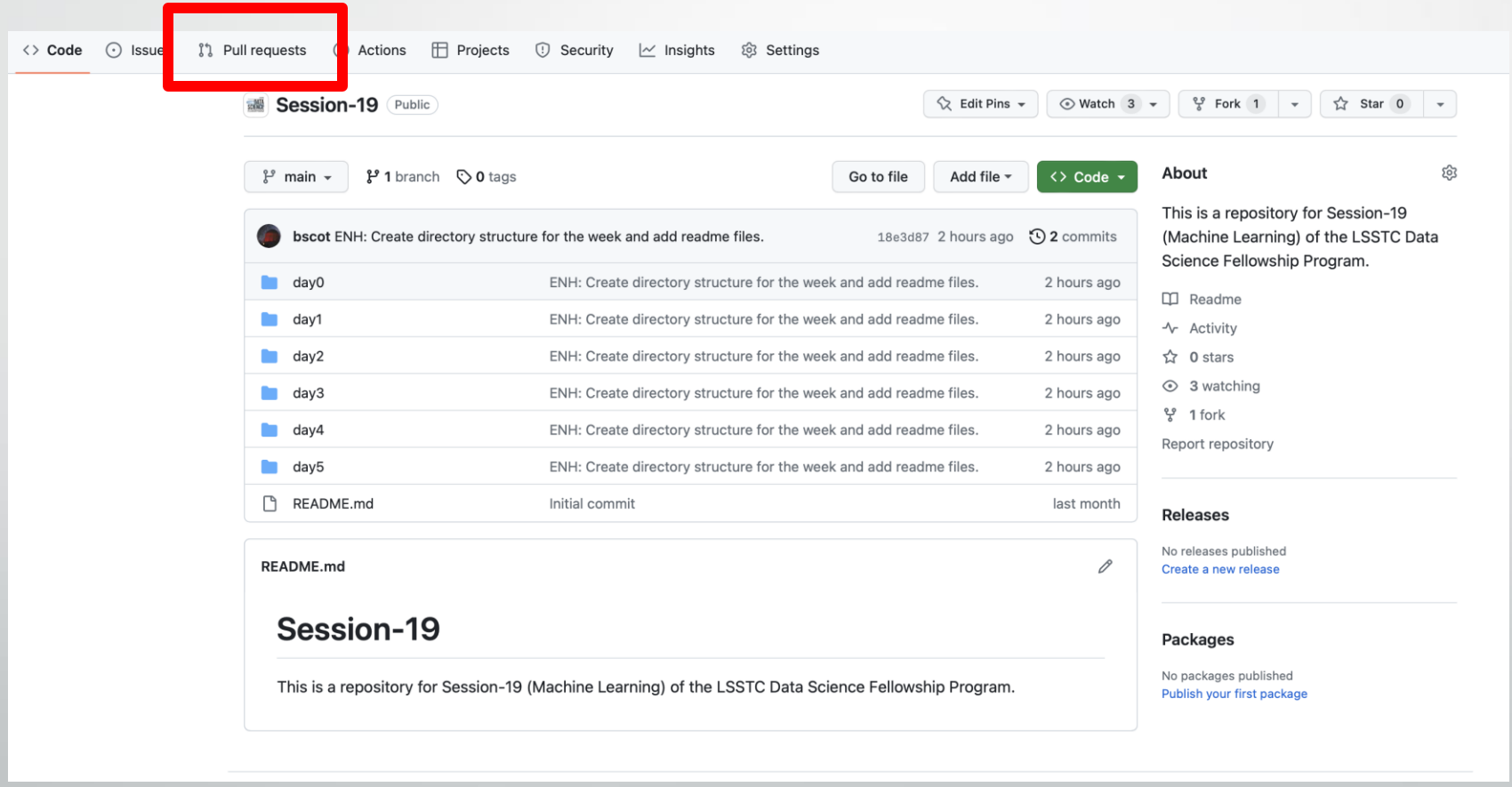
- "git push origin main"
- You may have merge conflicts – if two commits differ on their version of a line of a code – we'll discuss how to handle merge conflicts in detail at a future session.

Bonus: Pull Requests

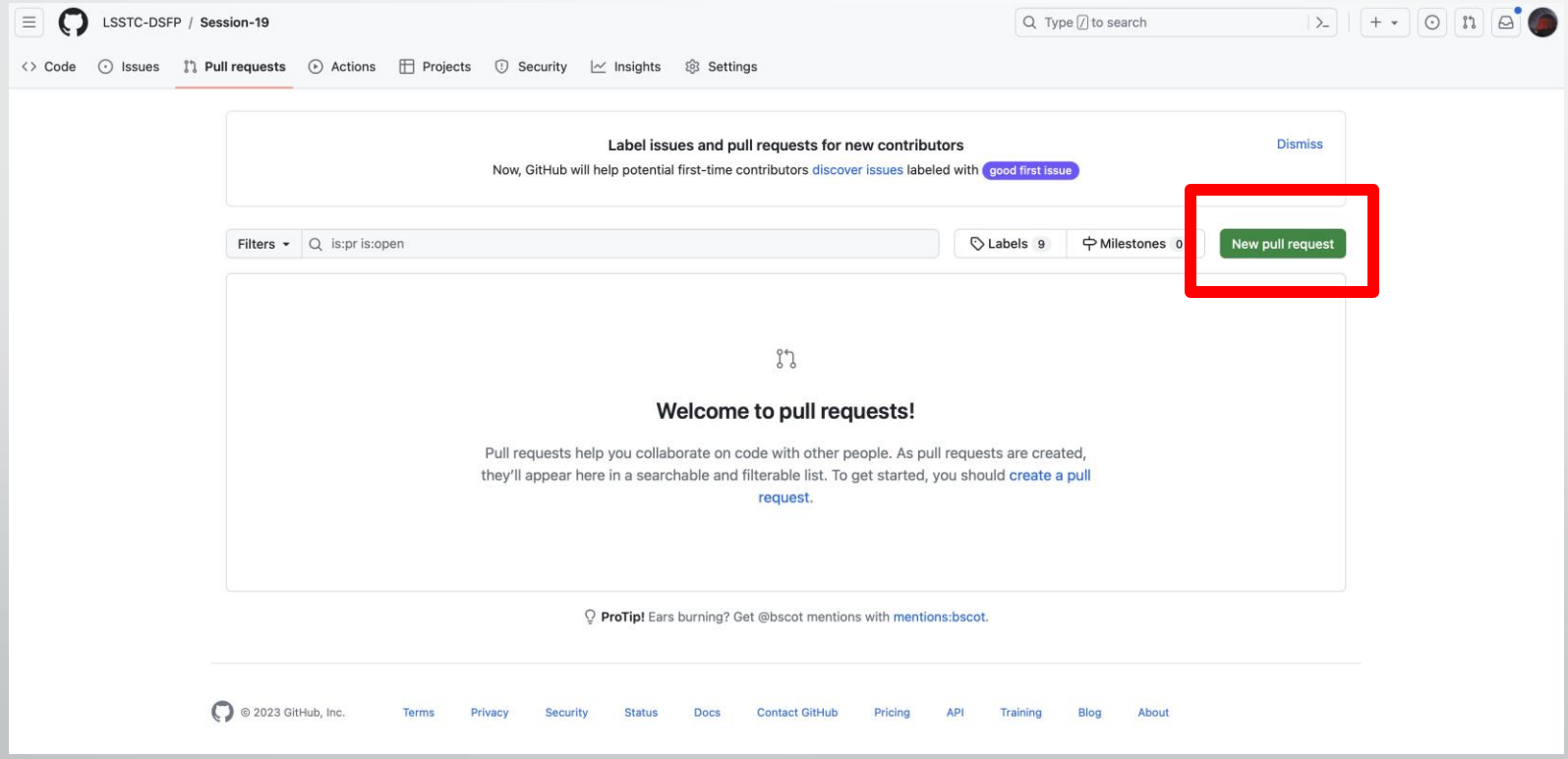
For this next part, we'll put you pairs in breakout rooms.

- One of you should begin by forking the other's Session-19 repository.
- Then make an edit to the README.md file to add your name as an author of the repository
- Once you've done that, open a pull request in the original repository to merge in the changed README.md
- The other partner should then review the pull request to confirm they agree with the changes and approve or deny the pull.

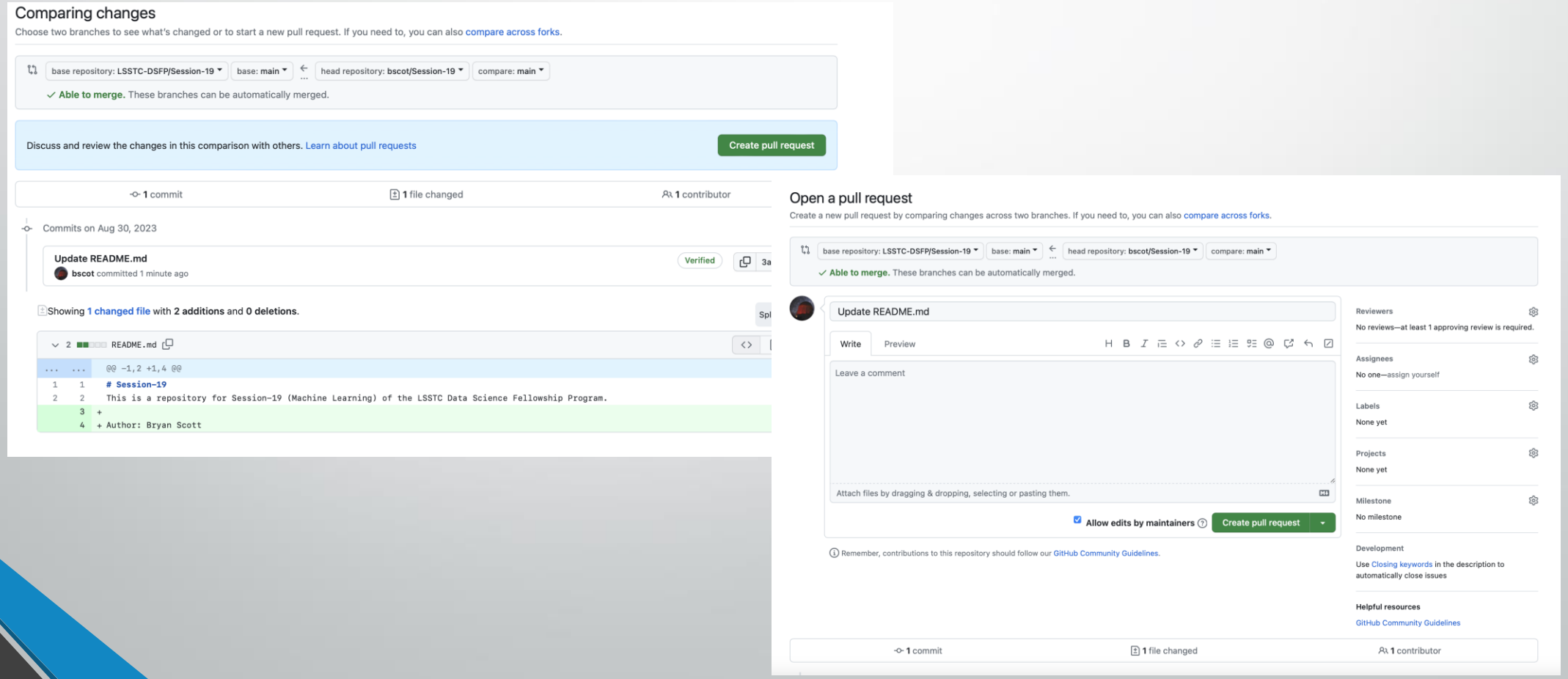
Pull Requests



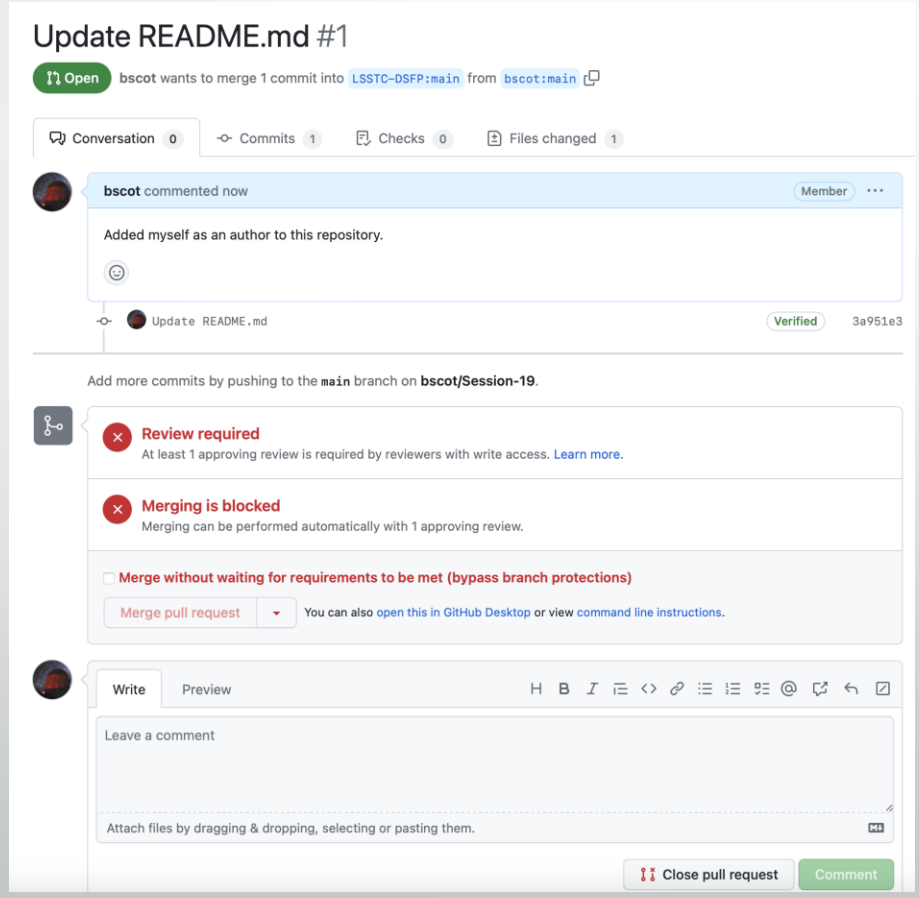
Pull Requests



Create The Pull Request



Approving the Pull Request



Git "muscle memory'

git add [filename] # adds a file so that git tracks it

git status # shows what files have been added & modified + more

git commit –m [brief description] # commits the file to your repo

git push origin main # sends your repo to the remote repository

Summary: Git and Version Control

Version control gives you a 'global undo' button to revert changes in your code. It also helps make your code more open and reproducible, which makes your (and everyone else's) science better!

We've walked through cloning and forking the DSFP Session 19 repository, as well as how to push and pull to/from your fork/branch of it. For each session, we'll ask you to repeat this.

Summary: Git and Version Control

Version control gives you a 'global undo' button to revert changes in your code. It also helps make your code more open and reproducible, which makes your (and everyone else's) science better!

We've walked through cloning and forking the DSFP Session 19 repository, as well as how to push and pull to/from your fork/branch of it. For each session, we'll ask you to repeat this.