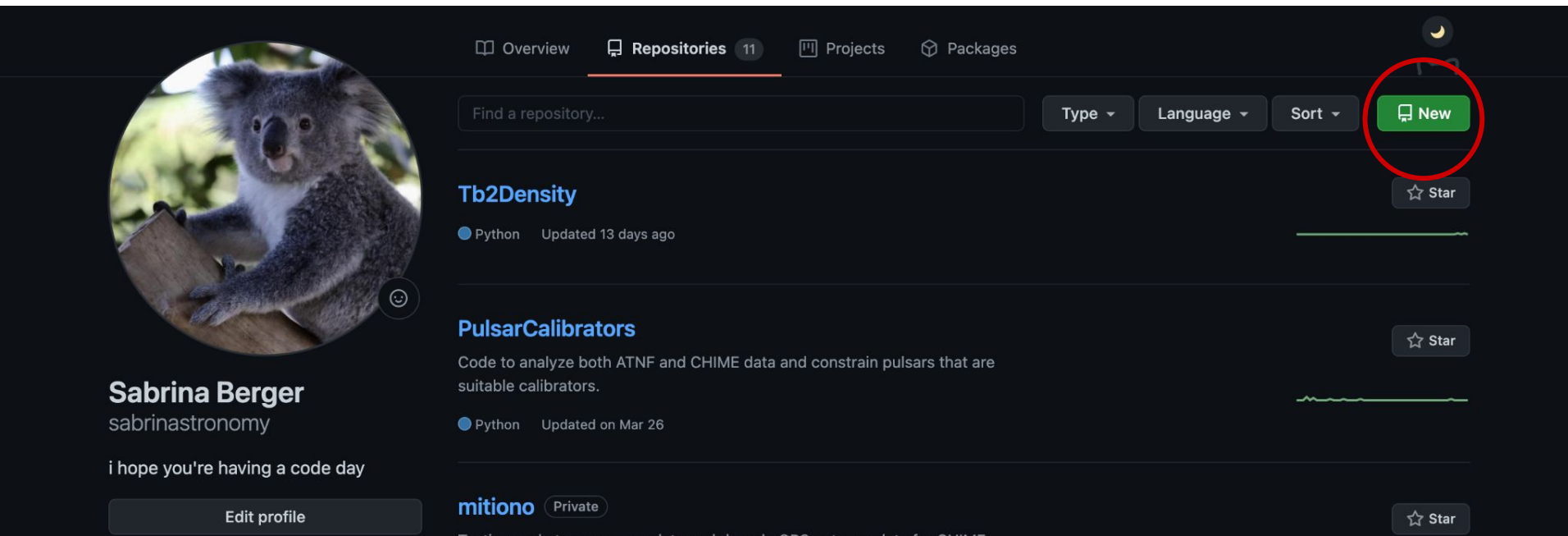


Brief Git Tutorial

Astrophysics CODE CLUB
University of Melbourne

How to make a remote git repository*

1) Click the green new button to create a new repository.



The screenshot shows the GitHub profile of Sabrina Berger (sabrinastronomy). The 'Repositories' tab is active, displaying a search bar and filters (Type, Language, Sort). A red circle highlights the 'New' button in the top right corner of the repository list. Below the search bar, two repositories are visible: 'Tb2Density' (Python, updated 13 days ago) and 'PulsarCalibrators' (Python, updated on Mar 26). Each repository has a 'Star' button. At the bottom, a repository named 'mitiono' (Private) is partially visible.

*Note that we can also create a git repository locally (*git init*), which creates a **./git** folder locally. We can then push to GitHub later

2) Follow the instructions to create a repository (don't worry about any of the extra features for now).

Create a new repository


A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository](#).

Repository template

Start your repository with a template repository's contents.

No template ▾

Owner *

 sabrinastronomy ▾

Repository name *

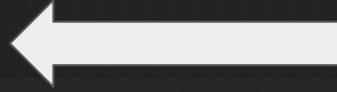
Great repository names are short and memorable. Need inspiration? How about **solid-octo-carnival?**

Description (optional)

3) Now we want to *git clone* the repository we just made on to our local machines (wherever we'd like!). This will copy over all the git history from the online project (**./git** folder!)

- **Note:** this makes a **remote branch** called **main** which is attached to the cloud, this can be a confusing detail if you accidentally make a local branch and then try and merge it into origin

```
Last login: Sun Jul 30 12:16:21 on console
(base) [ 2:52PM ] [ sabrinaberger@sabrinaastronomy:~ ]
$ git clone https://github.com/sabrinaastronomy/CodeClub.git
Cloning into 'CodeClub'...
remote: Enumerating objects: 10, done.
remote: Counting objects: 100% (10/10), done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 10 (delta 1), reused 6 (delta 0), pack-reused 0
Unpacking objects: 100% (10/10), done.
```



4) Try *cd*-ing into your new repository and trying *git status*

```
(base) [ 2:56PM ] [ sabrinaberger@sabrinaastronomy:~ ]  
$ cd CodeClub  
(base) [ 2:56PM ] [ sabrinaberger@sabrinaastronomy:~/CodeClub(main✓) ]  
$ ls  
LICENSE  README.md  Resources  
(base) [ 2:56PM ] [ sabrinaberger@sabrinaastronomy:~/CodeClub(main✓) ]  
$ git status  
On branch main  
Your branch is up to date with 'origin/main'.
```

Important basic git commands

- **git --help**
 - Gives a helpful list of git commands! Can also do `git {command} --help` for specifics on commands
- **git config --global user.name "{username}"**
- **git config --global user.email {email}**
 - **Note:** these are *git* associated names/emails, so they don't have much to do with GitHub! Just useful for identifying yourself in local git projects
- **git status**
 - Shows you current changes
- **git log**
 - Shows you the commit history for your project
- **git fetch**
 - Updates information stored in the local **./git** folder, such as new remote branches

How to make your first commit

- The most basic git workflow consists of three important steps:
 - a. Use ``git add {file}`` to “add” a new file, or to “add” changes to an already existing file
 - You can use “wildcard” operators with this! E.g. ``git add *.py`` to add .py file changes
 - b. Once you’re happy with your additions, use ``git commit -m “{useful message}”`` to add a commit with a helpful commit message explaining your changes
 - c. Finally, we need to push our changes to the cloud. We’ll do this with the command ``git push origin main``
 - Note that the *origin* here specifies the remote cloud, and *main* is the branch we’re committing to. By default, GitHub repos start with only a main branch

How to pull from remote

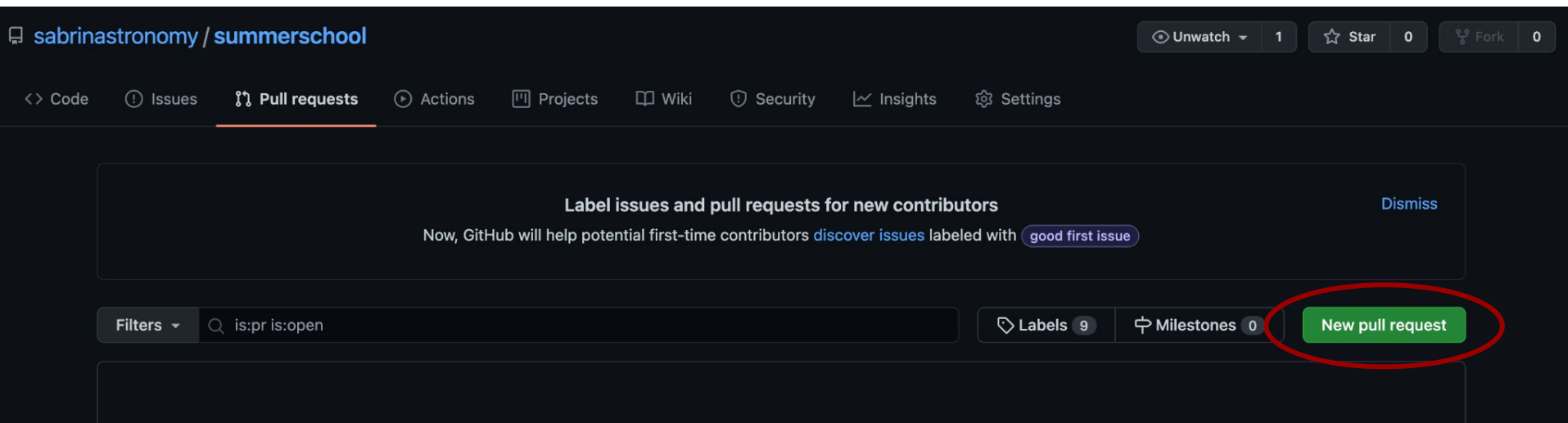
- To pull in any changes from collaborators, just use ``git pull`` in the relevant directory
 - This won't do anything for us now, since we just made this repo for ourselves... But it's VERY important to pull the most recent version of the repository before you start making changes!
 - Ideally if everyone was working on their own branches and being responsible about workflow, this wouldn't be an issue... But nobody's perfect :)

Brief Git Workflow Summary

- Step 0: PULL changes that might have been made by collaborators
- Step 1: ADD our changes
- Step 2: COMMIT changes with a message (-m)
- Step 3: PUSH changes to the cloud
- **NOTE:** all the while we can check the STATUS of our additions!

How to make a pull request

- When you're working in a collaboration, and you're ready to incorporate your changes into the master branch, you can make a pull request!



- Pull requests are a super useful way of keeping major code changes organized
- Rule of thumb: **master** branch should ALWAYS be deployable
 - This is why pull requests exist: typically if you're a part of a collaboration, there will be other people working on the code base with you. Usually there'll be one/a few people who manage most of a given repository, and making a pull request allows you to give them a chance to view your code, review it, suggest changes, and then finally accept the merge into master once it's deemed ready
- Workflow goes something like
 - Propose a new feature
 - Checkout a new branch to start working on your feature (make sure nobody interferes with your work so you don't get merge conflicts)
 - Keep pushing changes to your branch until things are stable/finished, then make a pull request

How to go back to an old commit when your new code breaks

- You can get a log of all previous commits with
 - *git log*
- This should return your previous commits along with their corresponding hash, e.g.,

```
(base) sabrinaberger@sabrinaastronomy summerschool % git log
commit 5d29d753054e787005f0202364e286d1211032db (HEAD)
```

- You can revert to a previous commit with
 - *git checkout <commit hash>* (where commit hash is that long green string after the commit above)

Resources

- [Amazing git branching tutorial](#)
- [How to install git on any OS](#)
- [A nice ELI5 git series](#)
- ["What is git" from Atlassian](#)
- [Basic git tutorial](#)
- [An in-depth summary of remote branches](#)
- [Tutorial on how to deal with merge conflicts](#)