



Working with Git and GitHub

Working with Git and GitHub

Instructor: Brandon Reyes

- Website: www.rc.colorado.edu
- Helpdesk: rc-help@colorado.edu

My Goal

- Convince you that basic Git/GitHub fluency is:
 - Easy
 - Practical
 - An extremely important tool in your tool belt!



Learning Goals

- Understand the basics of version control
- Differences between Git, GitHub
- Basic Git fluency
- How to collaborate with Git



Outline

- Brief overview of Git and GitHub
 - What is version control?
- Creating your own repository locally
- Pushing local changes to GitHub
- Collaboration

Have you set up Git/GitHub?

This is meant to be a mostly hands on tutorial. If you haven't yet, you may still be able to get everything set up in time using the link:

[https://github.com/ResearchComputing/Summer_Camp_2023/blob/main/D
ay_Three/Using_git/README.md](https://github.com/ResearchComputing/Summer_Camp_2023/blob/main/Day_Three/Using_git/README.md)

Git vs GitHub

- Git: version control system
 - the actual software
- GitHub: Cloud-based storage website
 - Hosts repositories (“repos”)
 - Provides a GUI for many Git features
 - Allows for easy collaboration
 - Issues, pull requests
 - GitHub basic is free (up to 5GB of storage)
 - GitHub Enterprise (free for CU affiliates)
 - <https://oit.colorado.edu/services/business-services/github-enterprise>



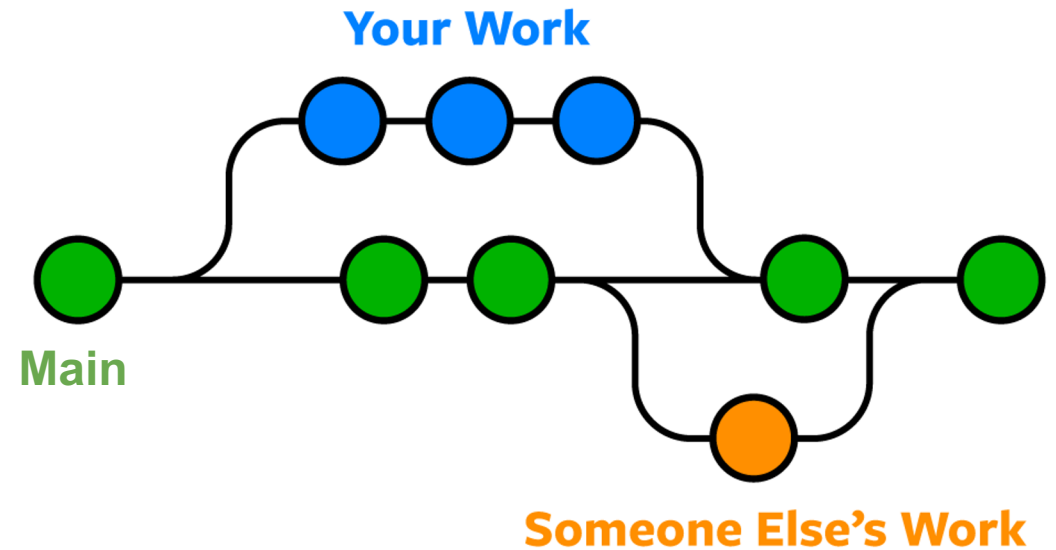
What is version control?

Version control is the practice of tracking and managing changes to files.

- Why do I need it?
 - Revert to various states of files
 - You can think of this as a backup
 - Allows you to modify items without harming the original copy
 - Not limited to code
 - documents, images, etc...

Additional benefits of version control

- Using version control provides
 - Clear tracking of the repo's history
 - Management and view of different branches (work)
 - Collaboration through merging of branches



Images: nobledesktop.com

Different Version Control Systems

- Subversion (svn)
 - Mercurial
 - CVS
 - etx
-
- We're going to stick to Git
 - industry standard
 - widely known
 - most resources



Images from Wikipedia

Getting Started with Git (local)

Setting Git up locally

Many systems have Git installed; however, you may need to download it on your local machine

- See <https://git-scm.com/book/en/v2/Getting-Started-Installing-Git> for more information

Today we are going to stick with using Git on a login node

Logging into RC via a Terminal

To login to an RC login node:

```
$ ssh <username>@login.rc.colorado.edu
```

- Supply your IdentiKey password and your Duo app will alert you to confirm the login
- Confirm Git has been configured (by you using the README)

```
$ git config --list
```

Hands on tutorial

Goal: Create a simple project that contains a markdown file

First let's create a new directory for our project:

```
$ cd /projects/$USER
```

```
$ mkdir git-tutorial
```

```
$ cd git-tutorial
```

Git Repository (Repo)

A Git repository tracks and saves the history of all changes made.

- All of this information is stored in “.git”, which is the repository folder

We can make a directory (folder) a Git repo using “git init”

Git Init

In your “git-tutorial” directory run

```
$ git init
```

- Git creates the "hidden" directory called “.git”

```
$ ls -a
```

- Your directory is now a repo!
 - Git is now ready to be used
 - Allows us to tell Git what items to watch

Create the main branch

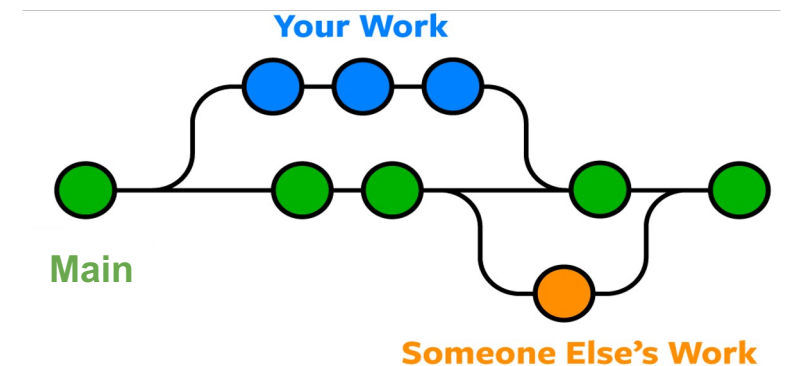
Now that we have a repo, we can create branches. Branches are a version of the repository.

- It is customary to name the primary branch “main”
- This can be done as follows (after an init)

```
$ git checkout -b main
```

- You can switch between branches

```
$ git checkout <branch-name>
```



Let's add a file!

It is customary to add a README.md

- Description of repo and any helpful information

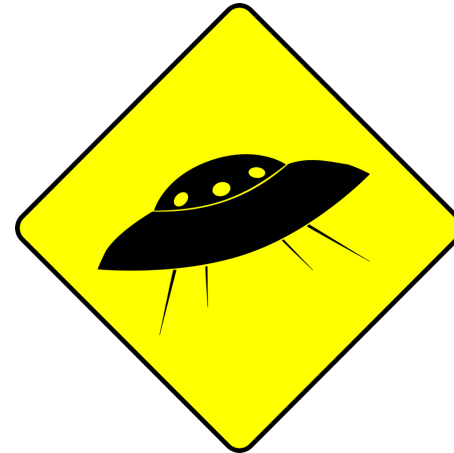
To add a README.md, in “git-tutorial” create and edit the file using nano (or an editor of your choice)

```
$ nano README.md
```

- Add anything you would like!
- Be sure to save the file when you exit.



Git does not know about README.md yet!!



Areas of Git Workflow

Working Area

- Items that you are currently working on
- Are not tracked by Git!
- Exists locally

Staging Area

- When Git starts tracking and saving your work
- Exists locally
- Items are added to this area by using “git add”

Snapshot Area

- All staged items are captured
- Version of the repo
- Exists locally
- Items are added to this area by using “git commit”

GitHub

- Exists locally and on GitHub!
- Items are added to this area using “git push”

Git Status

The git status command **displays the state of the working and staging area.**

Let's see what area README.md is in

```
$ git status
```

- We see it is an untracked file, so it is in the working area

What if you don't want Git to track something?

.gitignore

We can add a file named “.gitignore” to our repo

- Specifies what items (files, directories, etc.) should never be tracked

Let's create a file to ignore!

```
$ echo "Super secret stuff" > confidential_data.txt
```

Add “.gitignore” to “git-tutorial” and put “confidential_data.txt” in it

```
$ echo confidential_data.txt > .gitignore
```

Let's add our files to the staging area now!

Git Add

The git add command **adds a change in the working area to the staging area**

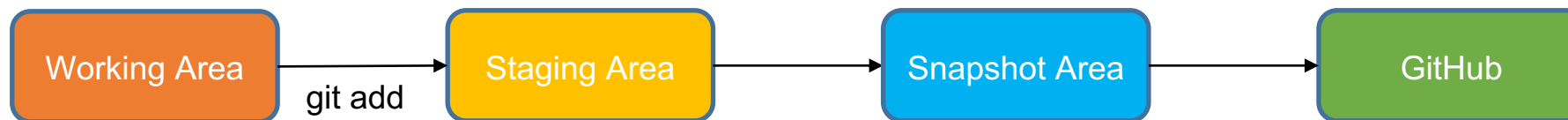
Let's add our README.md to the staging area

```
$ git add README.md
```

or add everything in the current directory

```
$ git add .
```

- Anytime a change is made, you need to do a git add (to track them)



Git Commit

The git commit command **captures a snapshot of all staged items**

- Commits can be thought of as a version of the repo
- Commits should be accompanied with a brief message

Let's commit our staged item!

```
$ git commit -m 'Create repo, add README.md, add .gitignore'
```

```
$ git status
```



Common practice – add, commit

- `git add`
 - Can be performed as much as you want
 - Doesn't need to be done after every change
- `git commit`
 - Always include a comment!!
 - Bundle common staged items together
 - Try not to put too many things in a commit

Git Log

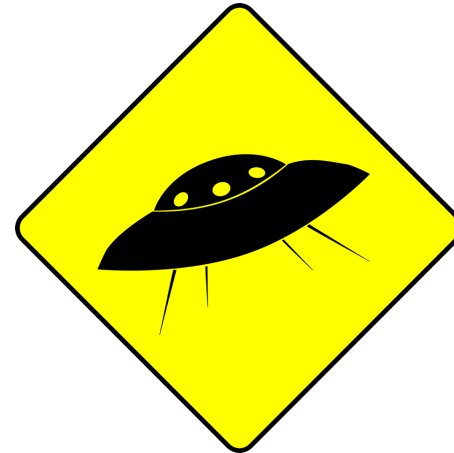
The command git log **lists the commits made in that repository**

- Lists the most recent commits first

\$ git log



All changes and files are only locally stored right now!



To GitHub we go!

GitHub

When you first create a repo locally, you will need to setup a new repository on GitHub too

- Go to <https://github.com>
- Sign in
- Click on “Create New Repository” or just “New”

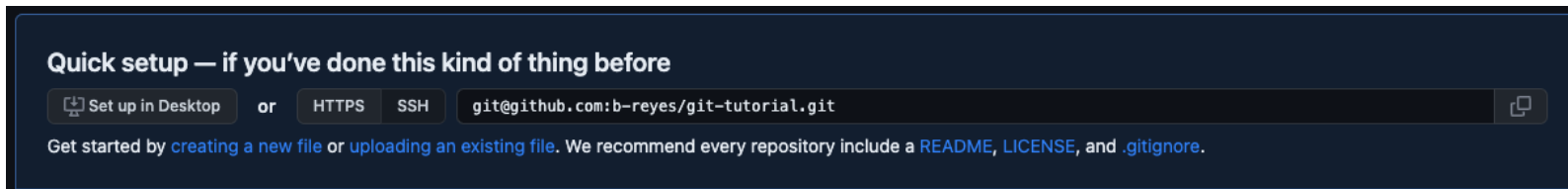
Recent Repositories

 New

Find a repository...

Create Repo in GitHub

- Name your repo, I chose “git-tutorial”
- Don’t add a README or a .gitignore
- Click “Create repository”
- We have set everything up in the previous slides, we only need to copy the ssh link!



Linking local repo to GitHub repo

Git Remote

- Used to identify the remote (e.g. GitHub) repos are linked to your local repo
- Used to link remote repos to your local repo

To view currently linked remote repos:

```
$ git remote -v
```

To link our remote repository:

```
$ git remote add origin git@github.com:<user>/test-tutorial.git
```

Sending local changes to GitHub

Git Push

Uploads local repository content to a remote repository

- Pushing is how you transfer commits from your local repo to a remote repo

```
$ git push <name of remote repo> <branch>
```

```
$ git push origin main
```



GitHub

- Go back to GitHub and refresh your page
 - should see the files we have added (and not the ones we've ignored)
- Some cool features!
 - look at our commits
 - directly edit/commit in the browser
- Let's do that! Let's something and commit it on GitHub
 - But now our remote repo is one commit ahead of our local one...

Git Fetch & Merge

- Git fetch retrieves the changes from the remote repo

```
$ git fetch
```

- Git merge combines two branches

```
$ git merge origin/main
```

There's an easier way!

Git Pull

Git pull combines the fetch and merge commands

```
$ git pull <name of remote repo> <branch>
```

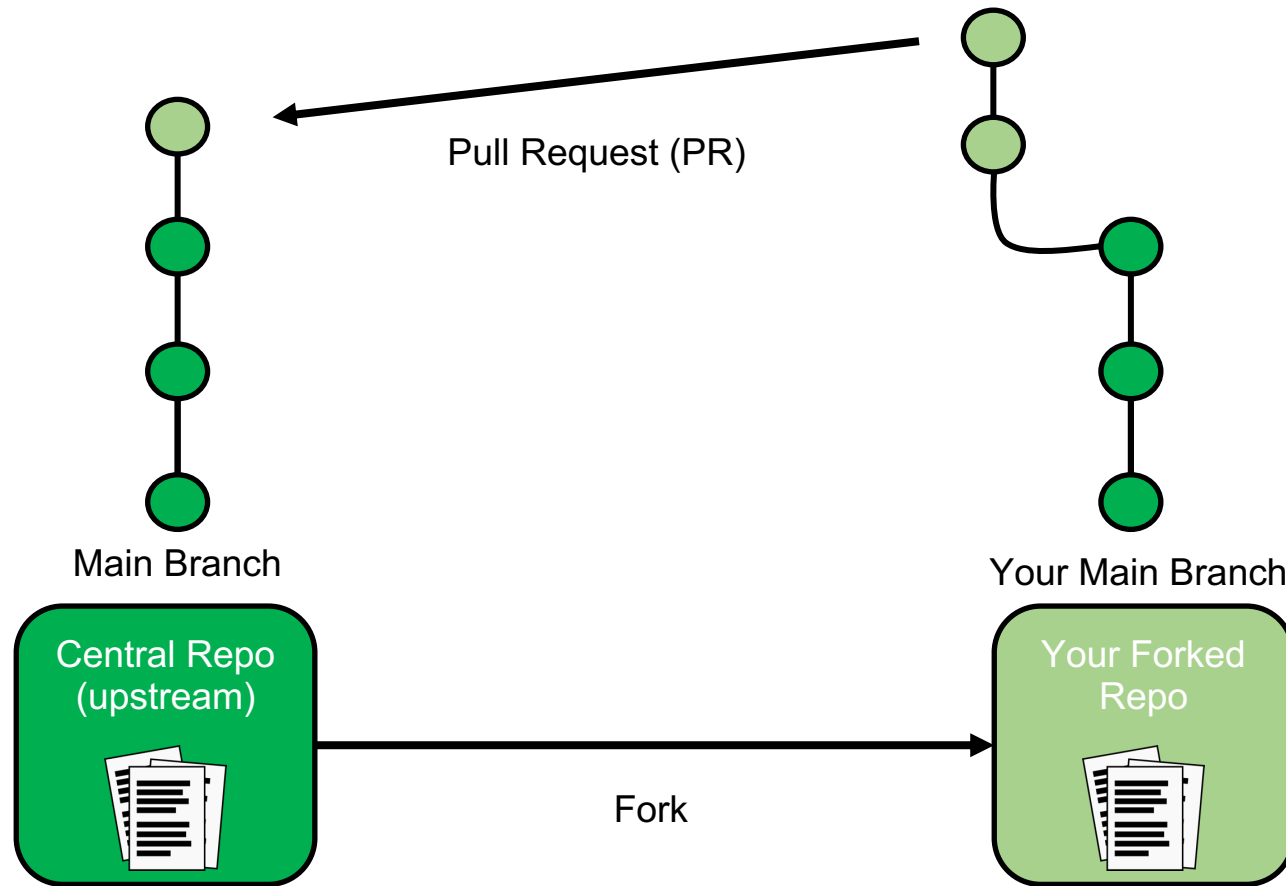
```
$ git pull origin main
```

IMPORTANT!

- Make sure no commits have been done on local branch
- It is fine to have staged items (git add)
- ALWAYS do git pull before any commits!

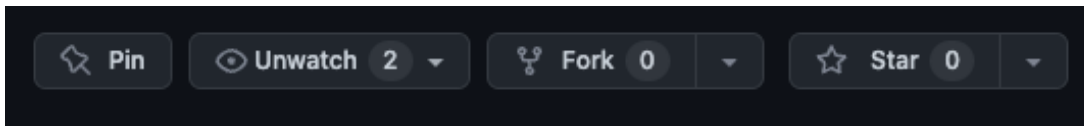
Collaboration

GitHub Forks



GitHub Forks

- Improves collaboration
 - Don't have to worry about disturbing the upstream repo
 - Improves transparency through pull requests
- Go ahead and Fork my repo:
 - Go to <https://github.com/b-reyes/git-tutorial>
 - Click “Fork” button
 - Click “Create fork”
- Creates your own version of my repo under your GitHub



Git Clone

- Git clone makes a clone (or copy) of a remote repo in a new directory, at another location.

```
$ git clone <url> <optional new name>
```

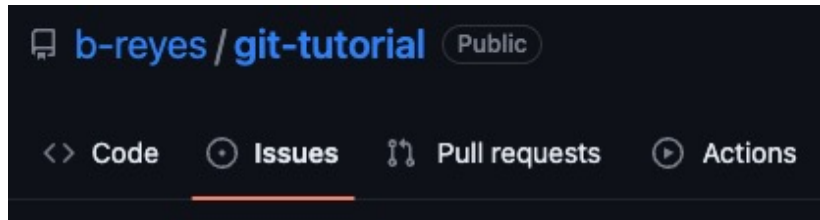
- Easy way to grab third-party code, or pre-existing code you might need to work on

```
$ cd /projects/$USER
```

```
$ git clone git@github.com:<user>/git-tutorial.git
```

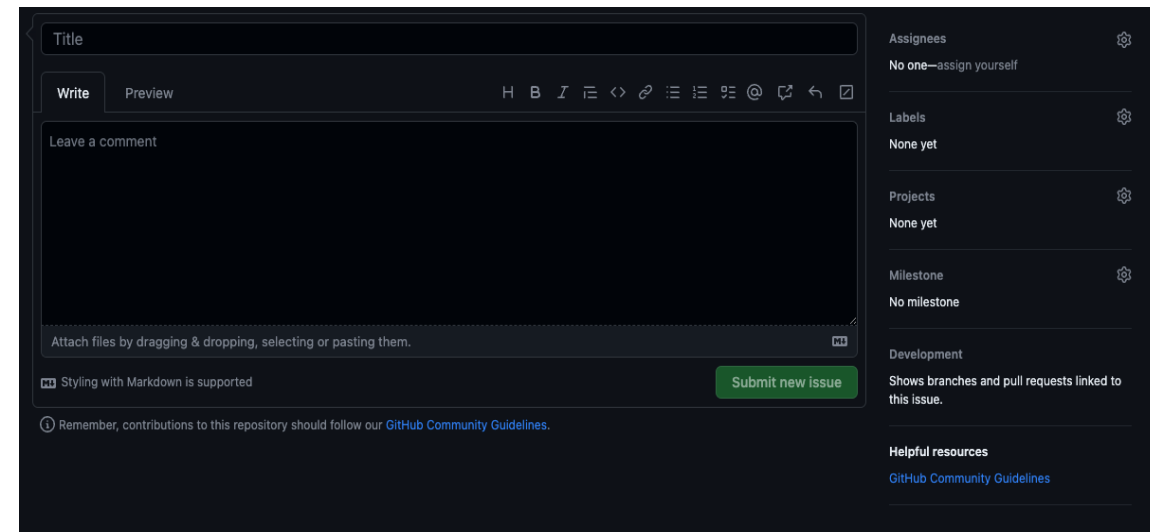
GitHub Issues

- Allows you to discuss the project
- Point out issues, request features, ask for help
- Useful place to see past user discussion



GitHub Issues

- Include as much detail as possible
 - Version of software
 - Operating system
- Provide a simple minimal example, if possible
- If a feature request
 - Outline possible implementation
 - Highlight its value



The image shows the GitHub 'New Issue' form interface. It features a 'Title' input field at the top. Below it are 'Write' and 'Preview' tabs, with a rich text editor toolbar containing icons for bold, italic, link, code, and other formatting options. The main body of the form is a large text area with the placeholder text 'Leave a comment'. Below the text area is a file upload section with the text 'Attach files by dragging & dropping, selecting or pasting them.' and a 'Submit new issue' button. On the right side, there are sections for 'Assignees' (showing 'No one—assign yourself'), 'Labels' (showing 'None yet'), 'Projects' (showing 'None yet'), and 'Milestone' (showing 'No milestone'). At the bottom right, there is a 'Development' section with a note about linked branches and pull requests, and a 'Helpful resources' section with a link to 'GitHub Community Guidelines'.

Pull Requests (PRs)

- A request that an upstream repo pull your branch into their branch
- Starting a PR does not automatically merge changes
 - Notifies maintainers of upstream repo
 - Allows maintainers to review your changes
 - Discussion of changes
 - Requested additional changes
- Maintainers of upstream repo merge in the changes

PR steps

1. Fork upstream repo
2. Clone the forked repo
3. Connect forked and cloned repo to upstream repo

```
$ git remote add upstream git@github.com:b-reyes/git-tutorial.git
```

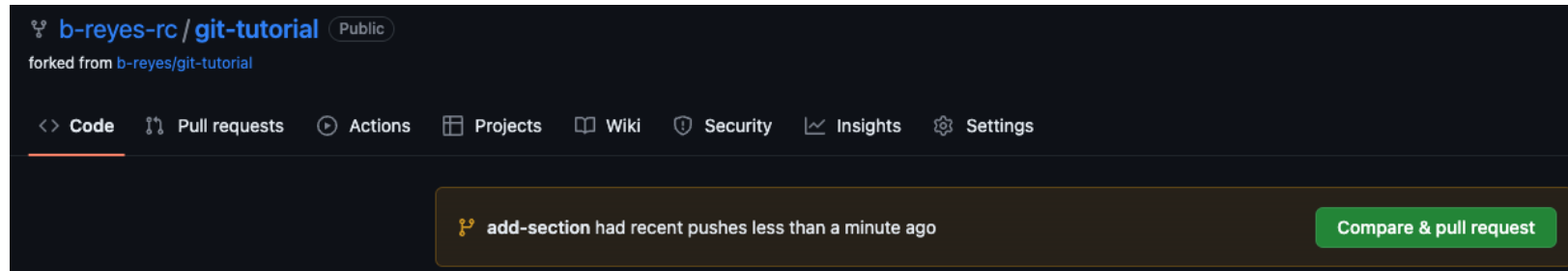
4. Create a new branch specific to your change

```
$ git checkout -b <new-branch> <branch-to-copy>
```

5. Make your changes on this branch
6. Perform a git add, commit, and push to origin
7. Create a PR from GitHub

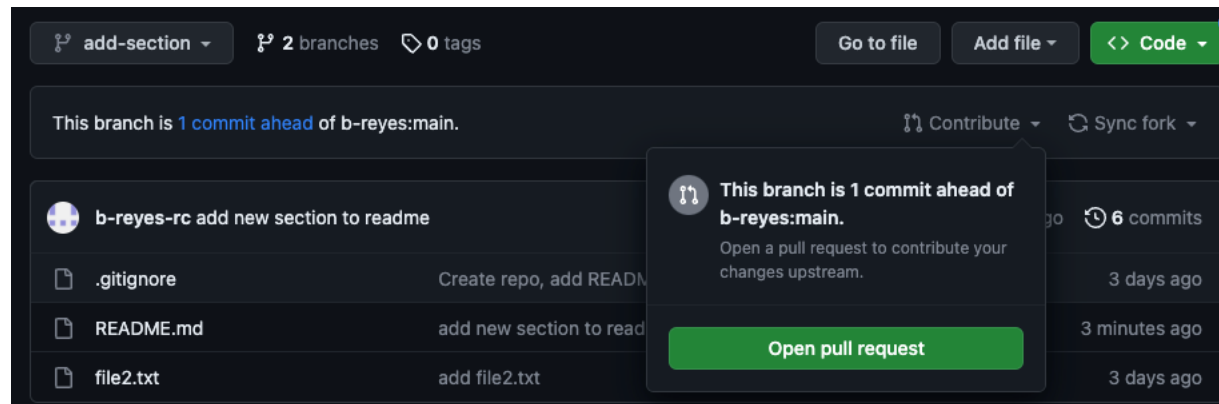
Creating a PR

- After you push your changes to the forked repo, you can click the pop-up “Compare & pull request” on GitHub
 - Will disappear after some time



Creating a PR

- After you push your changes to the forked repo
 1. Switch to your new branch
 2. Click the drop-down arrow next to “Contribute”
 3. Click “Open pull request”
- Will NOT disappear!



Creating a PR


Upstream repo and branch we want to merge into











Our forked repo and branch with our changes

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: b-reyes/git-tutorial base: main head repository: b-reyes-rc/git-tutorial compare: add-section

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

 Add hobby section to README

Write Preview H B I          

Leave a comment

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

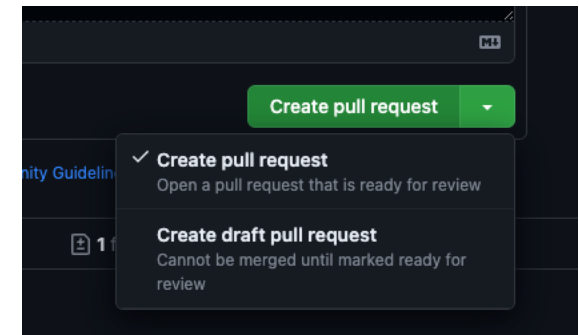
☒ Allow edits by maintainers

Create pull request

Remember, contributions to this repository should follow our [GitHub Community Guidelines](#).

Pull Requests – Best Practice

- Create a new feature branch of forked repo
- When submitting a PR
 - Provide a short descriptive title
 - In comment section
 - Link to any current issue
 - Describe what the PR does and reasons for it
- Draft pull requests
 - PR is a work in progress
 - Can be used for discussion



Merging

- When doing a “git pull” you are merging in changes
- This process can be done manually
- When collaborating, multiple individuals can be working on the same item
 - Conflicts can happen!
- One needs to manually resolve conflicts
- Fantastic tutorial on merging:

<https://www.atlassian.com/git/tutorials/using-branches/git-merge>

Help! I'm stuck, where do I go?

- **Documentation:** curc.readthedocs.io/
- **Trainings with Center for Research Data and Digital Scholarship (CRDDS):**
<https://www.colorado.edu/crdds/>
- **Software Carpentries tutorial:** <https://swcarpentry.github.io/git-novice/index.html>
- **GitHub Student Developer Pack:** <https://education.github.com/pack>
- **Helpdesk:** rc-help@colorado.edu