Why do I need Git/GitHub??

Version Control with Git/GitHub

Instructor: Gerardo Hidalgo-Cuellar

- Website: www.rc.colorado.edu
- Helpdesk: rc-help@colorado.edu
- Slides: https://github.com/ResearchComputing/New_User_Seminar

- Survey: http://tinyurl.com/curc-survey18

RMACC Cyber Infrastructure Portal

- https://ask.cyberinfrastructure.org/c/rmacc/65
- This forum provides opportunity for RMACC members to converse amongst themselves and with the larger, global research computing community.
- The “go to” general Q&A platform for the global research computing community - researchers, facilitators, research software engineers, CI engineers, sys admins and others.

My Goal

- Convince you that Git/GitHub fluency is:
  - Easy!
  - Practical (as a researcher!)
  - An important (if not *the* most important!) tool in your tool belt*!

Learning Goals

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

*I may be biased

images: wikipedia

Outline

- Setting started with Git (Locally)
- Getting started with GitHub (Remote)
- Collaboration
- Advanced Topics (if time allows)

GitHub Account Check

- Create a free account at: https://github.com/signup
- Necessary for the GitHub portion of the class

Getting Started with Git (local)

You may know about GitHub!

- Thumbs up if you have visited a GitHub project before?
- What kinds of things have you used GitHub for?
- Git and GitHub are different, and we'll get into that!

What is version control?

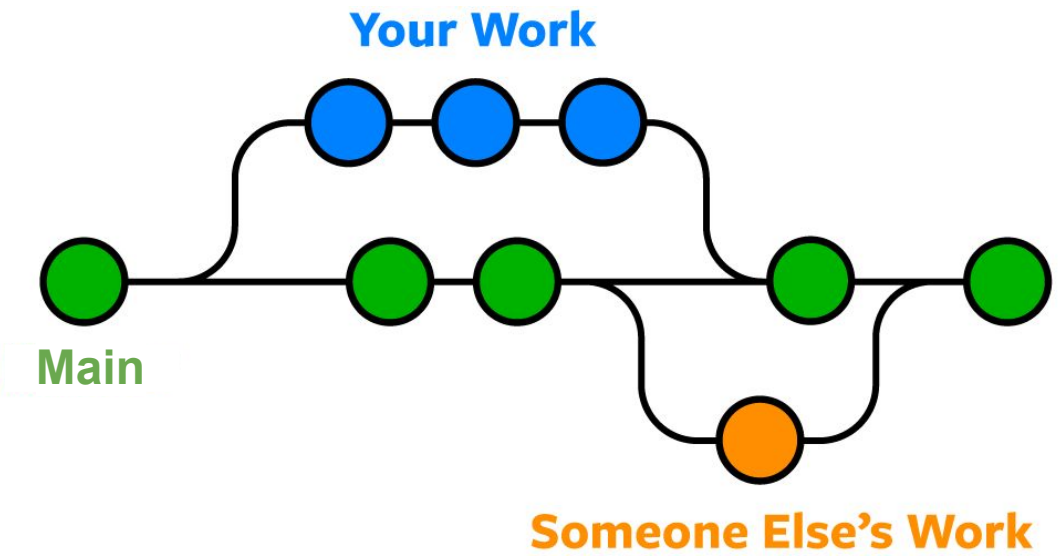
- Why do I need version control as a researcher? Isn't it for developers?

- NO! Version control systems let you track changes you make to your files over time.
  - Revert to various states of files
  - Test things out without harming originals
  - Not limited to source code
    - test files, images, etc…

What is version control?

- Think Google Drive with jet engines!

- You have direct control over:
  - history
  - paths (alternate universe)
  - merging new changes into projects



Images: wikicommons, 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

Git vs GitHub

- Git: version control system (installed locally)
  - the actual software

- GitHub: Cloud-based storage (repository, or “repo”) site
  - a common/shared area to host projects
  - many Git features as a web GUI

Setting Git up locally

- Many systems already have Git installed
  - check in a terminal with: `git --version`
  - if you don't have it, you'll need to install it a the main Git website: https://git-scm.com/downloads

- You can follow along on your computer if you have git installed, or log into the RC system which has Git already installed.

Logging into RC via 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

---

If you're using a tutorial account (we provide password):

```
$ ssh <tutorial_user>@tlogin1.rc.colorado.edu
```

Configure Git

- configuration variables (like env) for Git

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

- Let’s set up our name and emails

```
$ git config --global user.name “Jane Smith”
$ git config --global user.email “jane@email.com”
```

Hands on tutorial

- We are going to create a simple HPC project that run a simple “hello world” python program with 5 different cores
  - Following the loadbalancer example at: https://curc.readthedocs.io/en/latest/software/loadbalancer.html

- What do we need?
  - hello world python script
  - loadbalancer command file
  - slurm script

Project Directory

- First lets create a new directory for our project:

```
$ cd /projects/$USER    # or wherever
$ mkdir git-tutorial
$ cd git-tutotial
```

Git Repository (Repo)

- A Git repo is a set of files that keep track of changes within a directory (folder)
- We need to tell Git to actually set this up

Create a file

- Now let's create the first file for our project, the python "hello_world.py" script.

- Keep it simple for now:
  - use your favorite text editor (vi/vim, nano, emacs) to create it:

    ```
    $ vim hello_world.py
    ```

  - Enter the following line into the file, save and exit

    ```
    print("Hello, World!")
    ```

Git Init

- Git init will initialize a directory as a Git project:

```
$ git init
```

  This will tell Git to get ready to start watching your files for every change that occurs.

- What's actually happening here?
  - The Git program has created a "hidden" directory called .git

```
$ ls -a
```

  - This is where the “magic” happens!
  - Project history and other Git configs get stored here
  - Can also remote this directory to remove Git from project

An aside: Main vs. Master

- Default is changing from Master -> Main as default branch or “trunk”
  - shorter
  - translates better into other languages
  - inclusive and recognizes issues with “master” language
  - now default

- We’ll talk about branches later, but it’s easiest at this point to rename your default branch with:

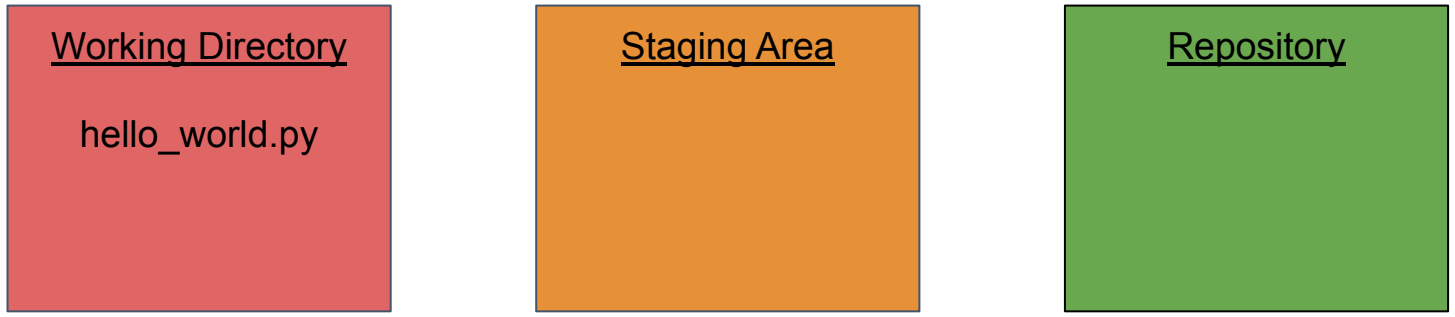
```
$ git branch -M main
```

Git Status

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

```
$ git status
```

- It lets you see which changes have been staged, which haven't, and which files aren't being tracked by Git.



Git Ignore

- You may have some files that you don't want tracked
  - secret keys (passwords, API tokens, etc)
  - build files
  - data sets
- Create a ignore.txt file

```
$ echo "ignore this file!" > ignore.txt
```

- Create a .gitignore file

```
$ vim .gitignore
```

- list any files/directories you don't want tracked:

```
ignore.txt
```

Git Ignore (RC use case)

- In you `.gitignore` you can choose to ignore output files:

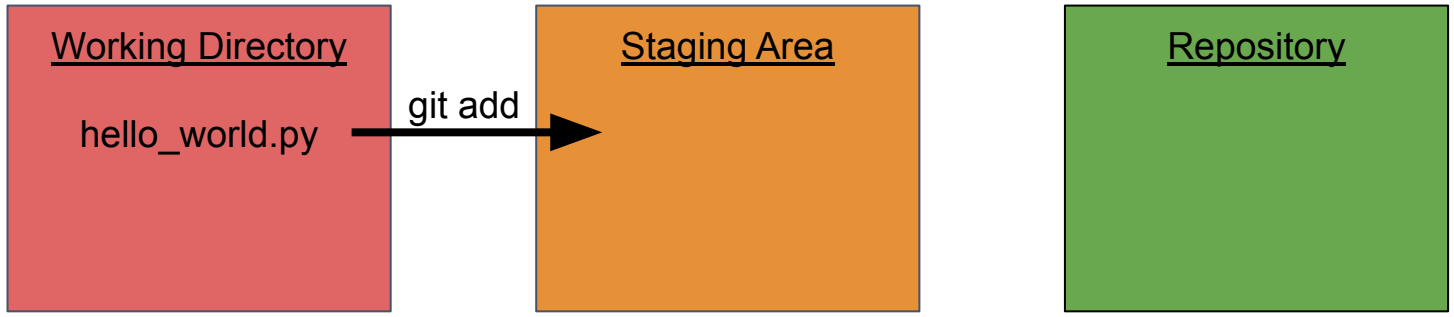
```
*out     # globbing, will get all files that end with “out”
```

Git Add

- The git add command **adds a change in the working directory to the staging area** (getting the “picture” ready for a snapshot)
- It tells Git that you want to include updates to a particular file.

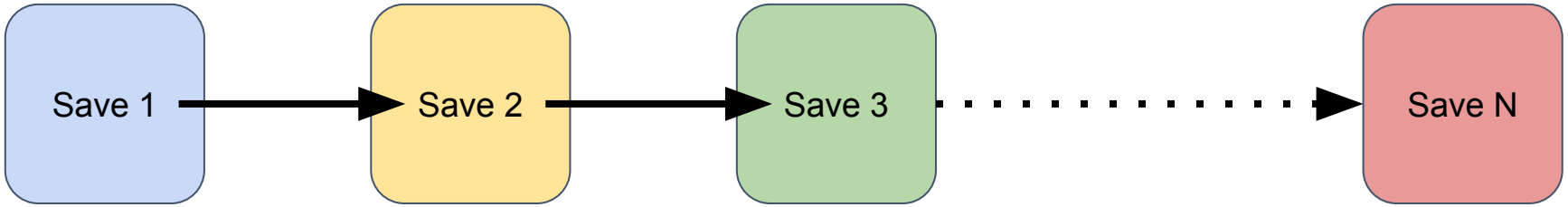
```
$ git add hello_world.py   # “git add .” to add all files
$ git status
```

**git add doesn't affect the repository - changes are not actually recorded until you run git commit



Your Git timeline

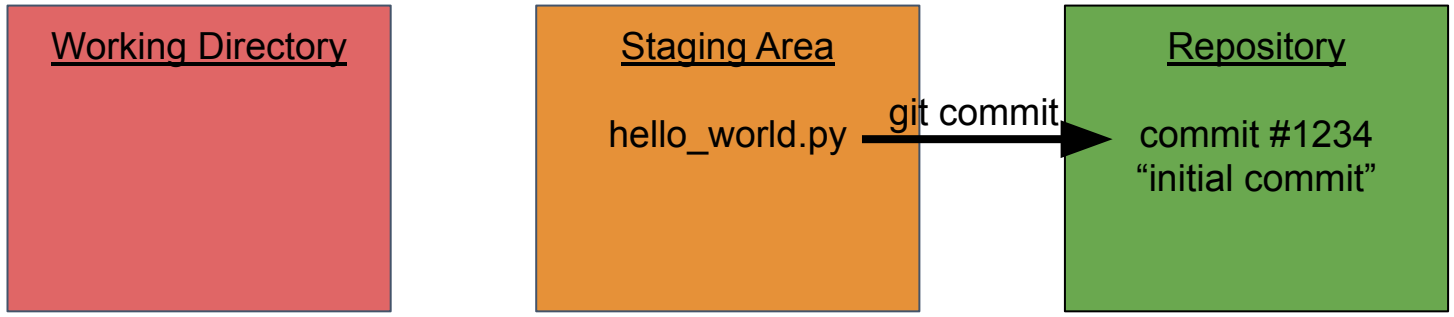
- Git commits are like savepoints or snapshots of your project



Git Commit

- The git commit command **captures a snapshot of the project's currently staged changes.**
- Committed snapshots can be thought of as “safe” versions of a project.
- Commits are logged with a brief message of what was changed

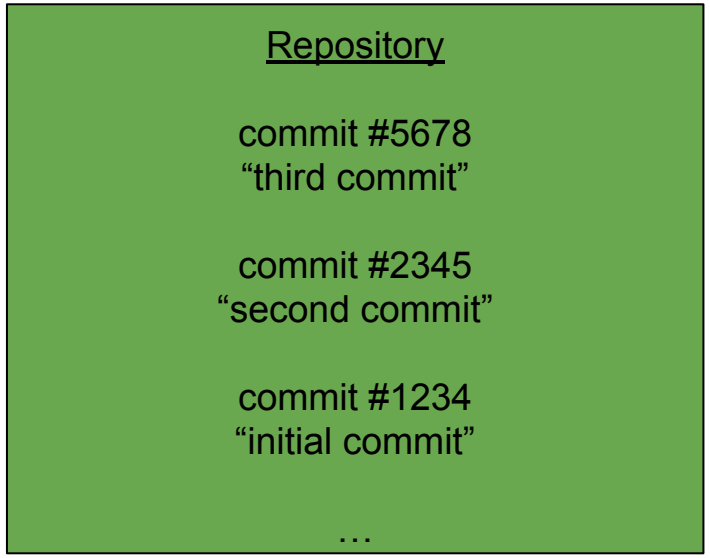
```
$ git commit -m “initial commit”
$ git status        # clean working directory
```



Git Log

- git log **lists the commits made in that repository** in reverse chronological order; the most recent commits show up first

  $ git log



Congrats! You now know Git!

- At least the basics

- Exercise: Update the “hello-world.py” file
  - The parallel `hello_world.py` will need the following changes
  - change, add, and commit them!

```
import sys
print “Hello World from process: ”, sys.argv[1]
typo! #on purpose…
```

Getting Started with GitHub (remote)

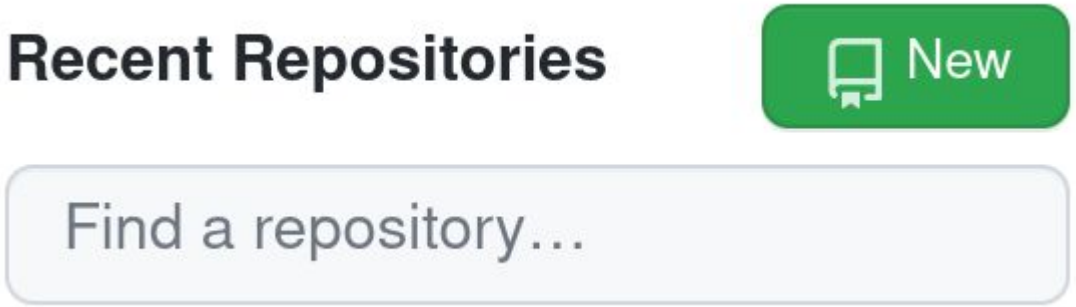
GitHub

- GitHub: Cloud-based storage (repository, or “repo”) site
  - a common/shared area to host projects
  - many Git features as a web GUI

- We’re going to demonstrate how to work with remote repositories using GitHub

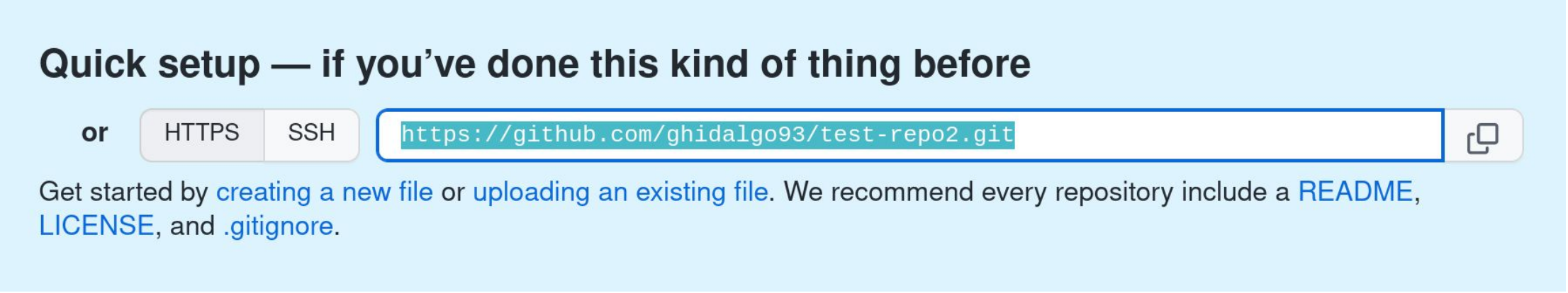
GitHub

- Go to: https://github.com
- Sign in (or create an account)
- Click on “Create New Repository” or just “New”



Create Repo in GitHub

- Create a new repo
- Call it whatever you would like
- Ignore directions for you, just copy the link
  - e.g. https://github.com/<user>/test-repo.git



Git Remote

- Git remote tells you **which remote repositories you have linked to your local project**.

```
$ git remote      # should return nothing
```

- To link our remote repository (accepts 2 values):

```
$ git remote add <name of remote repo> <url>
$ git remote add origin https://github.com/<user>/test-repo.git
```

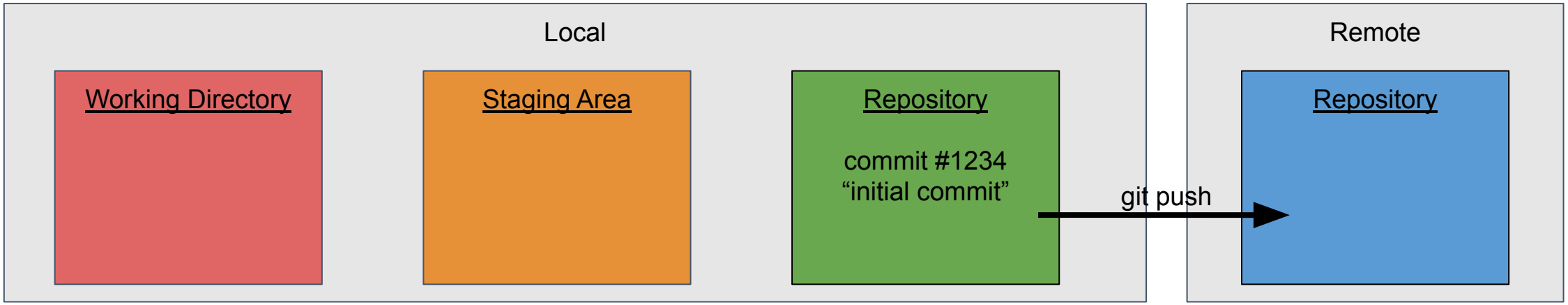
- View remote again

```
$ git remote
$ git remote -v    # view url as well
```

Git Push

- Sync up local code with remote GitHub repo!
- Git push **uploads a local repositories content to a remote repository**. Pushing is how you transfer commits from your local repository to a remote repo

```
$ git push <name of remote repo> <branch> # optional -u flag
$ git push origin main -u
```



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 fix the typo and commit it
  - 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
```

- But, there's an easier way!

Git Pull

- Git pull combines the fetch and merge commands
- **Must have clear working directory!**

```
$ git pull origin main
$ git pull    # because we used the -u flag earlier!
```

Git Clone

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

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

- Next, let's clone the lesson's repo to a new space (yes, this lesson was made with Git and hosted by GitHub!) and get the 2 other files for our project

```
$ cd /projects/$USER
$ git clone https://github.com/ResearchComputing/HPC_software_dev_course
$ cd HPC_software_dev_course
$ cp git/files/lb_cmd_file path/to/your/git-tutorial/repo
$ cp git/files/run_hello.sh path/to/your/git-tutorial/repo
```

Update your project! (practice)

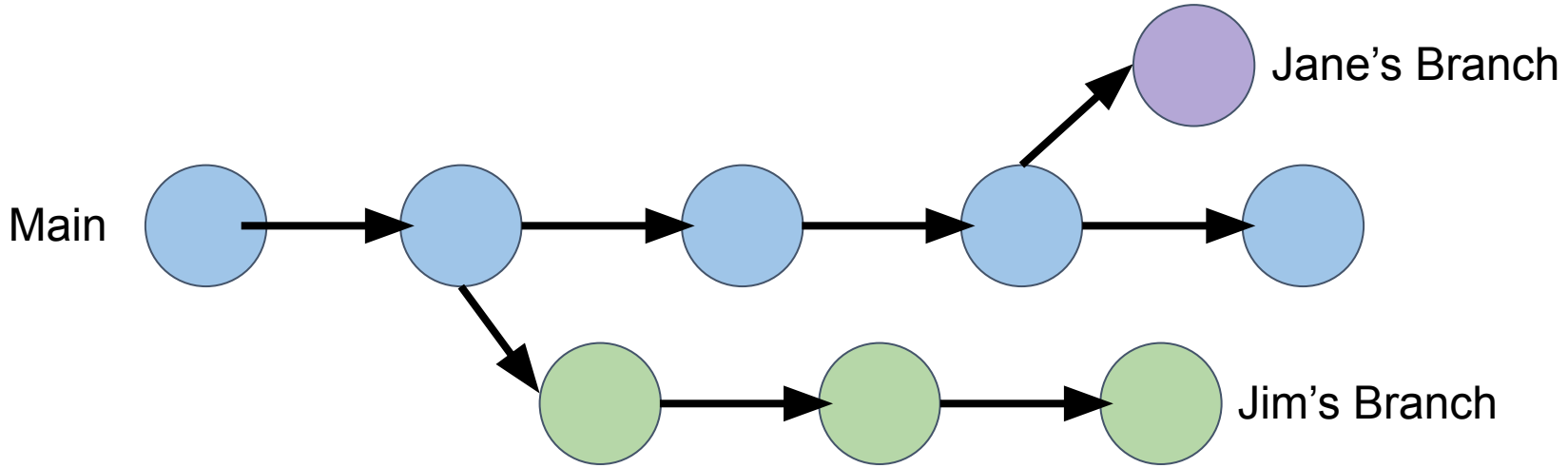
- Once you have all 3 files in your local git-tutorial repo, go ahead and Add + Commit them

- Then push up to your GitHub repo and ensure they are there!

Collaborate

Git Branch (alternate universe!)

- Allows you to collaborate with other team/lab members without getting in each other's way
- Or just do testing without messing up your working files!



Git Branch

- Git branch will allow you to create and list new branches

```
$ git branch     # list branches
```

- Create new branch

```
$ git branch coolbranch
$ git branch     # still on main branch!
```

- Delete branch

```
$ git branch -d <branch>     # safer
$ git branch -D <branch>     # forces
```

Git Checkout

- Git checkout moves us to branch

```
$ git checkout coolbranch
```

- Try to change some code, add, and commit it to new branch!
- Switch back to main branch

```
$ git checkout main
```

- The code disappears! Can go back to it with

```
$ git checkout coolbranch
```

- Create new branch *and* checkout into it (shortcut)

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

Pull Request

- Pull requests **let you tell others about changes you've pushed to a branch in a repository on GitHub**.

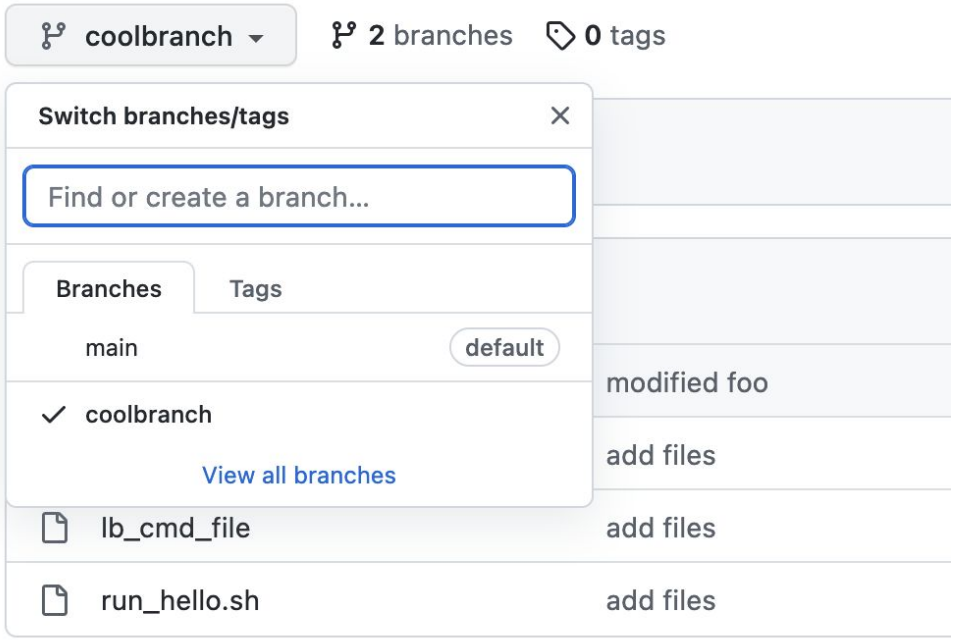
- Once a pull request is opened, you can:
  - discuss and review the potential changes with collaborators
  - add follow-up commits
  - all before your changes are merged into the base branch

- Let’s try it, on your new branch make a change, add, commit:

```
$ git push origin <branch> -u
```

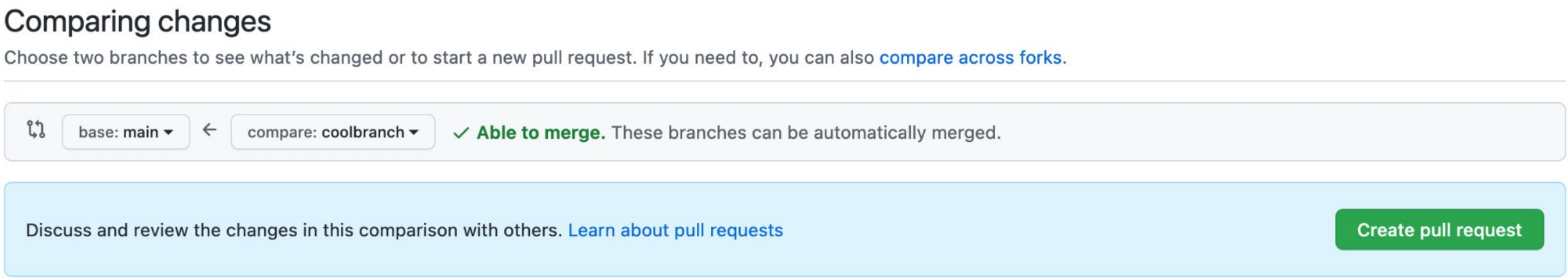
Pull Request

- Go back to the GitHub Repo
- Choose the new branch (coolbranch)!
- Should be able to see new changes



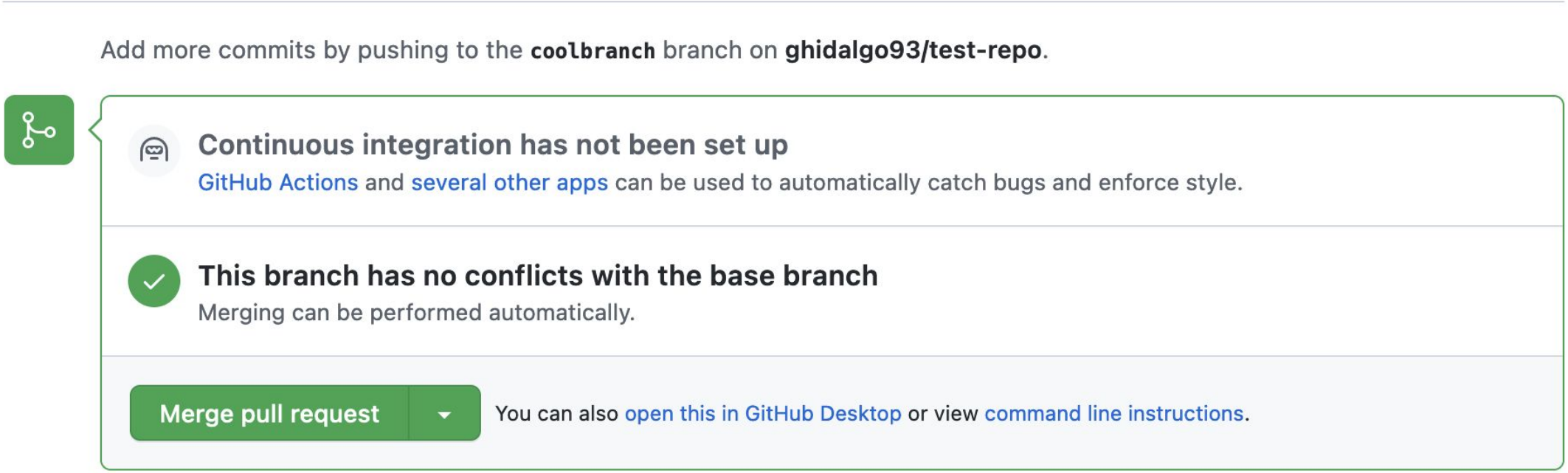
Pull Request

- Figure out which branch you want to pull in/merge
- Create pull request
- You can see all of the changes below



Pull Request

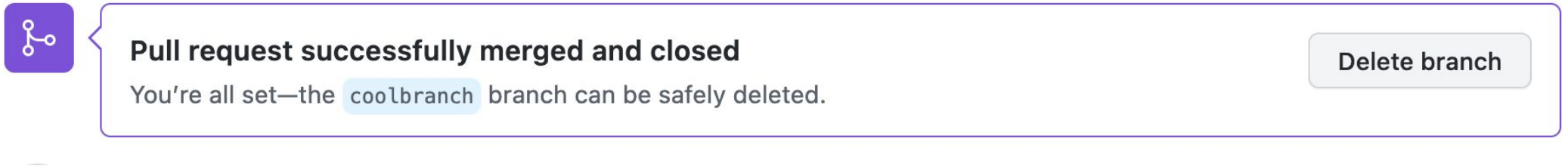
- If there are no merge conflicts we can merge automagically!
- You can accept the pull request on your own repos



Pull Request

- Make sure to clean up your old branches!
  - keeps your project clean for others

- Deleting this branch in GitHub only removed it in the remote repo, so you will still have a local copy



Advanced

Fork

- A GitHub fork allows you to copy over an existing open source project into your GitHub account
  - Maintains link to original repo where you can fetch updates
  - The original is called the “upstream” repository

- Practice: https://github.com/ghidalgo93/git-PR-practice
  - Once you have a local copy (fork, then clone it locally)
  - Make any changes (LOTS of spelling mistakes in github_wiki.txt)
  - Commit back up to your own GitHub repo
  - Submit a pull request to my repo!

Merge Conflicts

- Conflict occurs when you try to merge 2 different branches that modify the same lines

- Let's make a conflict!
  - Commit from 2 branches (main and coolbranch) with changes to a file on the same line
  - From main:

```
$ git merge coolbranch
```

  - You'll get an error that the merge failed

Merge Conflicts

- Go into the file that caused the commit to fail
  - or view those with: `$ git diff`

- Remove whichever lines are not needed
  - or keep both one after the other

- Then rerun your commit (called a merge commit)

Review: Learning Goals

1. Understand basics of version control
2. Differences between Git, GitHub
3. Basic Git fluency
4. How to collaborate on a project with Git

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/

- **Helpdesk**: rc-help@colorado.edu

Questions

Survey and feedback

http://tinyurl.com/curc-survey18