NYU COURANT INSTITUTE OF MATHEMATICAL SCIENCES

Social Coding With GitHub

github SOCIAL CODING

Fall 2020, CSCI-GA 2820, Graduate Division, Computer Science

Instructor:

John J Rofrano

Senior Technical Staff Member | DevOps Champion

IBM T.J. Watson Research Center

rofrano@cs.nyu.edu (@JohnRofrano)

Objectives

The objectives of this class are to:

- Learn the mechanics of using Git
- Introduce the Git Feature Branch Workflow
- Learn how Git encourages Collaboration
- Become a *Social Coder*

“What makes senior developers senior is not that they know the syntax of a given language better, but that they have experience working with large and complex projects with real users and business goals”

–Ariel Camus - medium

HOW TO BECOME A GIT JEDI

How to become a Git Jedi

Rule #1: Create a Git repository for every new project

How to become a Git Jedi

Rule #1: Create a Git repository for every new project

Rule #2: Create a new branch for every new feature

How to become a Git Jedi

Rule #1: Create a Git repository for every new project

Rule #2: Create a new branch for every new feature

Rule #3: Use Pull Requests to merge code to Master

Source Code Management

- Source Code Management is the practice of tracking versions of source code as it is being developed
- A source code manager (SCM) is a software tool used by teams of programmers to manage source code
- These are also referred to as Version Control Systems (VCS)
- SCMs can be Centralized or Distributed

History of Version Control Systems

- **First Generation:** SCCS (Source Code Control System) RCS (Revision Control System)
  - Local changes tracked for individual files and checked-out files could only be edited locally by one user at a time
- **Second Generation:** CVS (Concurrent Versions System) SVN (Apache Subversion) Perforce Helix Core
  - Centralized a version control systems introducing networking which led to centralized repositories that contained the 'official' versions of their projects.
- **Third Generation:** Git. Mercurial. BitKeeper. Darcs (Darcs Advanced Revision Control System)
  - Distributed VCS, all copies of the repository are created equal - there is no central copy of the repository
  - This opens the path for commits, branches, and merges to be created locally without network access and pushed to other repositories as needed

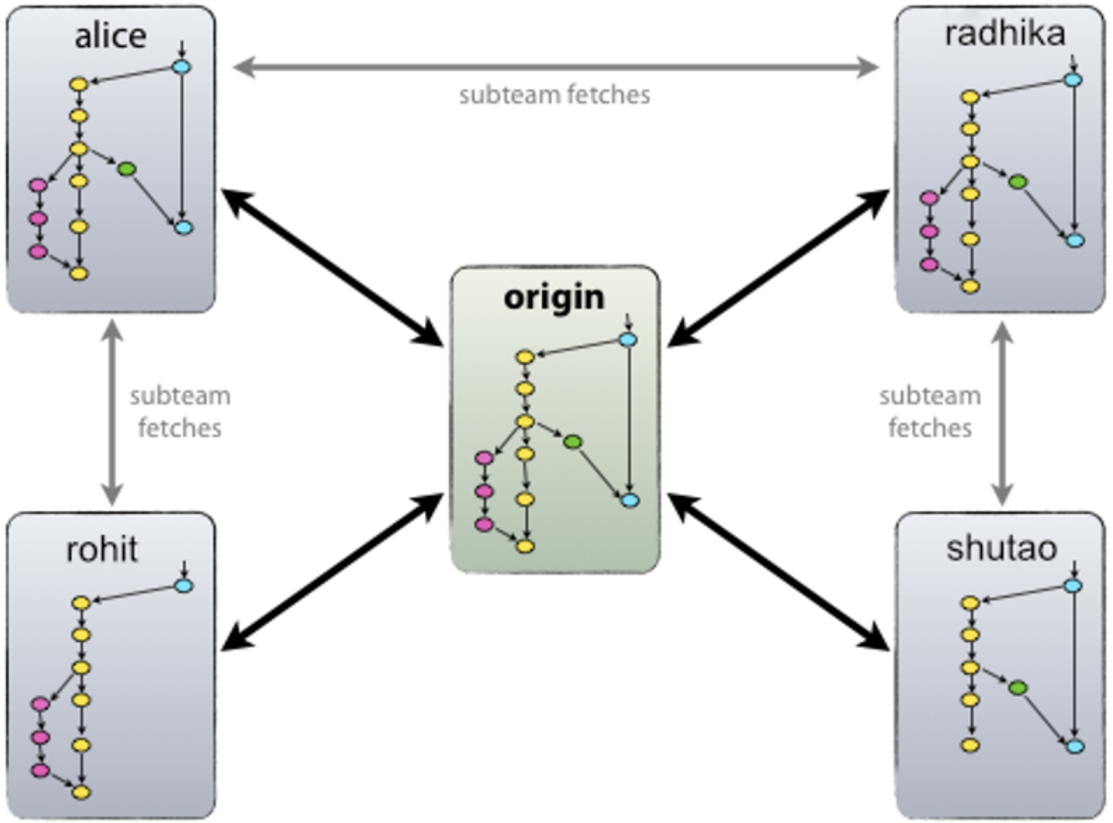
https://initialcommit.com/blog/Technical-Guide-VCS-Internals

Distributed or Centralized

- Distributed revision control (DRCS) takes a peer-to-peer approach, as opposed to the client-server approach of centralized systems
- Rather than a single, central repository on which clients synchronize, each peer's working copy of the codebase is a bona-fide repository
- Distributed revision control conducts synchronization by exchanging patches (change-sets) from peer to peer

Decentralized systems Like Git

- No canonical, reference copy of the codebase exists by default; only working copies.
- Common operations (such as commits, viewing history, and reverting changes) are fast, because there is no need to communicate with a central server.
- Communication is only necessary when pushing or pulling changes to or from other peers.
- Each working copy effectively functions as a remote backup of the codebase and of its change-history, providing natural protection against data loss.



Git

- A distributed source code management (SCM) tool invented by Linus Torvalds in 2005 for Linux kernel development
- Code is kept in a repository and every developer has a full copy
- Works locally without any server
- Works remotely with GitHub, GitLab, & BitBucket

Git

- The first distributed version control that changed the workflow:
  - A switch fundamental from the developer asking, “Can you add me to version control?”
  - To making their own copy, changing the code, and then checking in their contribution.
- This change was revolutionary in that developers no longer needed to be invited to contribute to open source code repositories as well as proprietary code.

GitHub

- A web site that hosts git repositories
- Founded in 2007, has 40 million registered developers and was acquired by Microsoft for a whopping $7.5 billion in 2018
- Free and Paid accounts
- Adds ability to track Issues and Bugs
- Provides webhooks to integrate other tools
- ...and a whole lot more

Bitbucket

- A web site that hosts git repositories
- Founded in 2010 and owned by Atlassian
- Free and Paid accounts
- Adds ability to track Issues and Bugs
- Provides webhooks to integrate other tools
- ...and a whole lot more

GitLab

- Founded in 2011 as an OpenSource competitor to GitHub
- You can run your own GitLab servers for free
- Also runs a web site to host git repositories
- Adds ability to track Issues and Bugs
- Provides webhooks to integrate other tools
- ...and a whole lot more

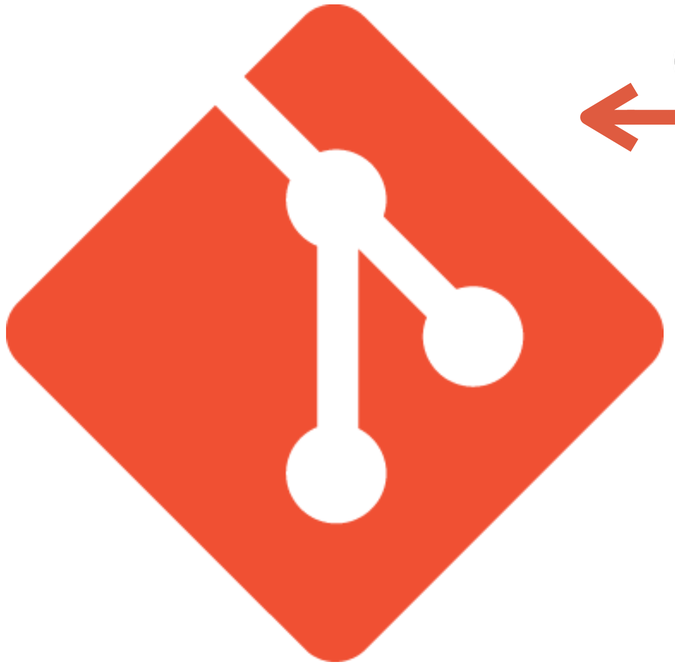
We will use GitHub for this class

...but we could have used any one of these

GitLab

Bitbucket

Git vs GitHub



Git is the tool

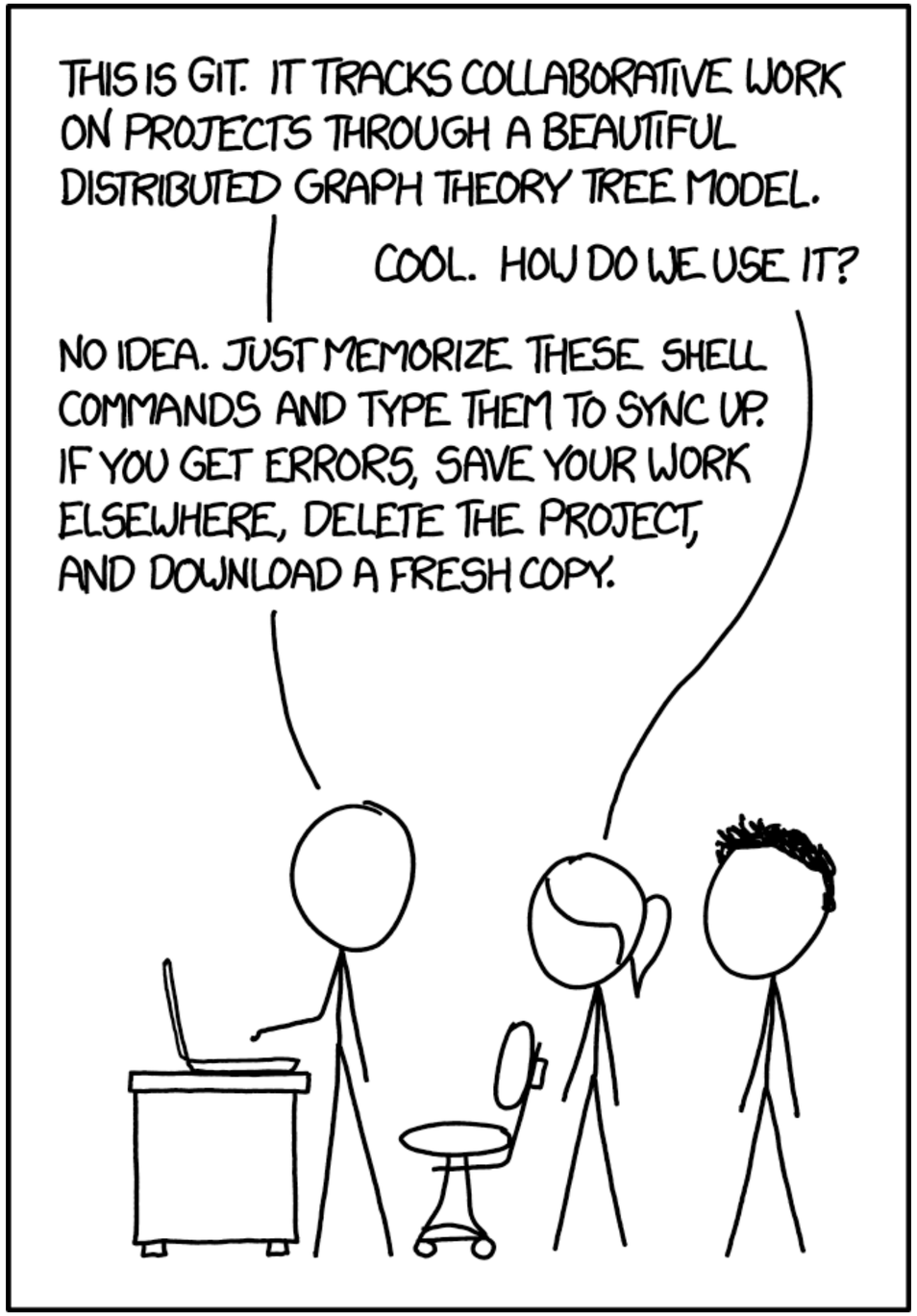
GitHub is the web site

Why You Need to Know Git

- The use of Git and software code repositories has fundamentally changed the way enterprises do software development for proprietary code and open source code alike.
- In addition, open source software has blossomed because some of the Git repositories host open source code for free.
- Most significantly, Git and code repositories have facilitated the DevOps methodology that is so impactful to software development.

How many of you know Git?

How many of you know Git?



Git Command Workflow

workspace

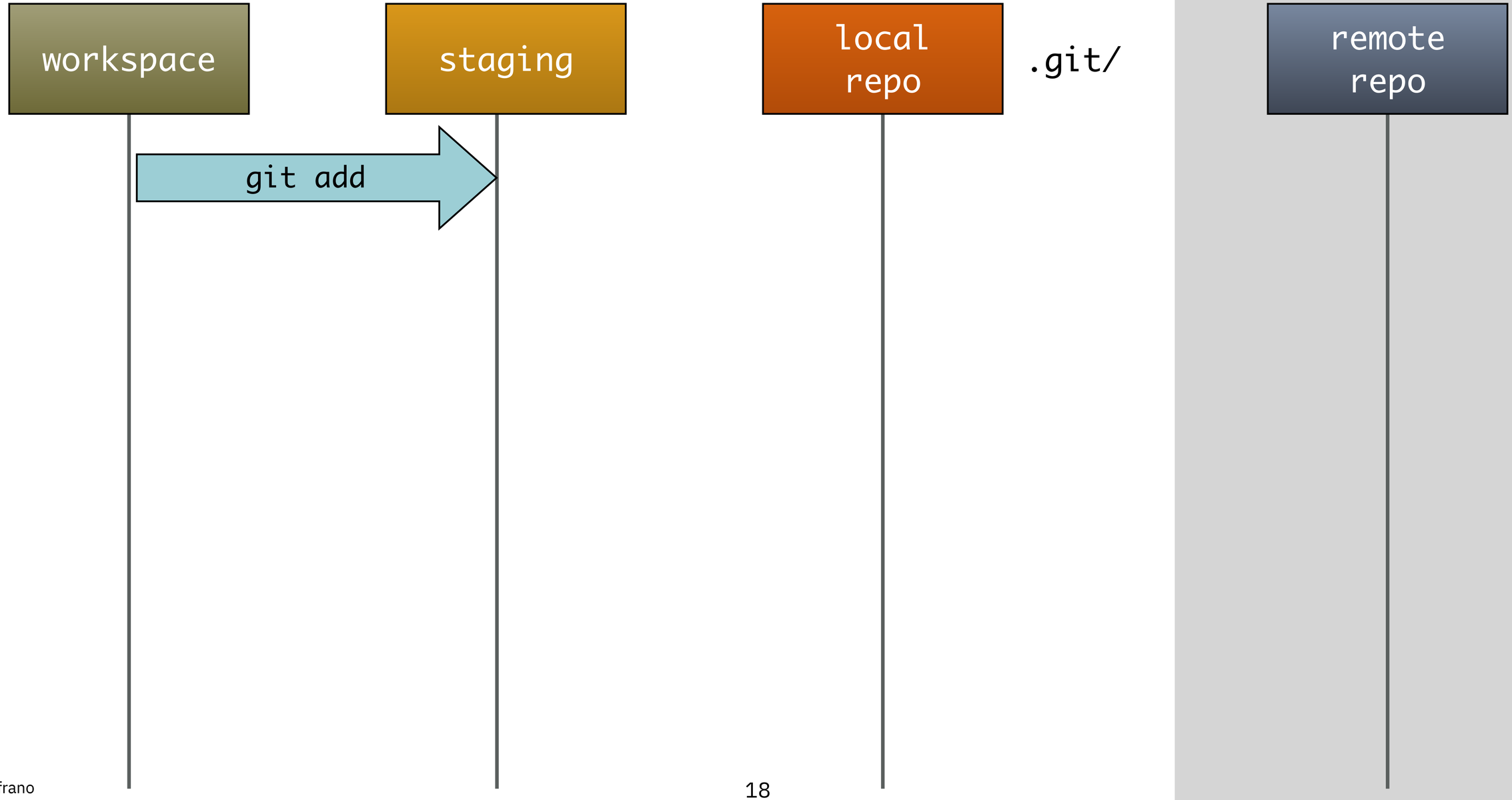
staging

local repo

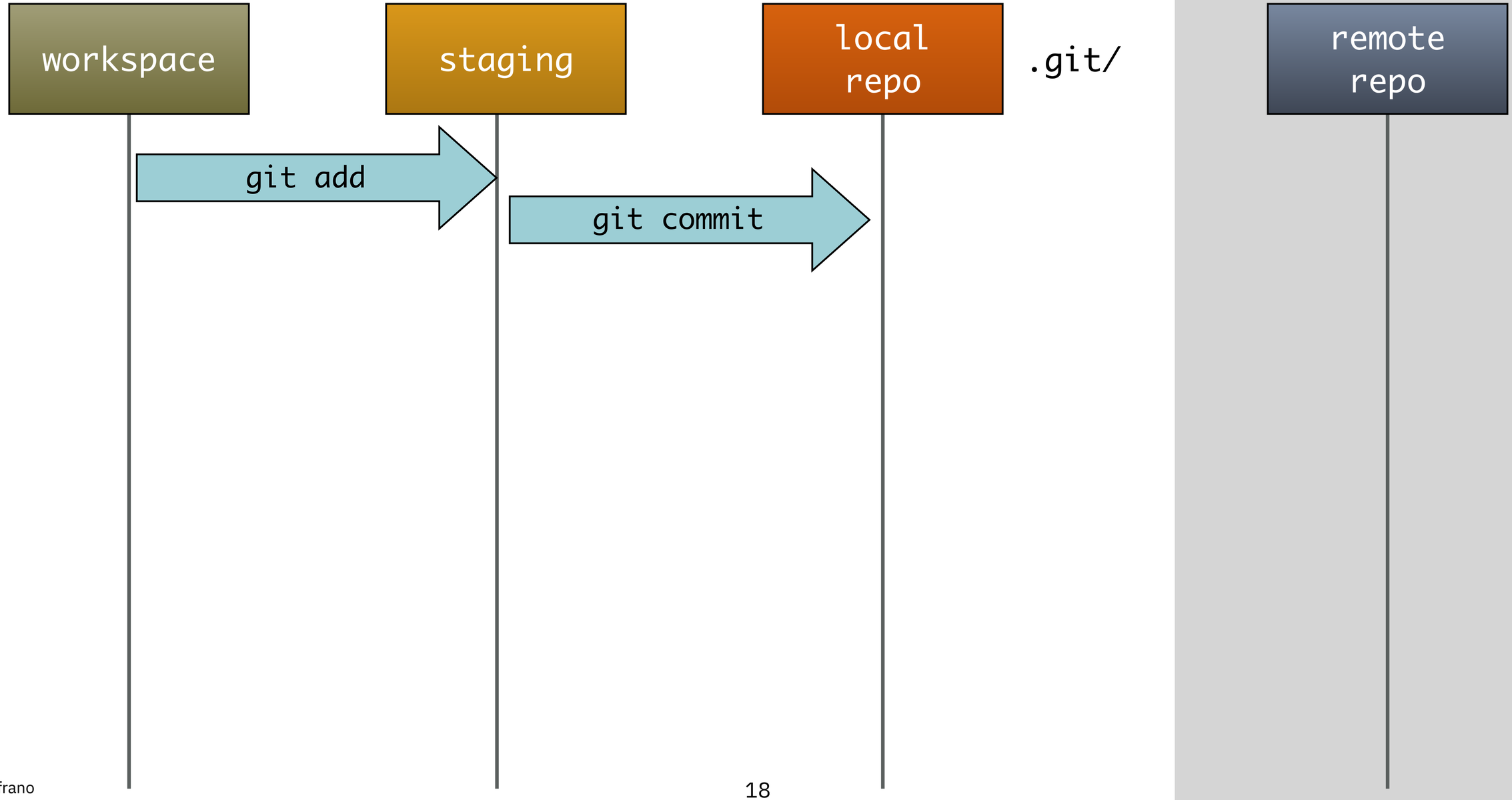
.git/

remote repo

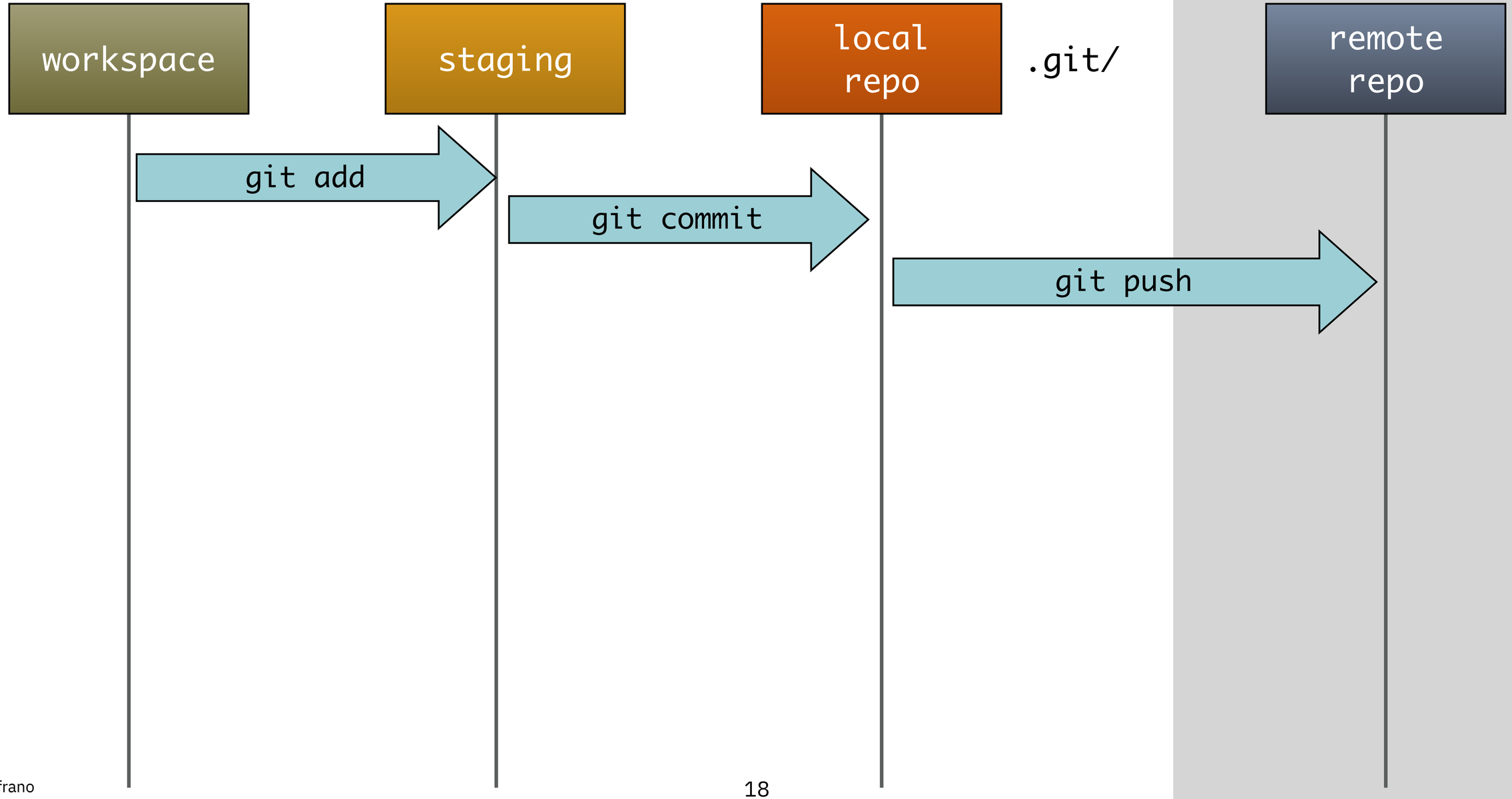
Git Command Workflow



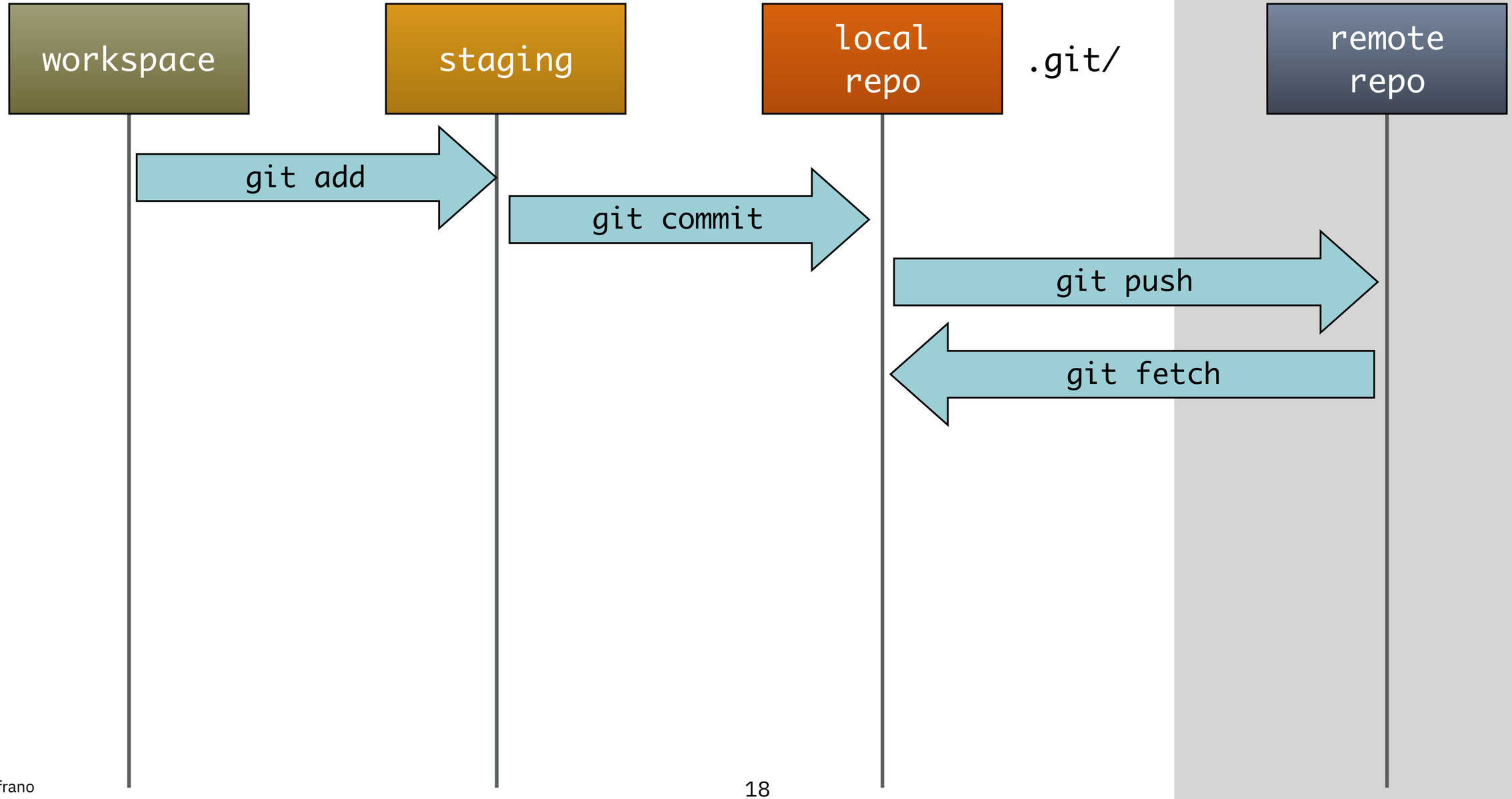
Git Command Workflow



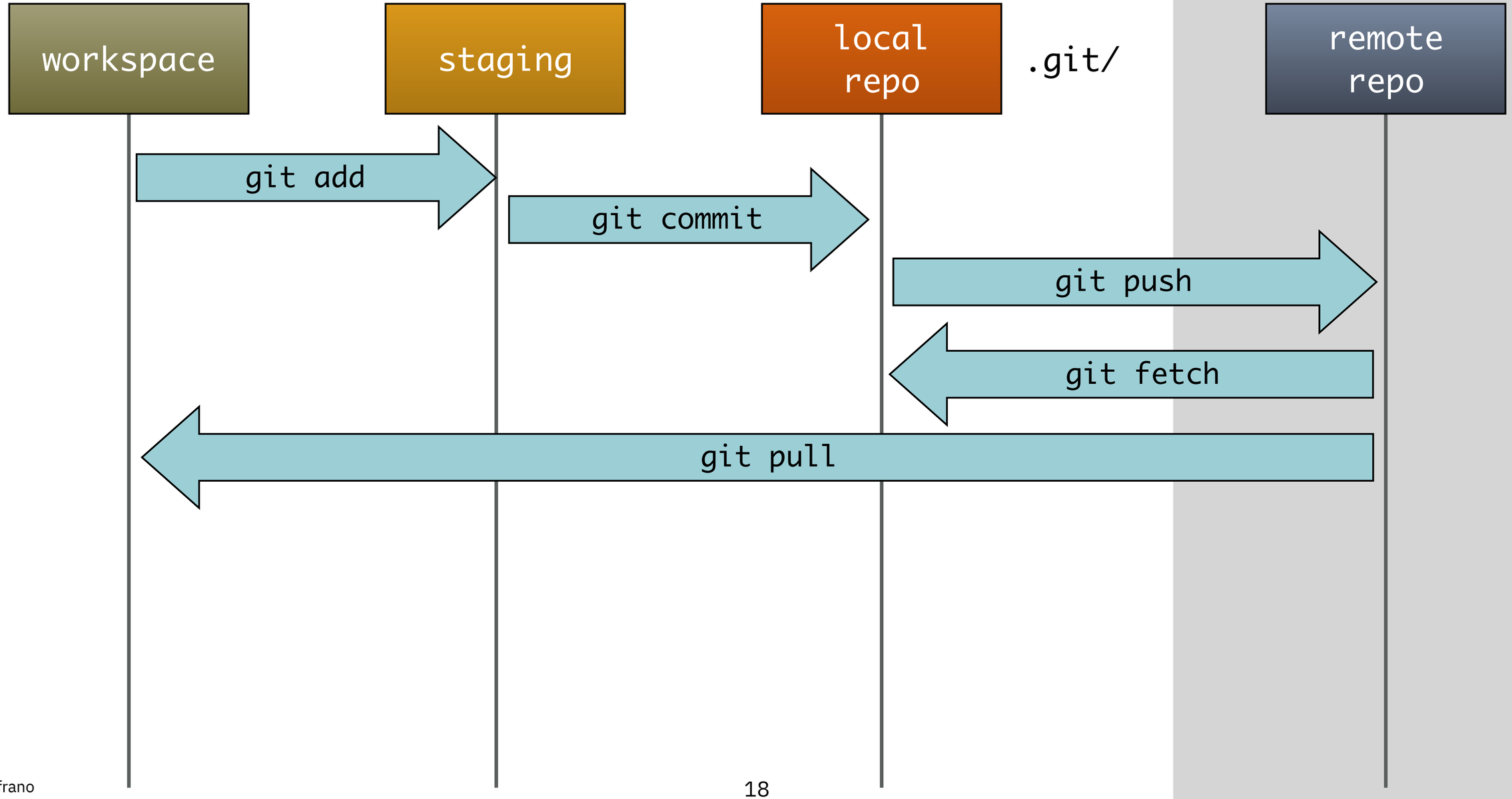
Git Command Workflow



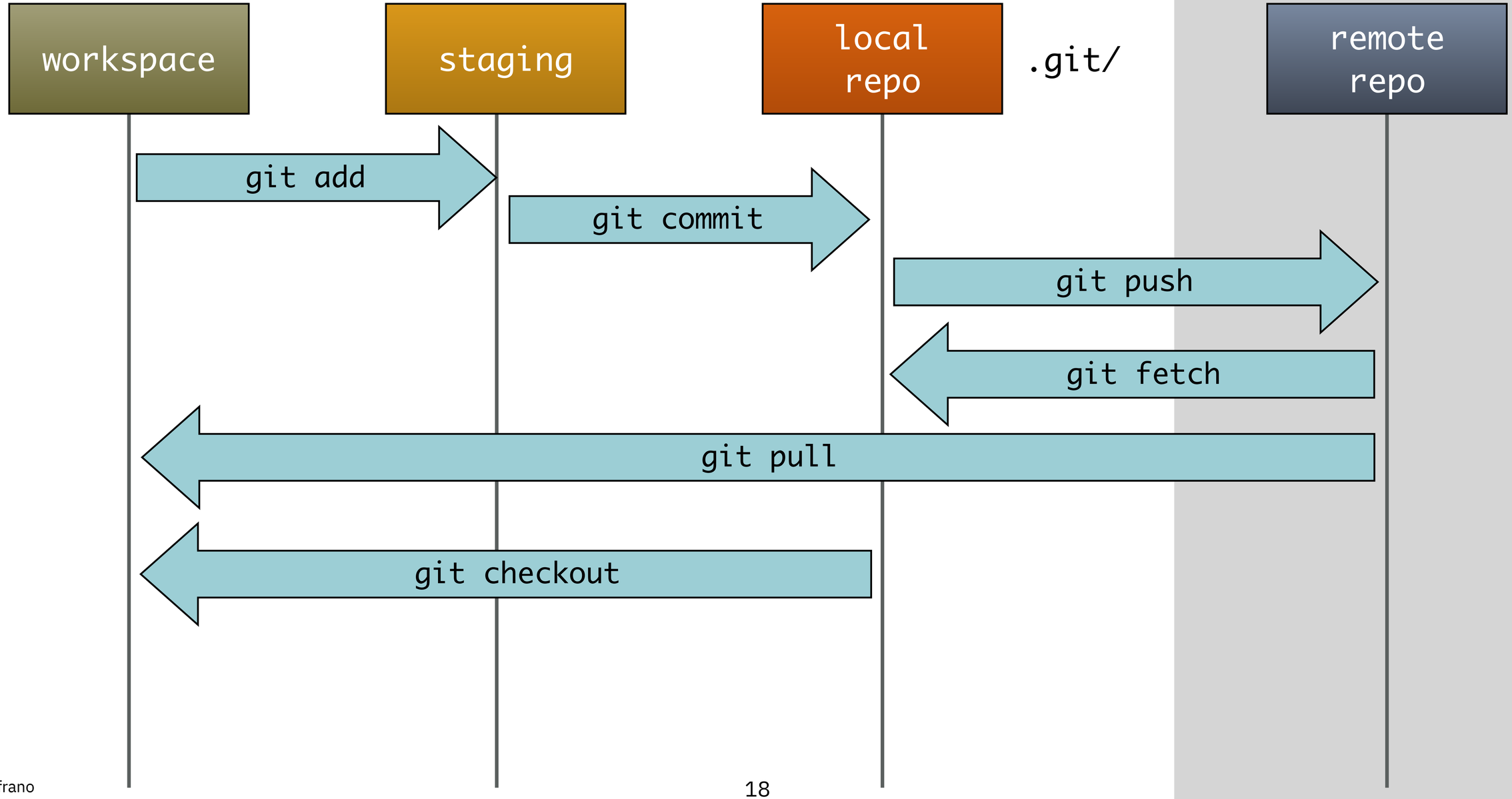
Git Command Workflow



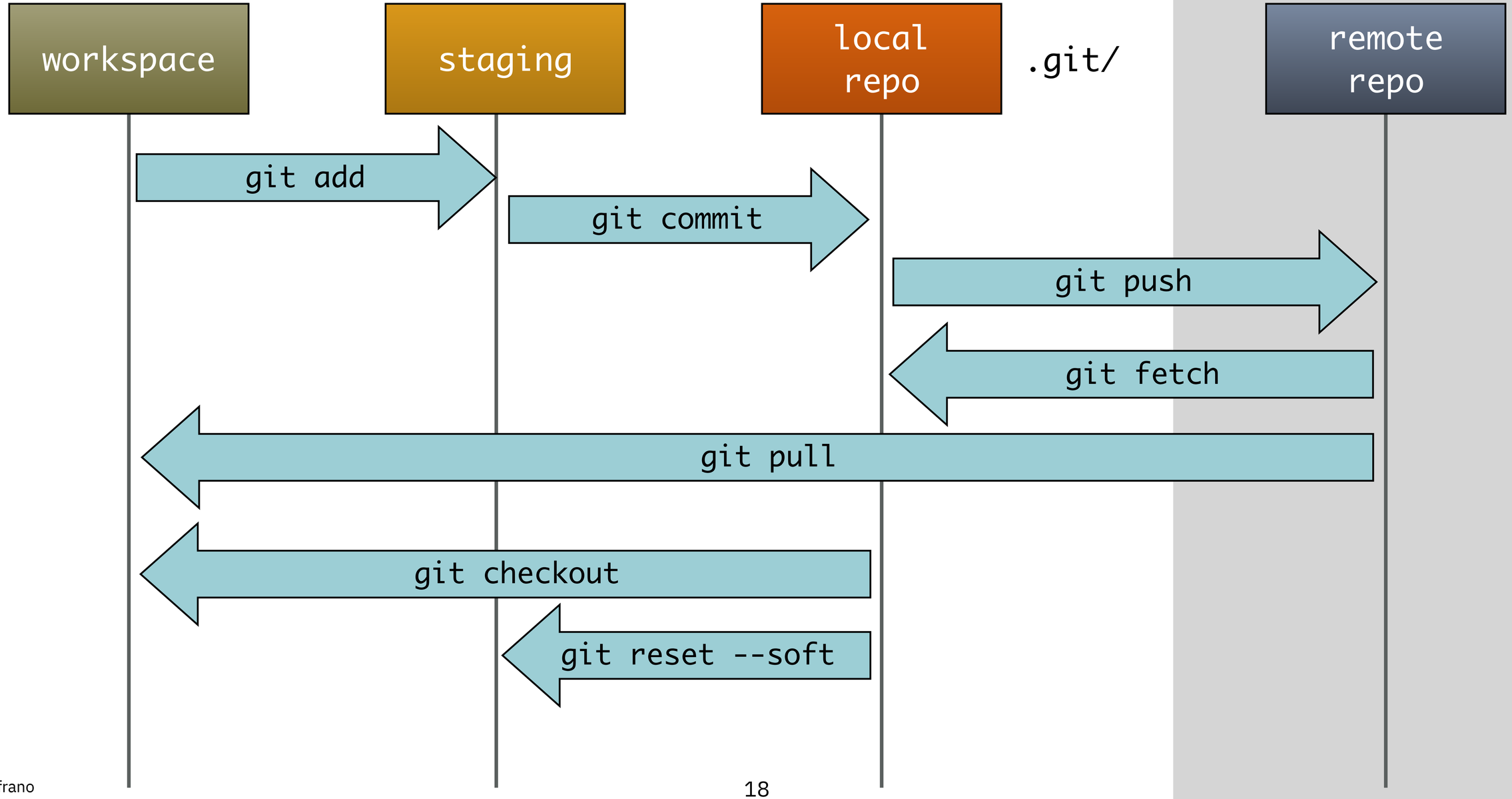
Git Command Workflow



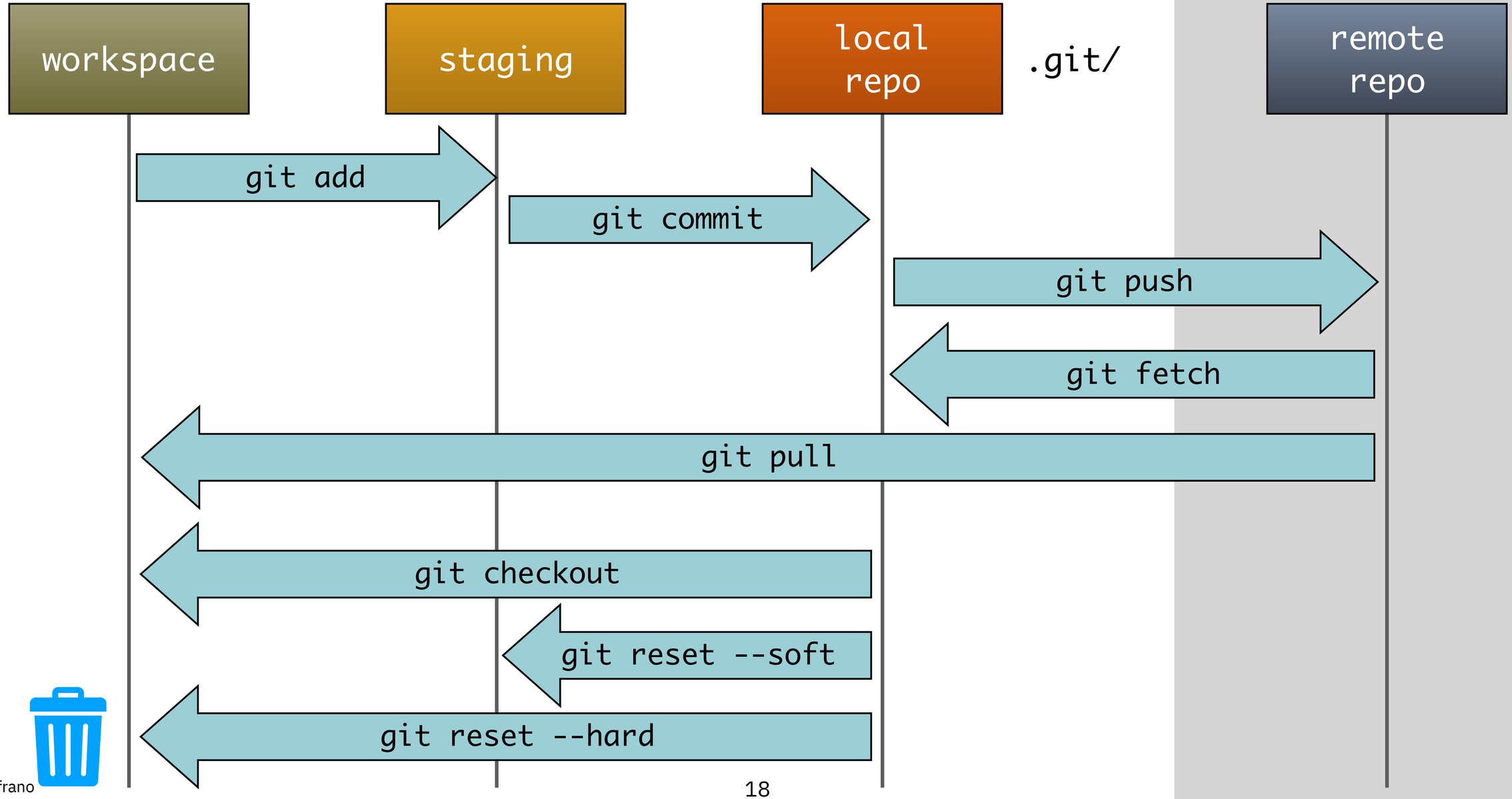
Git Command Workflow



Git Command Workflow



Git Command Workflow




github
SOCIAL CODING
BUILD SOFTWARE BETTER
TOGETHER

What is Social Coding?

- In the past developers worked on **private** repositories and you had to be a member of the team to contribute
- With Social Coding, repositories are **public** and everyone is encouraged to Fork the code and contribute
- You would think that anarchy would ensue but it actually works quite well because it is controlled by the repository owner

Code reuse Dilemma

- You see a project that is 80% of what you need but there are some missing features
- You feel that if you make a feature request of the project owner, your request will be at the bottom of their priorities
  - If they get funding cuts, you know that your feature request will be cut
- So you rebuild 100% of what you need so as not to have a dependency on another project

Social Coding Solution

- Discuss the new feature with the repo owner and agree to develop it
- Open an **Issue** and assign it to yourself so that everyone knows what you are working on
- **Fork** the code, create a **branch,** and make your changes
- Issue a **Pull Request** when you are ready to review and merge your work back into the main project

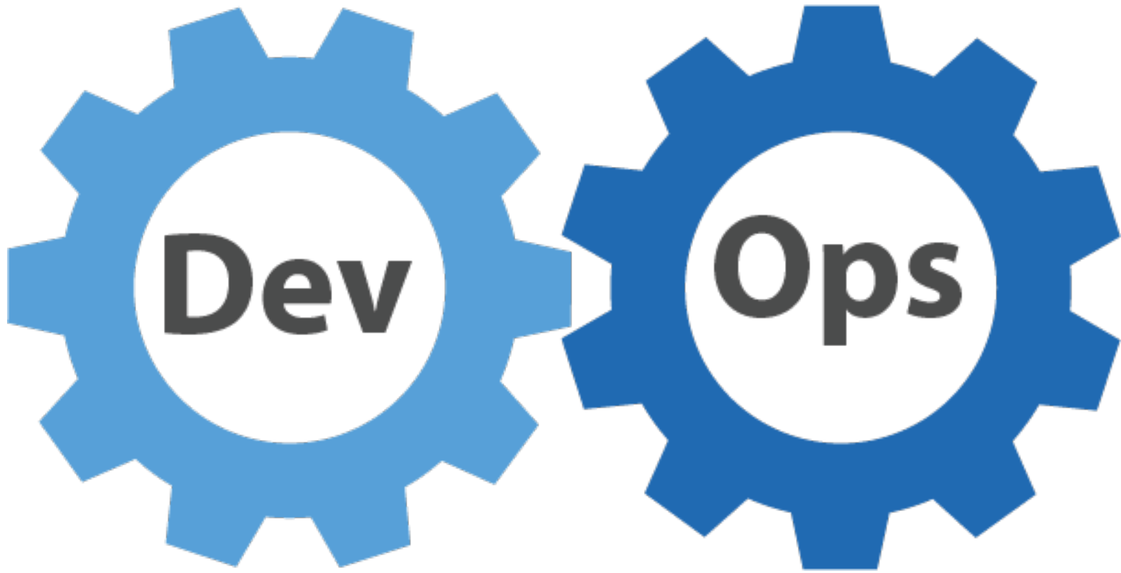
Style Guide for Python Code

- PEP 8 -- Style Guide for Python Code
  - https://www.python.org/dev/peps/pep-0008/
- Coding Standards are critical so that the code looks consistent even though it is written by multiple contributors
- PyLint can be used to enforce PEP8 plus more

Why Git and DevOps?

GitHub Is A DevOps Enabler

- Open Culture
  - √ Forking and Pull Requests enable openness
- Emphasize Collaboration and Sharing (no Silos)
  - √ Issues and Comments invite Collaboration and Planning
- Infrastructure as Code
  - √ Git the the perfect place to store all of your infrastructure scripts
- Automate Everything
  - √ Git works well with Travis CI, Jenkins, and other automation tools



Git Overview

- Git is a Decentralized Source Code Management (SCM) System
- Developers work in their own BRANCHES (even in FORKS)
- The MASTER branch should always be ready to deploy
- PULL REQUESTS are used to MERGE code BRANCHES into MASTER

Huh?

Branches?

Forks?

Spoons?

Merge?

Master?

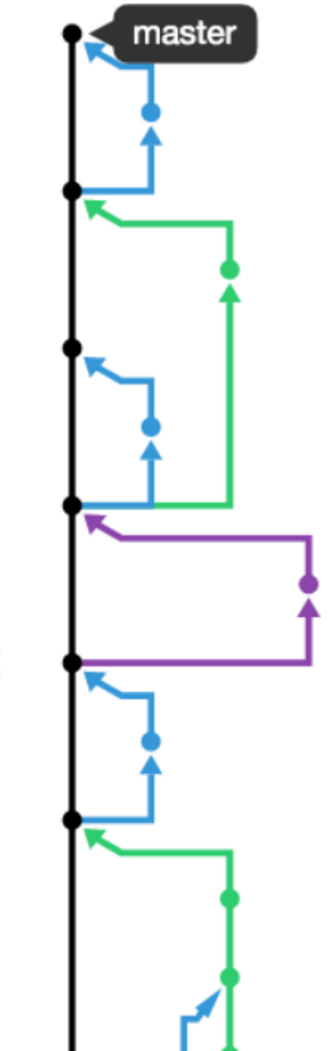
Pull Requests?

Pulling hair?

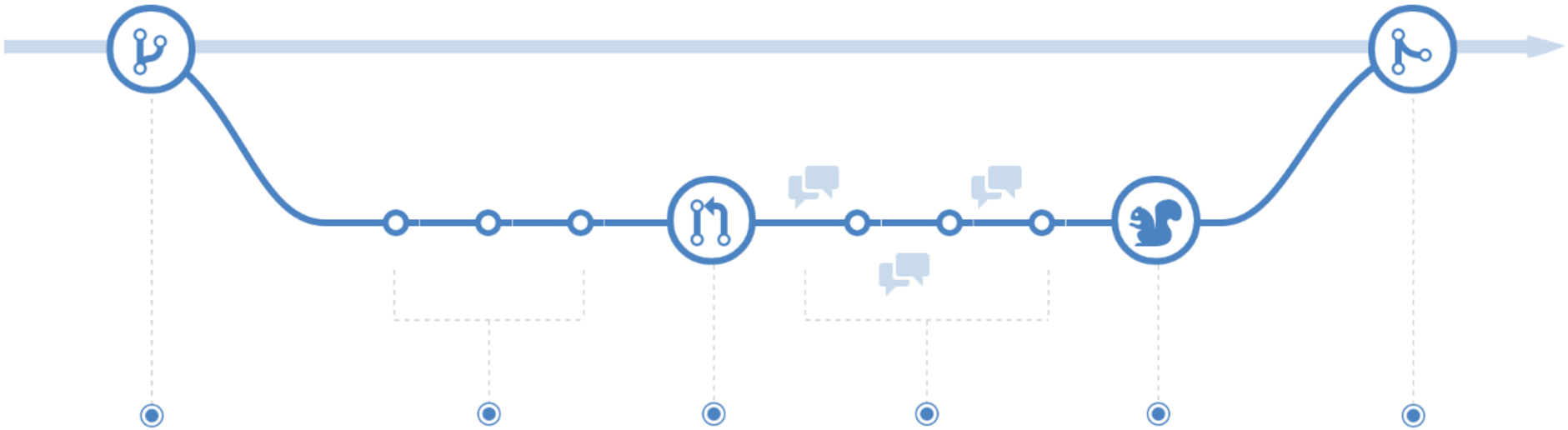
What?

Git Feature Branch Workflow

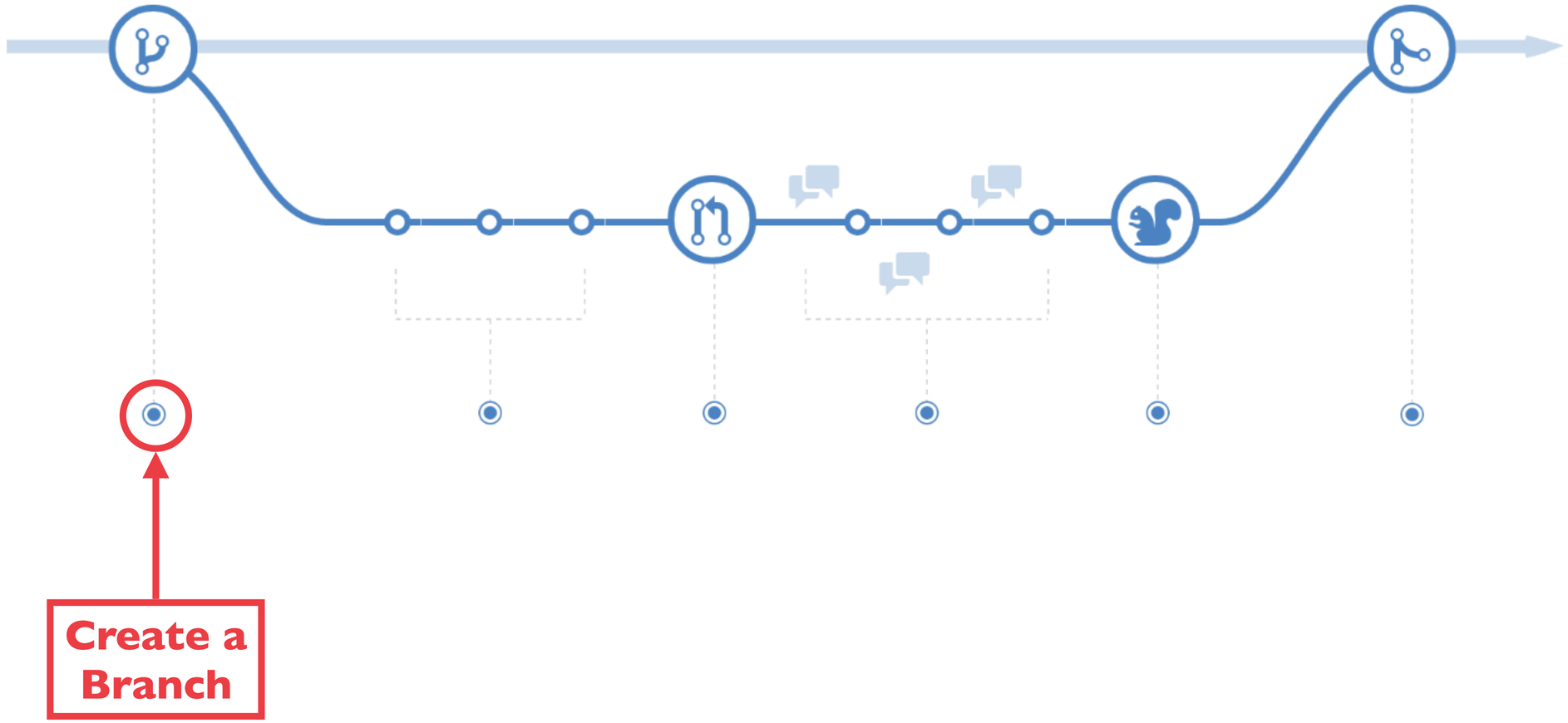
1. CREATE a new repository or FORK an exiting repository
2. CLONE it to your local system
3. Create a BRANCH to work on your ISSUE
4. COMMIT changes to that branch
5. PUSH your changes to the Remote branch
6. Issue a PULL REQUEST to have your work reviewed
7. MERGE your code to master and close the ISSUE



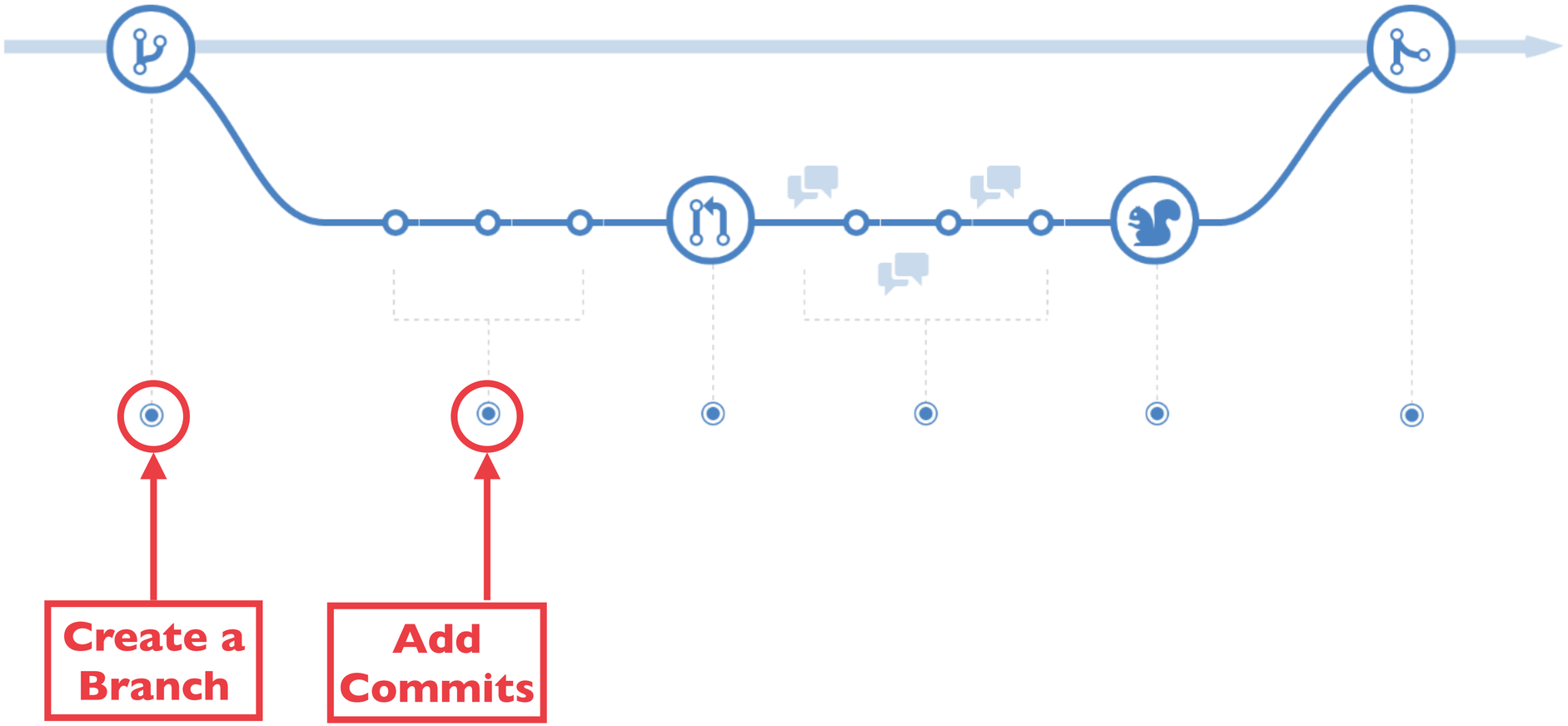
Git Feature Branch Workflow



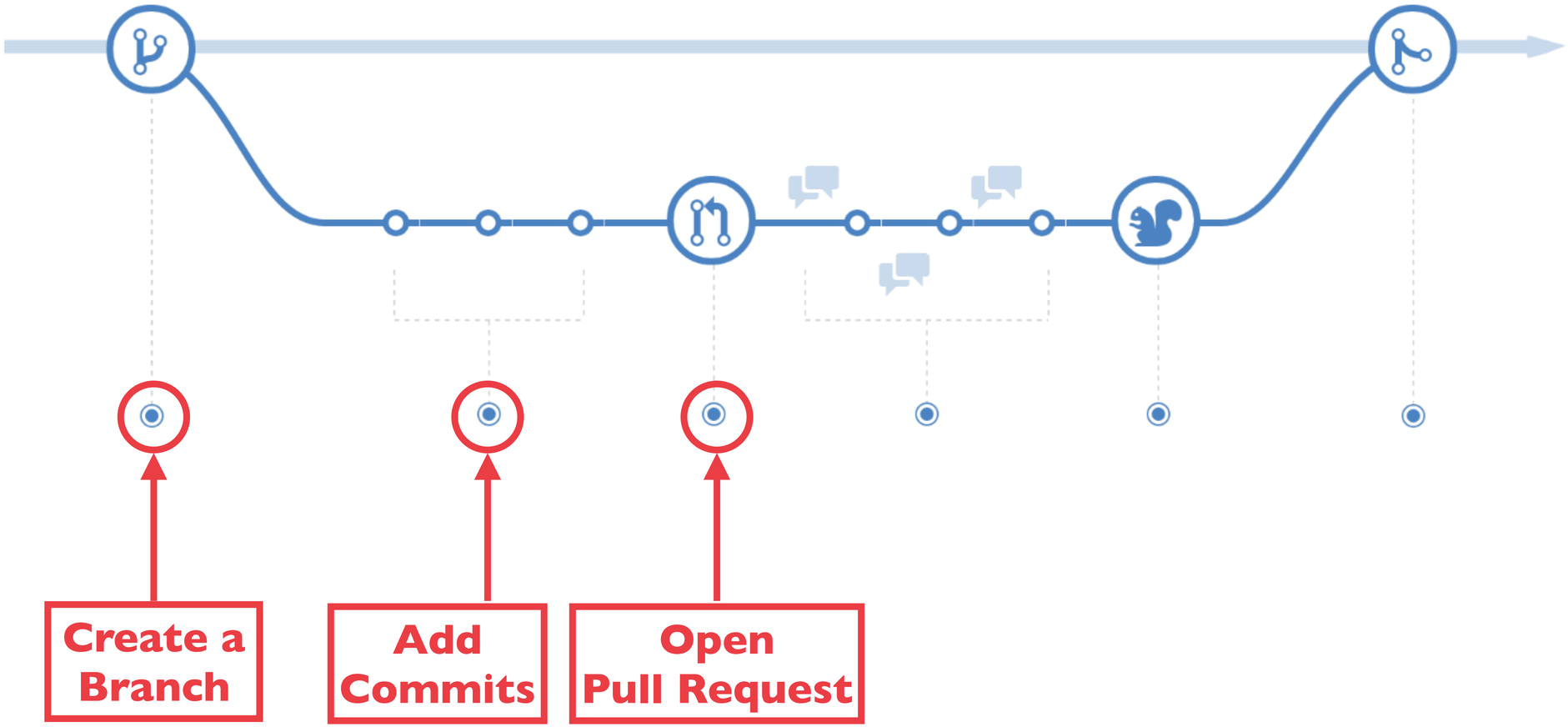
Git Feature Branch Workflow



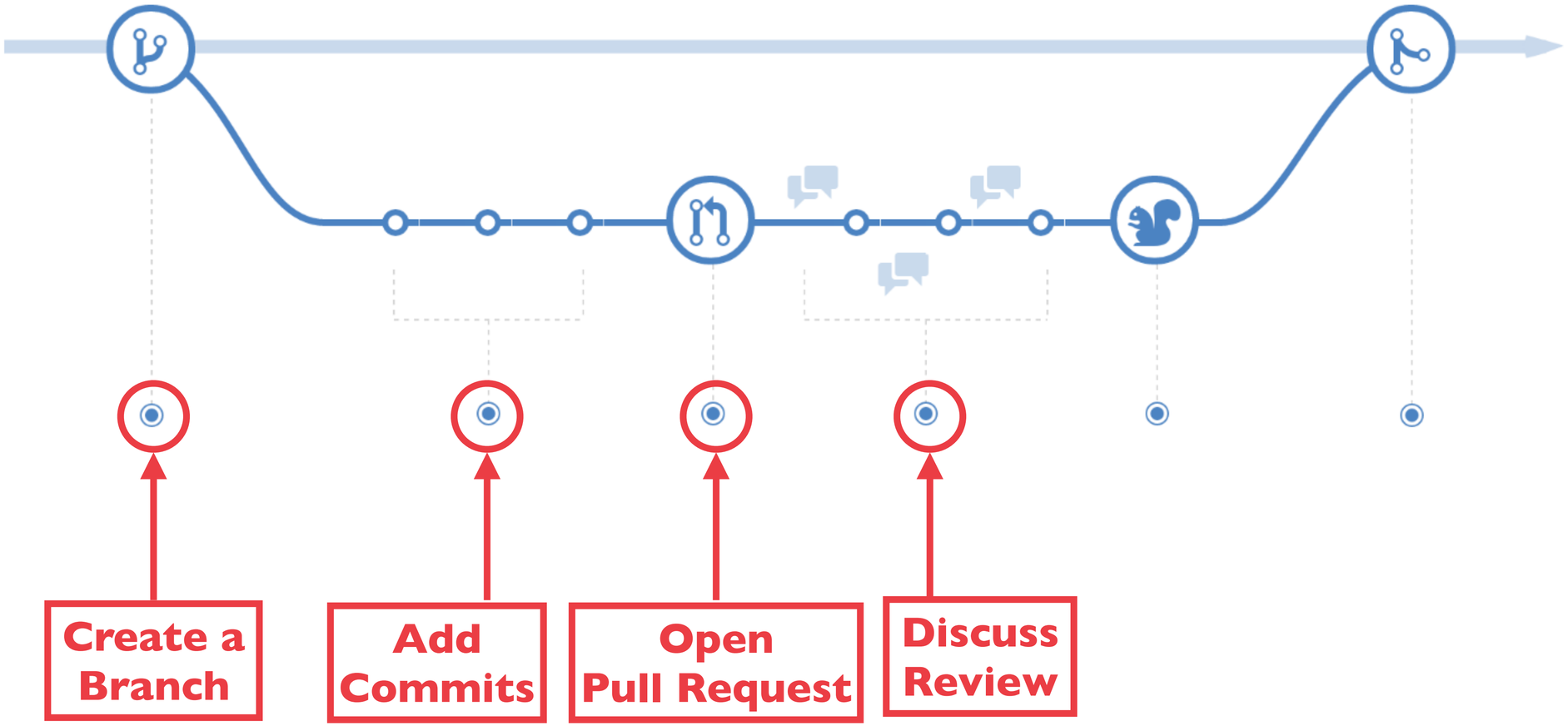
Git Feature Branch Workflow



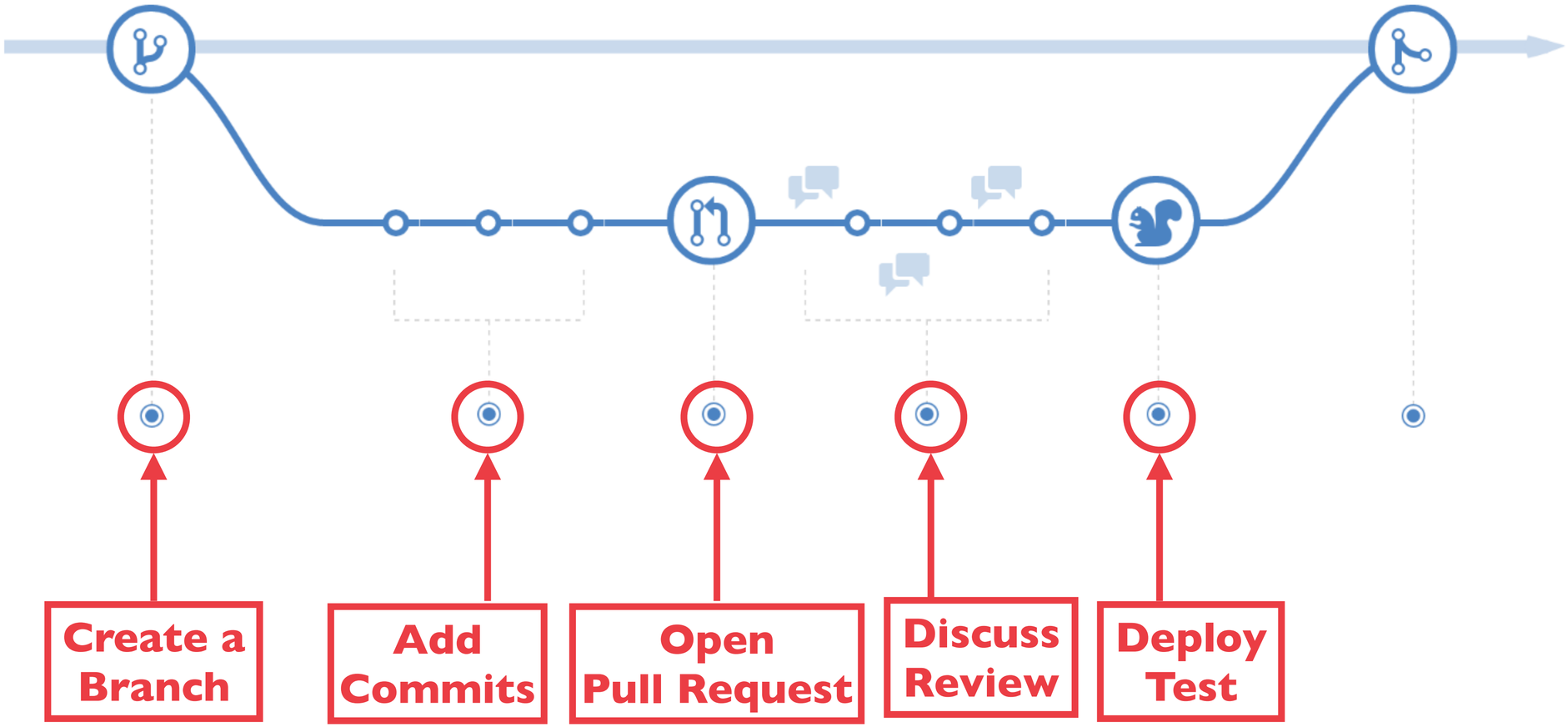
Git Feature Branch Workflow



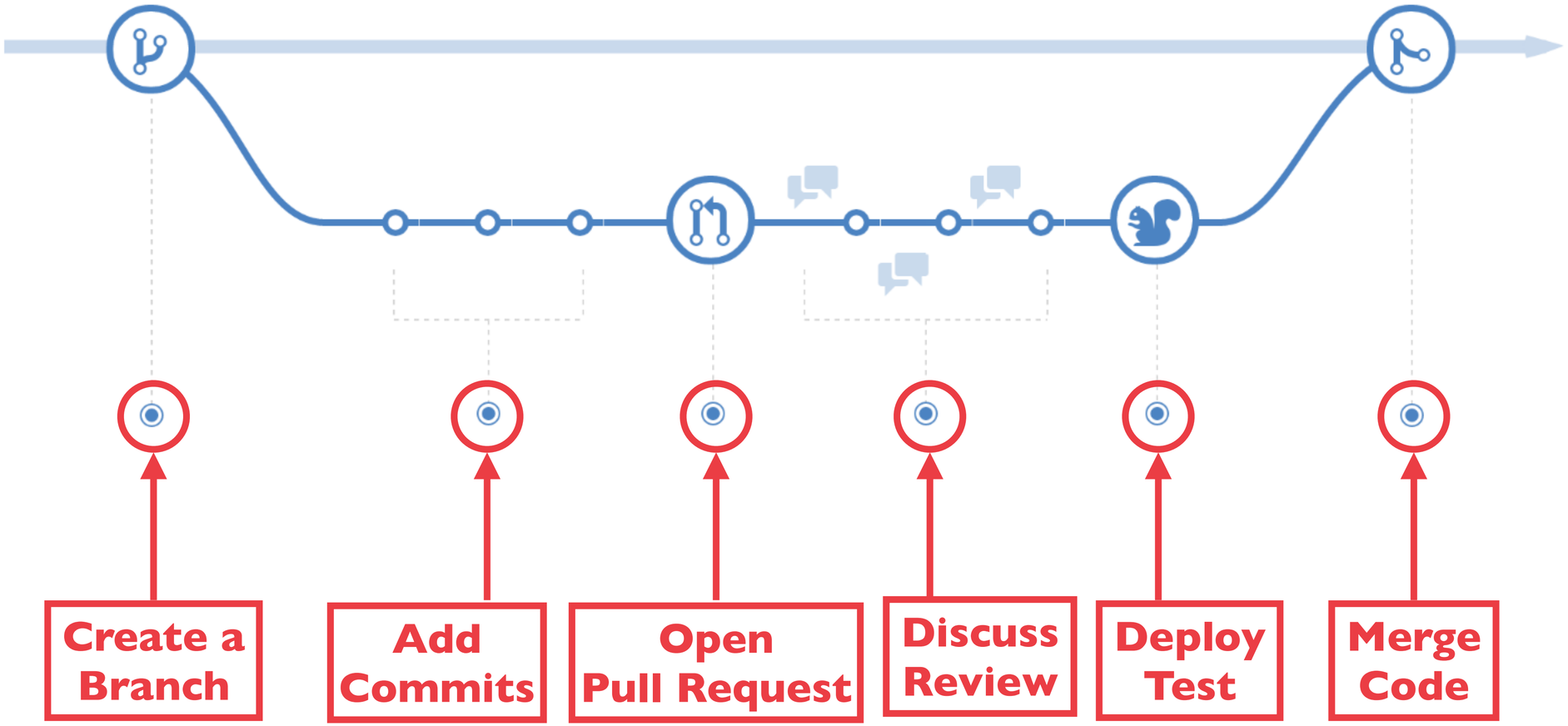
Git Feature Branch Workflow



Git Feature Branch Workflow

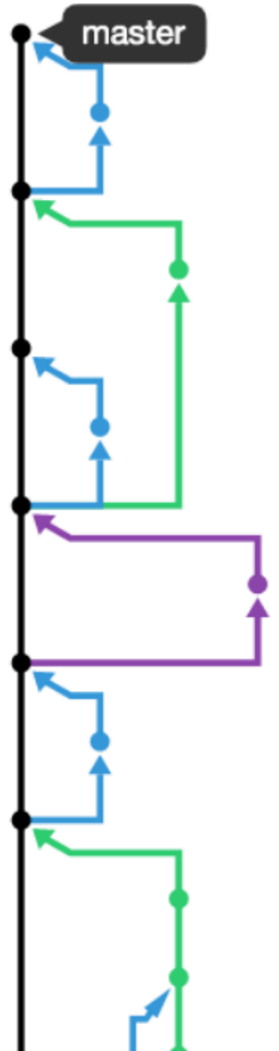


Git Feature Branch Workflow

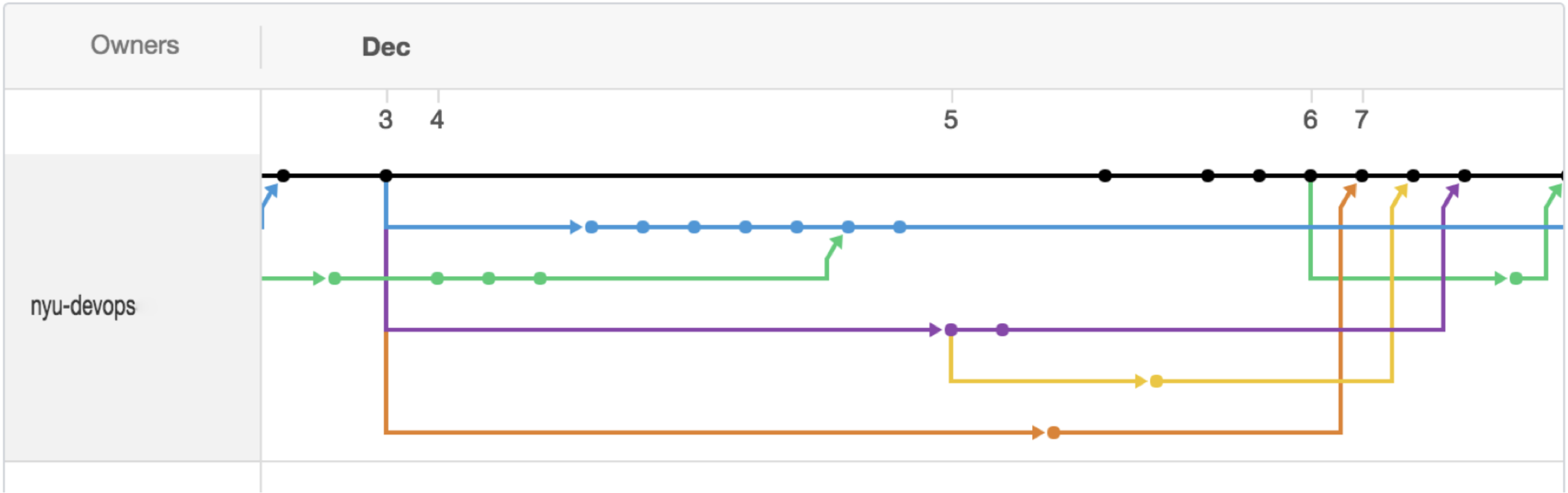


Detailed Feature Branch Workflow

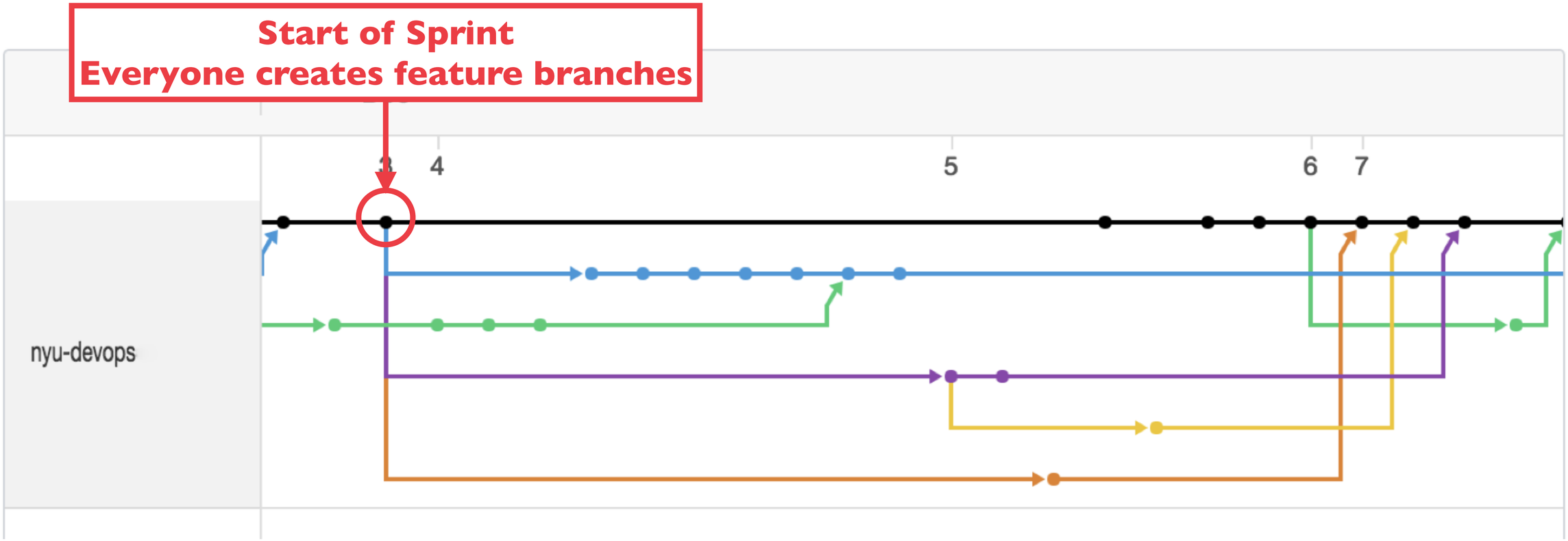
- CLONE a Repository (FORK first if not part of Dev Team)
- Assign the ISSUE for your work to yourself and place it in working status
- Create a BRANCH to work on an ISSUE
- Run the Test suite to make sure you can run the code
- Make changes to code and test cases and COMMIT to local BRANCH
- Run the Test suite early and often to make sure you didn’t break anything
- PUSH changes to remote BRANCH
- Did we mention testing the code early and often?
- Create PULL REQUEST when all tests pass and code is ready for review / MERGE



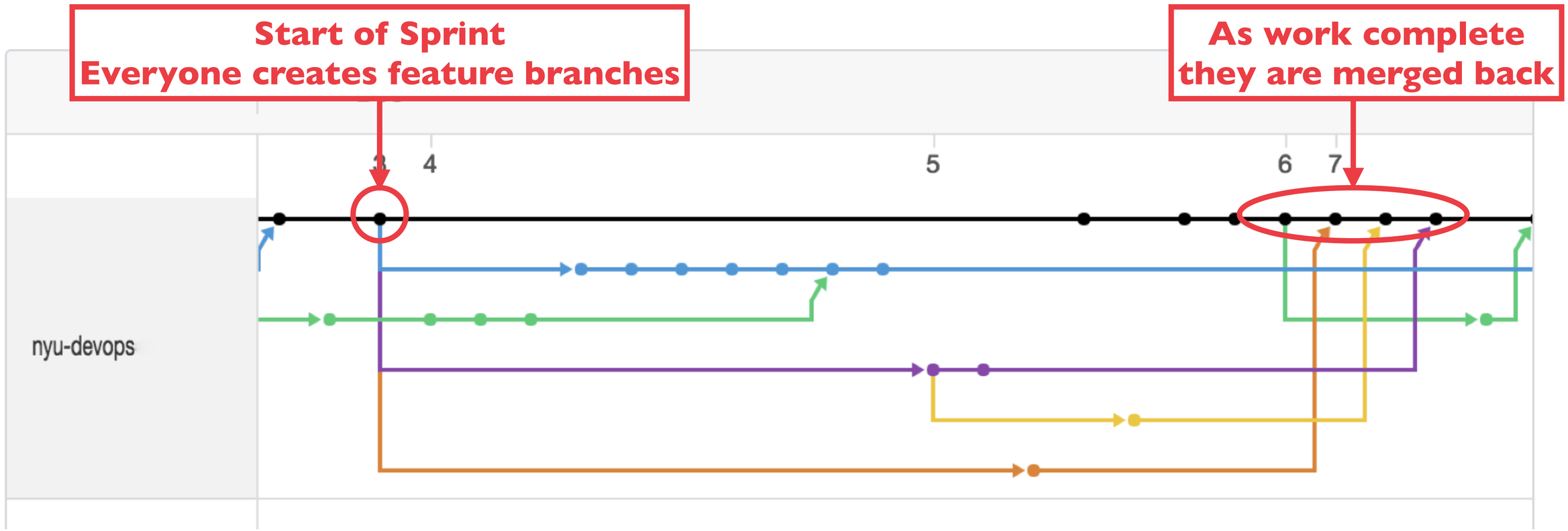
Branching example



Branching example



Branching example



Example workflow

- Your initial workflow should set up a remote branch that is tracked

```
git checkout -b my-new-feature
... write some code here ...
git add .
git commit -m 'initial working version of my new feature'
git push -u origin my-new-feature
```

- The next day you add more code and push to your remote branch

```
... write some code here ...
git add .
git commit -m 'added this cool code...'
git push
```

Example workflow

- Your initial workflow should set up a remote branch that is tracked

```
git checkout -b my-new-feature
... write some code here ...
git add .
git commit -m 'initial working version of my new feature'
git push -u origin my-new-feature
```

Sets up a remote branch and tracks it

- The next day you add more code and push to your remote branch

```
... write some code here ...
git add .
git commit -m 'added this cool code...'
git push
```

Example workflow

- Your initial workflow should set up a remote branch that is tracked

```
git checkout -b my-new-feature
... write some code here ...
git add .
git commit -m 'initial working version of my new feature'
git push -u origin my-new-feature
```

Sets up a remote branch and tracks it

- The next day you add more code and push to your remote branch

```
... write some code here ...
git add .
git commit -m 'added this cool code   '
git push
```

No need to use the branch name because it is tracked

Why is Testing So Important?

- Automated testing must pass before your Pull Request will be accepted
- Automated testing is your guarantee that you didn't break anything
- Without automated testing you cannot fully automate the DevOps pipeline

Hands-On

“live follow-along session”

Some Assembly Required

- Tools you will need to complete this lab:
  - GitHub Account
    - http://github.com
  - Git command line client
    - https://git-scm.com/downloads

.gitconfig

- It’s important to make sure that you have git configured correctly with your `user.name` and `user.email`
- You can use `git config -l` to list the values
- You can use `git config name value` to change them

```
git config --global user.name "Your Name Goes Here"

git config --global user.email you@example.com
```

Windows Users

- This will ensure that Windows doesn’t use crlf or flip the execute bit on all of your files (because Windows doesn’t support an execute bit so it doesn’t care)

```
$ git config --global core.fileMode false
$ git config --global core.eol lf
$ git config --global core.autocrlf input
```

Git Credentials

- If you are going to use HTTPS URLs when cloning Git repos, then you should setup the git command line to use a credential store
- On Windows

```
$ git config --global credential.helper wincred
```

- On MacOs you can use the OS X Keychain.

```
$ git config --global credential.helper osxkeychain
```

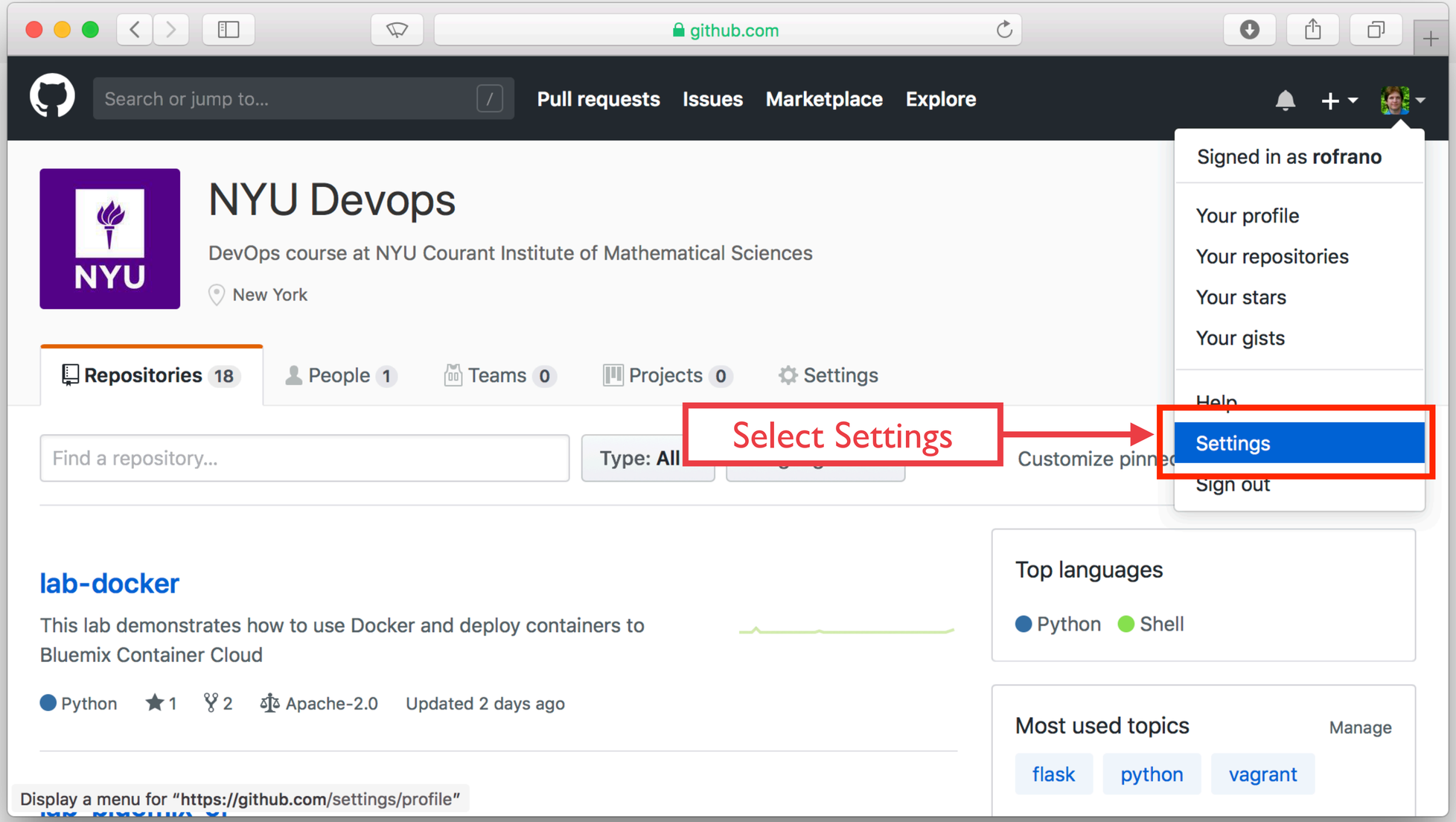
SSH Keys

- If you haven’t set up an ssh key to work with GitHub then you should do this now. In your Terminal window type the following:

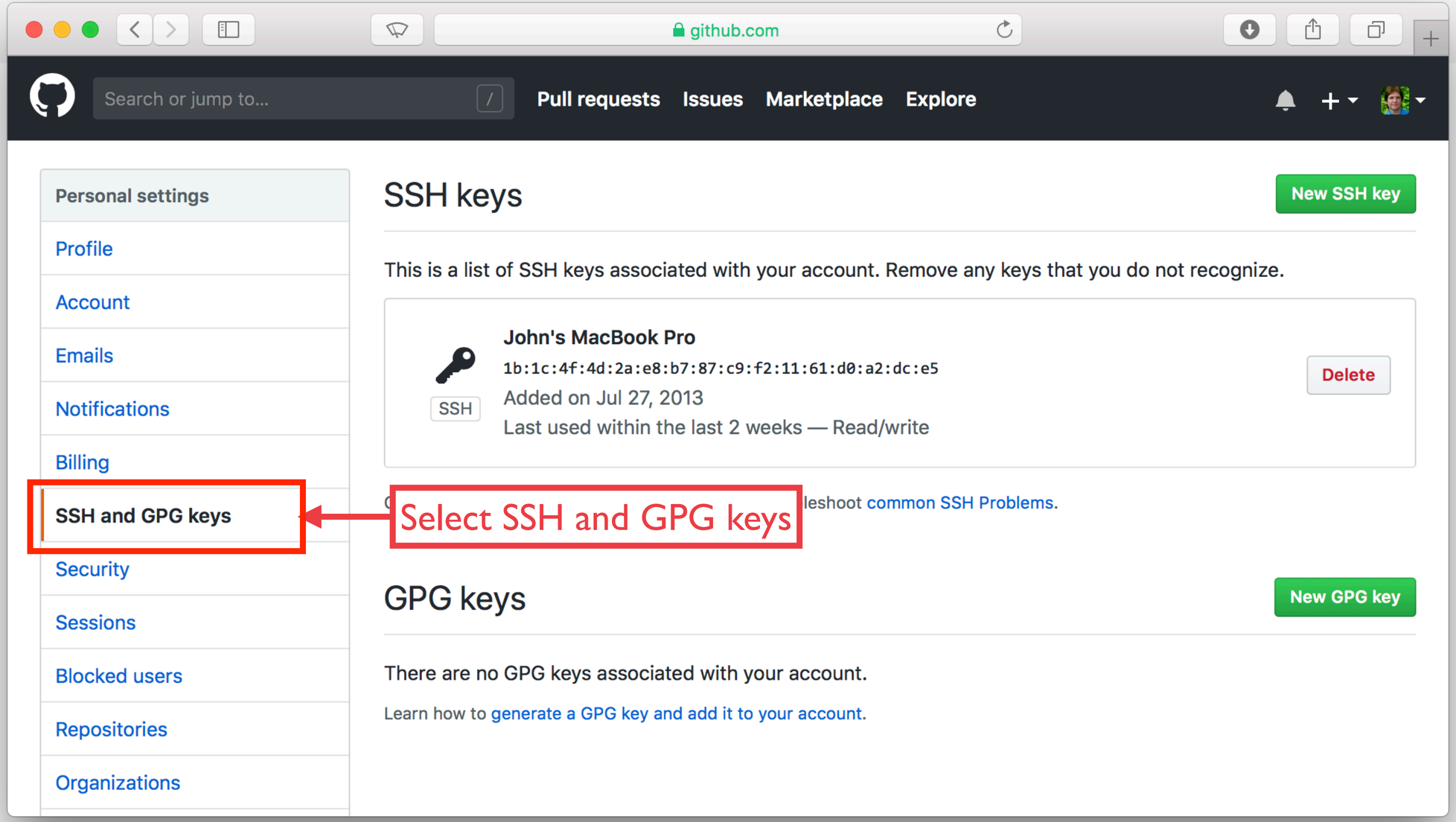
```
$ ssh-keygen -t rsa -b 4096 -C "youremail@nyu.edu"
```

- Just press ENTER at all of the prompts. This will generate a private and public ssh key that you can use with GitHub

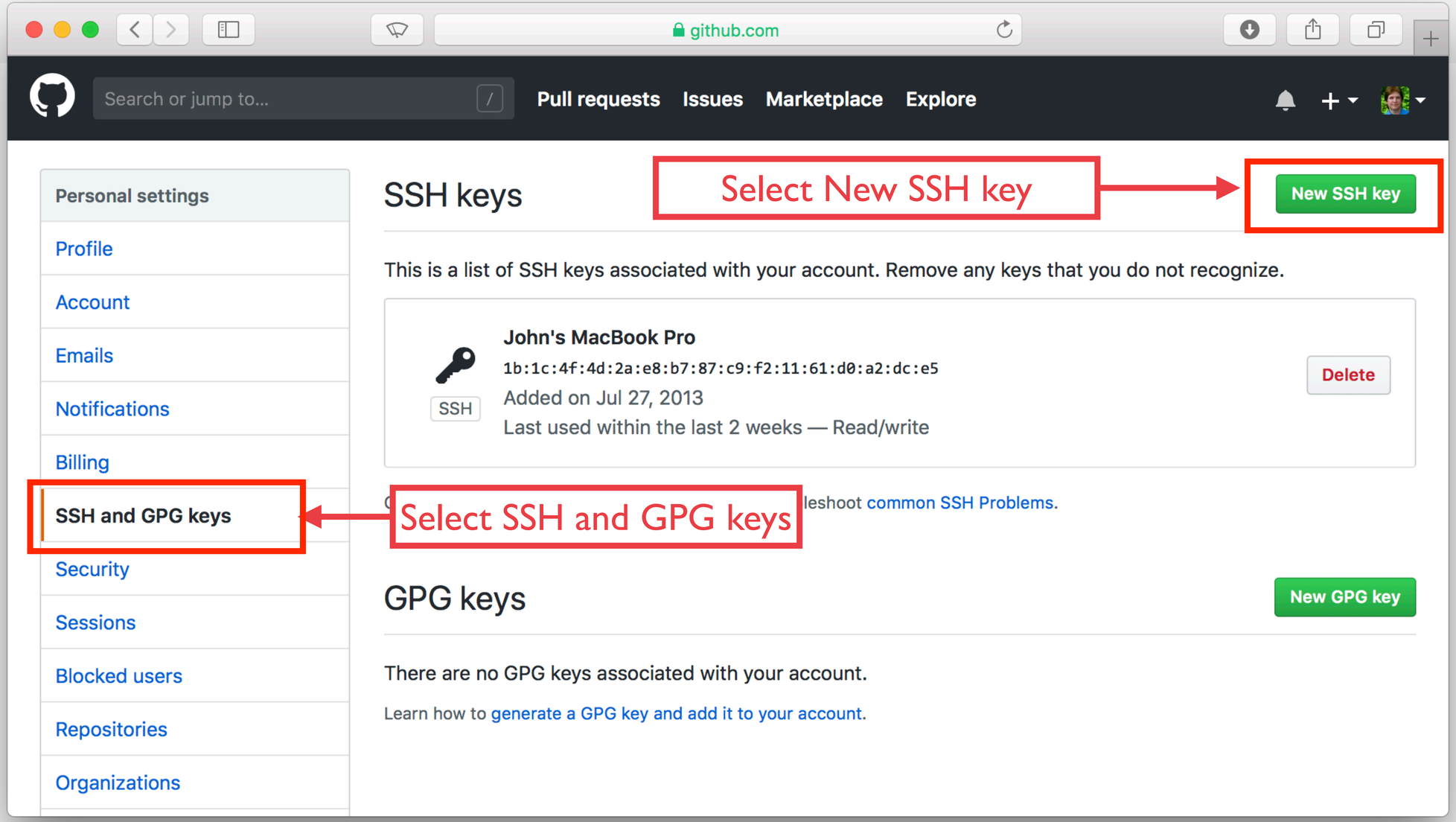
Add SSH key to GitHub



New SSH key


github.com
Search or jump to...
Pull requests
Issues
Marketplace
Explore
Personal settings
Profile
Account
Emails
Notifications
Billing
SSH and GPG keys
Security
Sessions
Blocked users
Repositories
Organizations
SSH keys
New SSH key
This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.
John's MacBook Pro
1b:1c:4f:4d:2a:e8:b7:87:c9:f2:11:61:d0:a2:dc:e5
SSH
Added on Jul 27, 2013
Last used within the last 2 weeks — Read/write
Delete
Select SSH and GPG keys
leshoot common SSH Problems.
GPG keys
New GPG key
There are no GPG keys associated with your account.
Learn how to generate a GPG key and add it to your account.

New SSH key

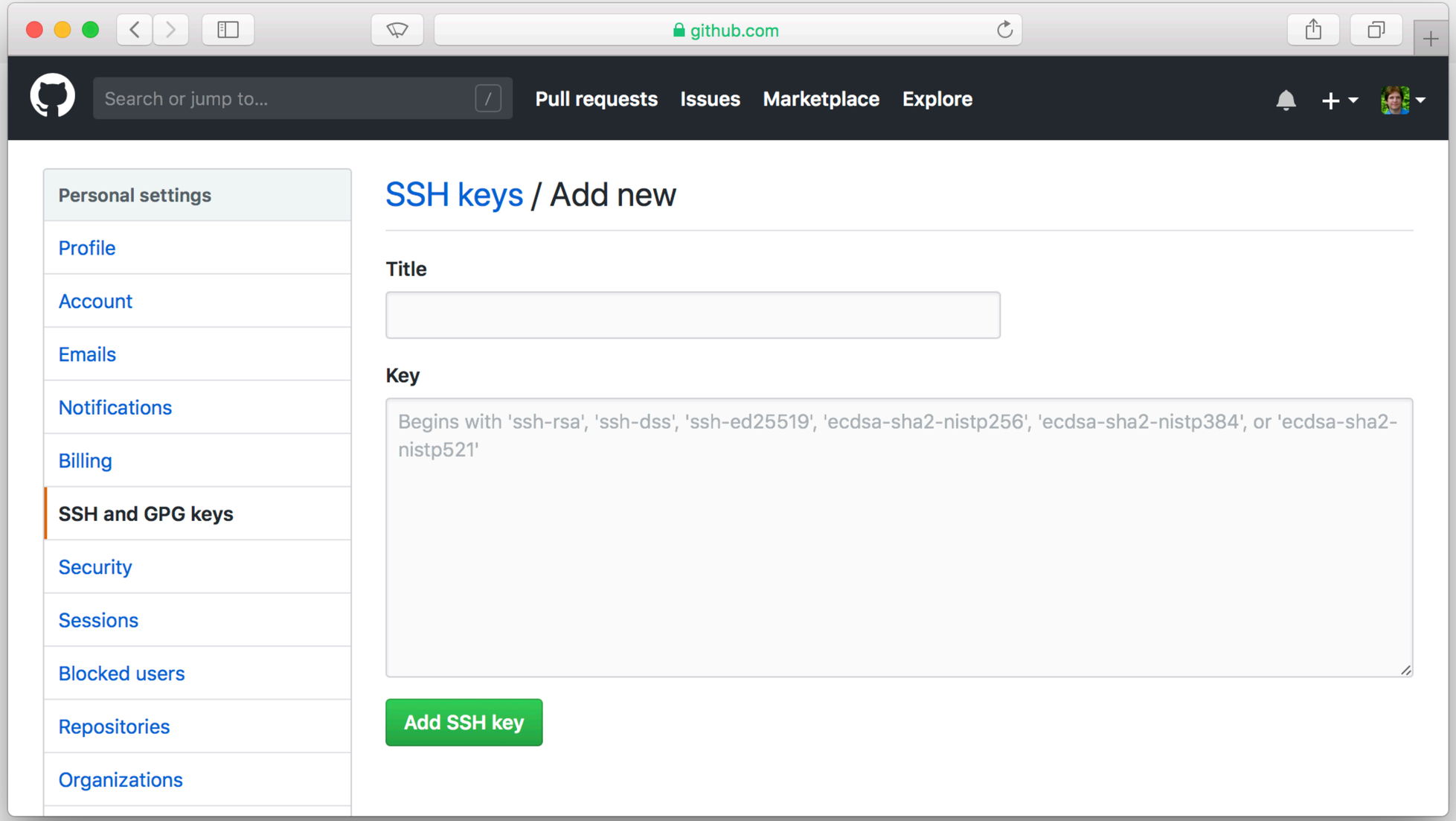

github.com
Search or jump to...
Pull requests
Issues
Marketplace
Explore
Personal settings
Profile
Account
Emails
Notifications
Billing
SSH and GPG keys
Security
Sessions
Blocked users
Repositories
Organizations
SSH keys
Select New SSH key
New SSH key
This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.
John's MacBook Pro
1b:1c:4f:4d:2a:e8:b7:87:c9:f2:11:61:d0:a2:dc:e5
SSH
Added on Jul 27, 2013
Last used within the last 2 weeks — Read/write
Delete
Select SSH and GPG keys
leshoot common SSH Problems.
GPG keys
New GPG key
There are no GPG keys associated with your account.
Learn how to generate a GPG key and add it to your account.

Copy Key to Clipboard

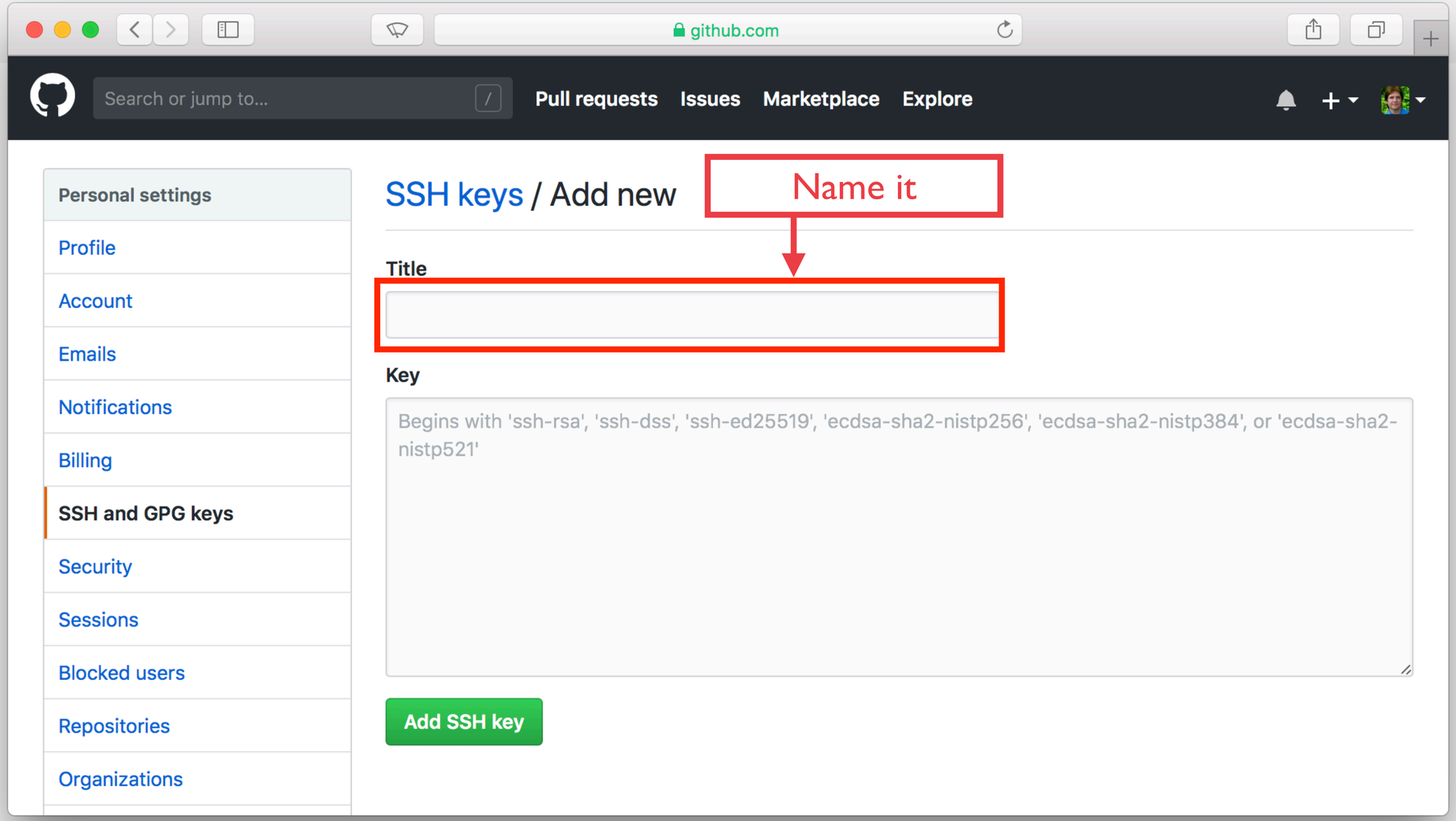
- Copy the key to your clipboard to paste into GitHub

```
$ pbcopy < ~/.ssh/id_rsa.pub
```

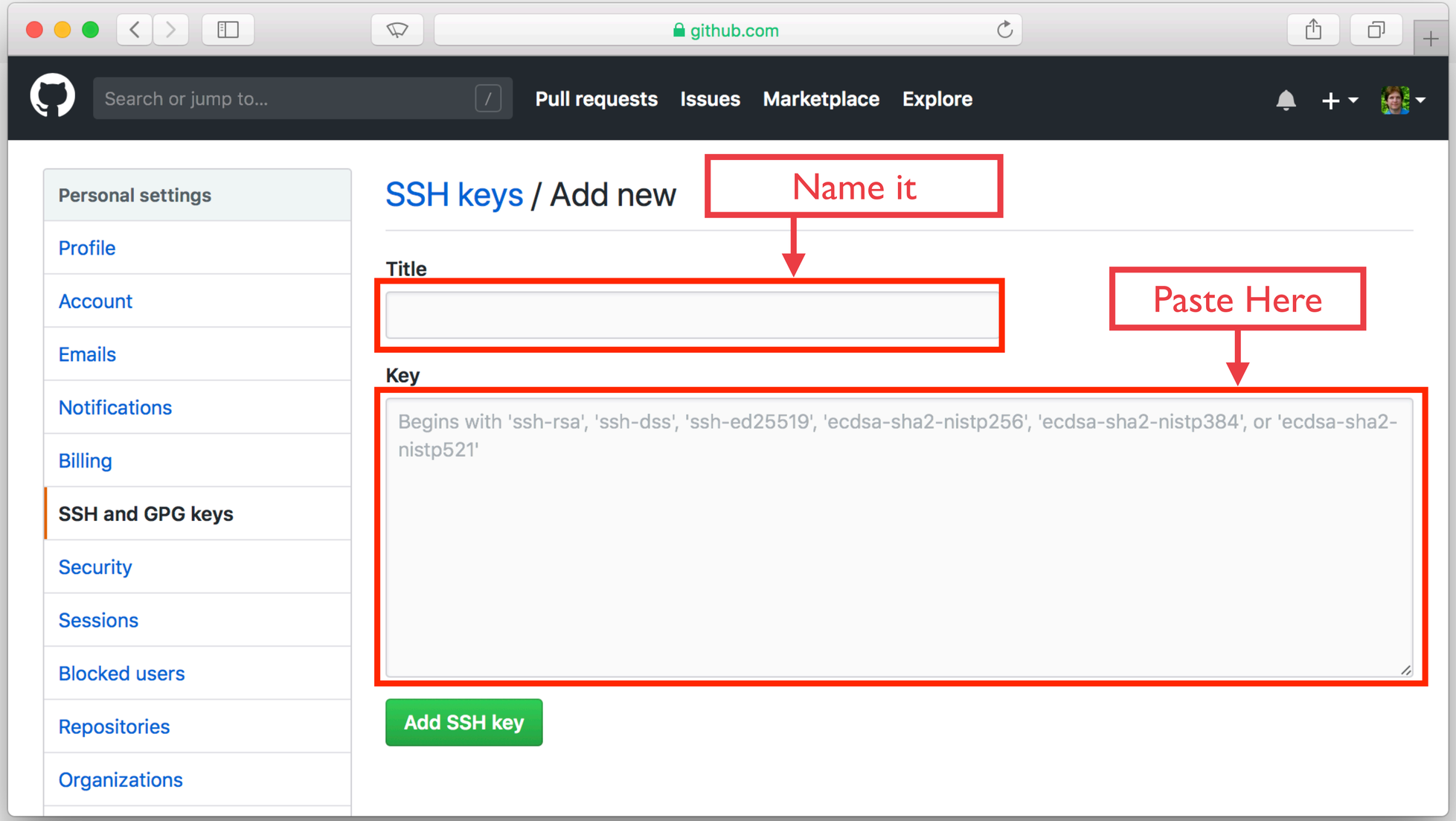
Add SSH key


github.com
Search or jump to...
/
Pull requests
Issues
Marketplace
Explore
Personal settings
Profile
Account
Emails
Notifications
Billing
SSH and GPG keys
Security
Sessions
Blocked users
Repositories
Organizations
SSH keys / Add new
Title
Key
Begins with 'ssh-rsa', 'ssh-dss', 'ssh-ed25519', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384', or 'ecdsa-sha2-nistp521'
Add SSH key

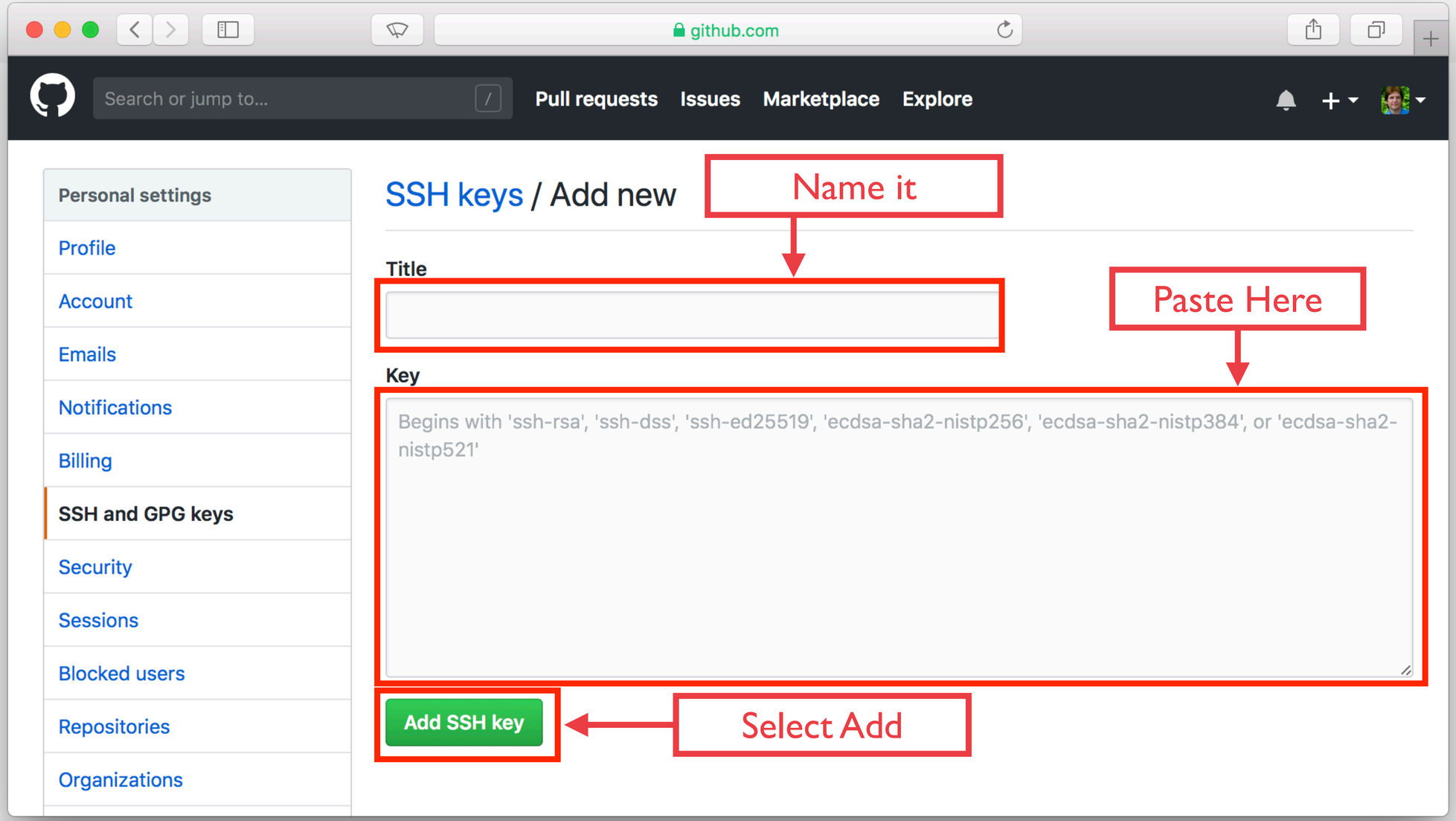
Add SSH key


github.com
Search or jump to...
Pull requests
Issues
Marketplace
Explore
Personal settings
Profile
Account
Emails
Notifications
Billing
SSH and GPG keys
Security
Sessions
Blocked users
Repositories
Organizations
SSH keys / Add new
Name it
Title
Key
Begins with 'ssh-rsa', 'ssh-dss', 'ssh-ed25519', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384', or 'ecdsa-sha2-nistp521'
Add SSH key

Add SSH key


github.com
Search or jump to...
Pull requests
Issues
Marketplace
Explore
Personal settings
Profile
Account
Emails
Notifications
Billing
SSH and GPG keys
Security
Sessions
Blocked users
Repositories
Organizations
SSH keys / Add new
Name it
Title
Paste Here
Key
Begins with 'ssh-rsa', 'ssh-dss', 'ssh-ed25519', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384', or 'ecdsa-sha2-nistp521'
Add SSH key

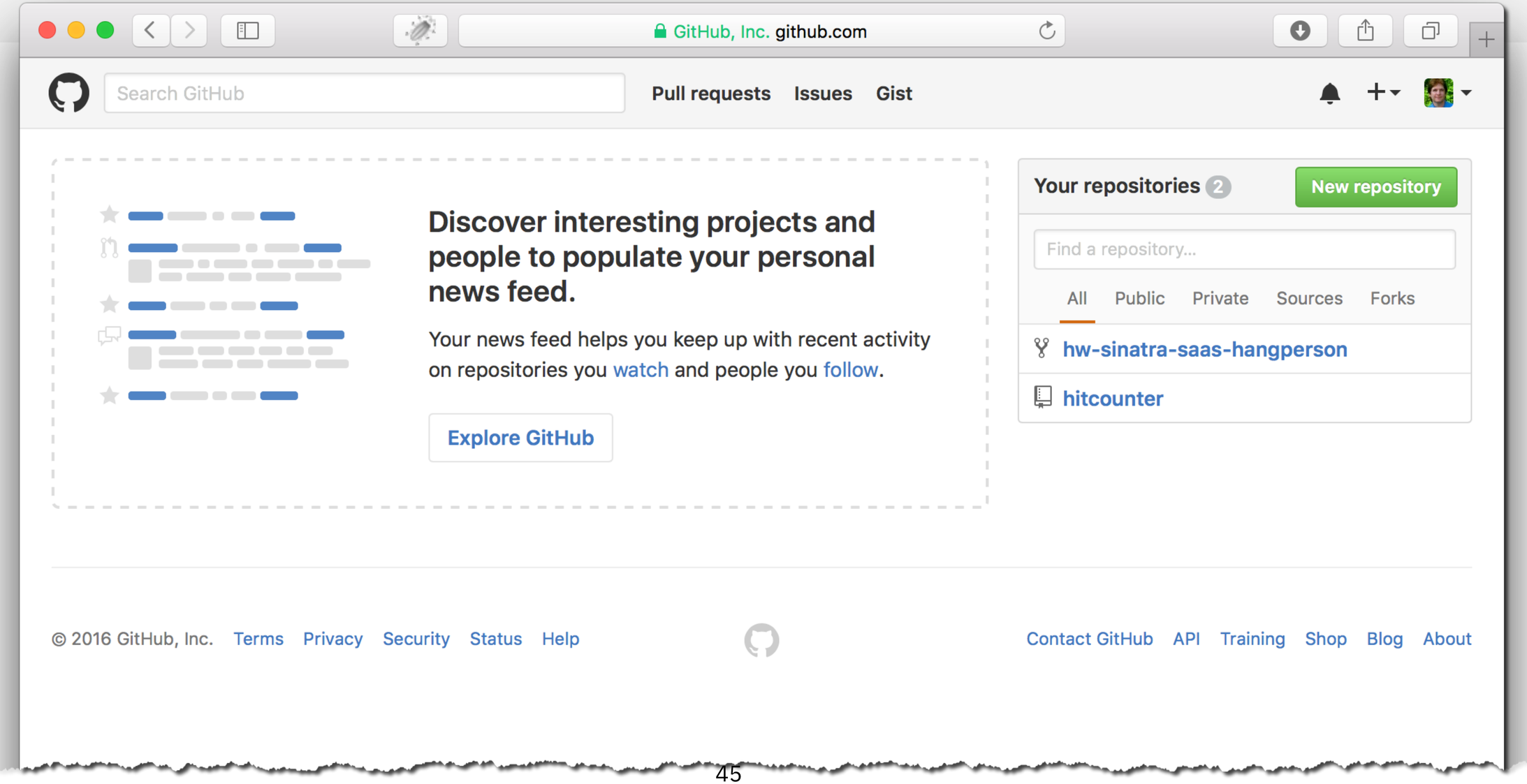
Add SSH key


github.com
Search or jump to...
Pull requests
Issues
Marketplace
Explore
Personal settings
Profile
Account
Emails
Notifications
Billing
SSH and GPG keys
Security
Sessions
Blocked users
Repositories
Organizations
SSH keys / Add new
Name it
Title
Paste Here
Key
Begins with 'ssh-rsa', 'ssh-dss', 'ssh-ed25519', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384', or 'ecdsa-sha2-nistp521'
Add SSH key
Select Add

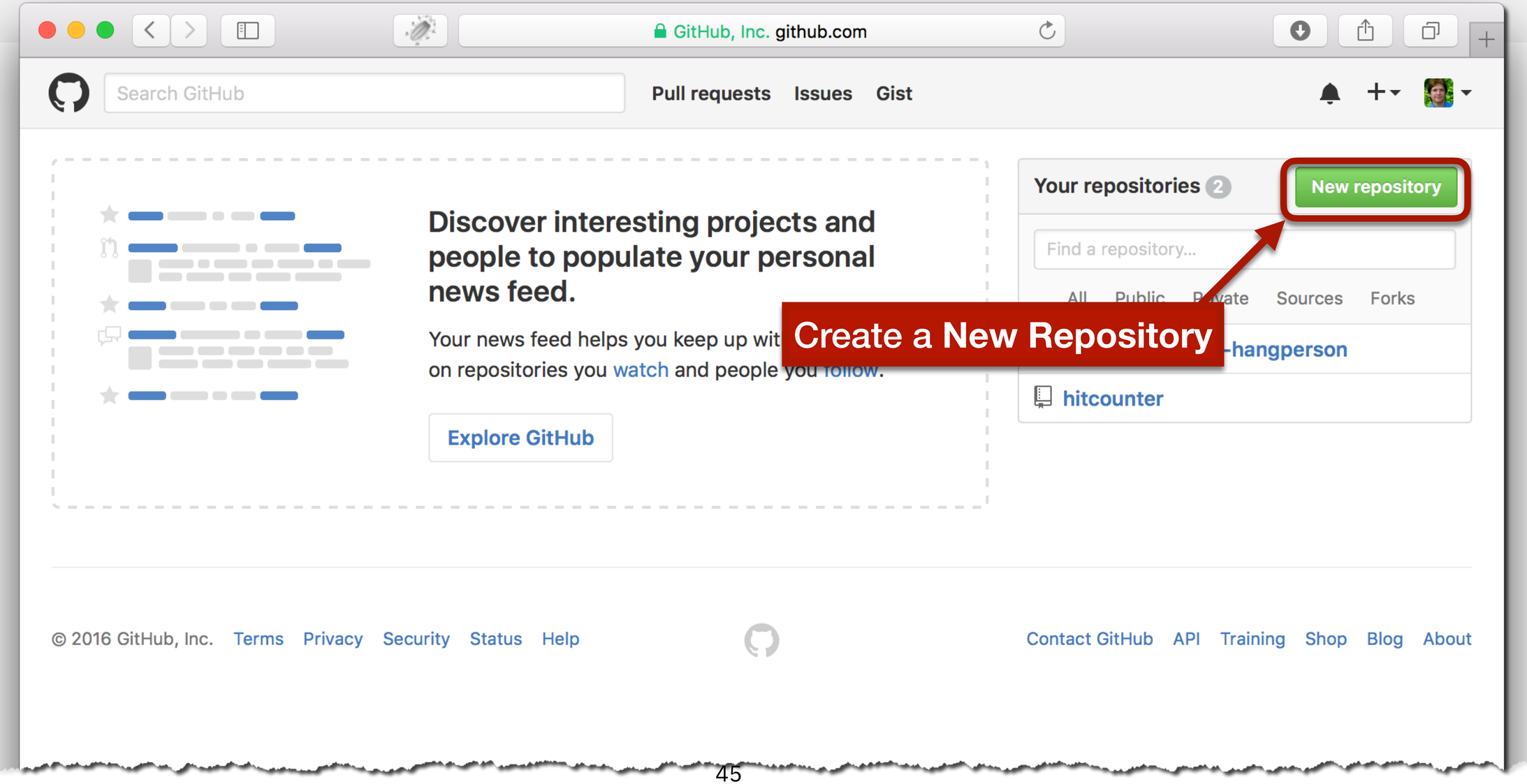
CREATE and FORK

- The Git Repository holds your code, documentation, etc.
- You need to CREATE one at github.com for new work or...
- Find one you want to contribute to and FORK it
- You only need to FORK a repo if you plan to contribute back
  - Otherwise you can just CLONE it

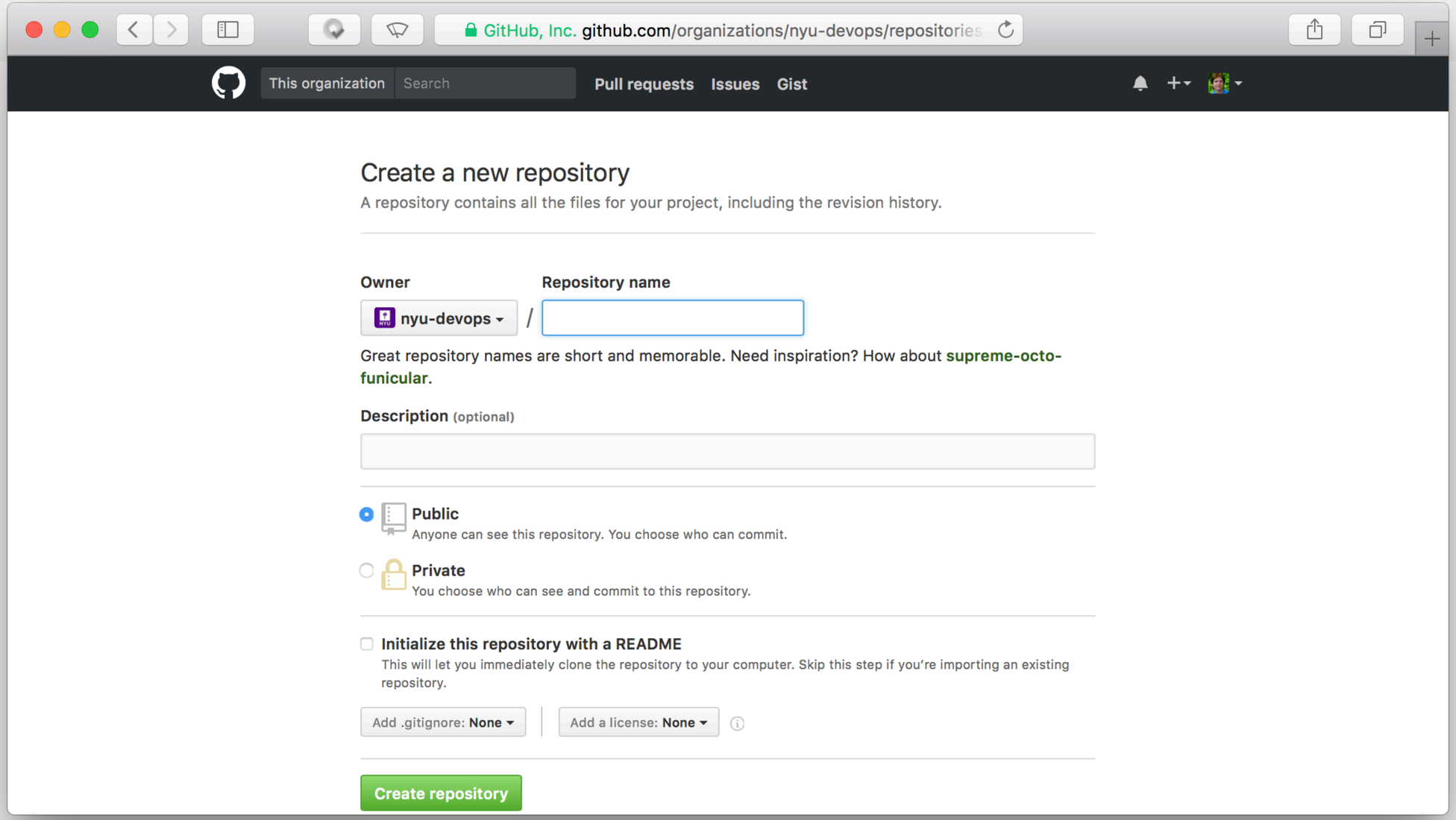
Create a New Repository



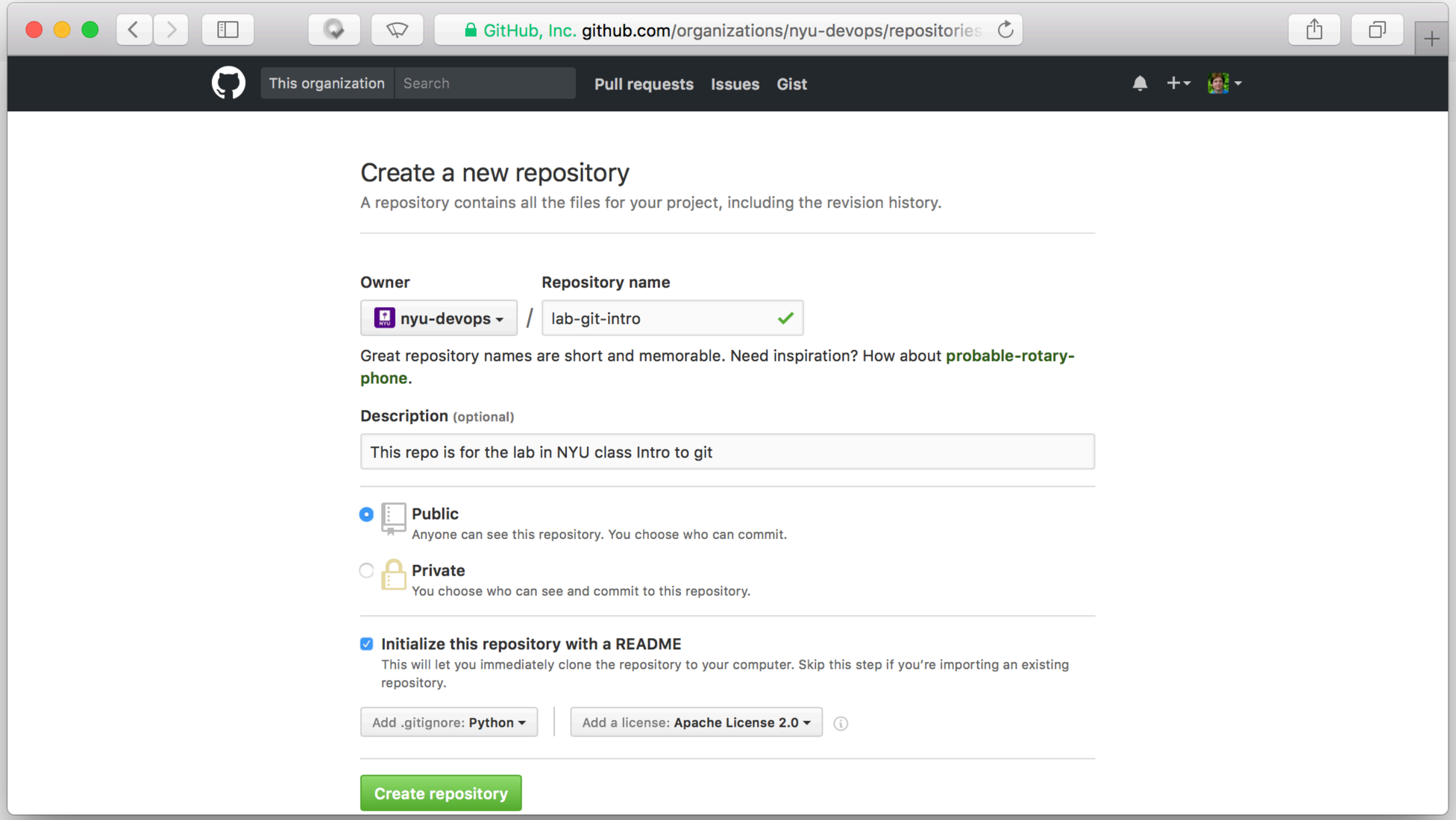
Create a New Repository



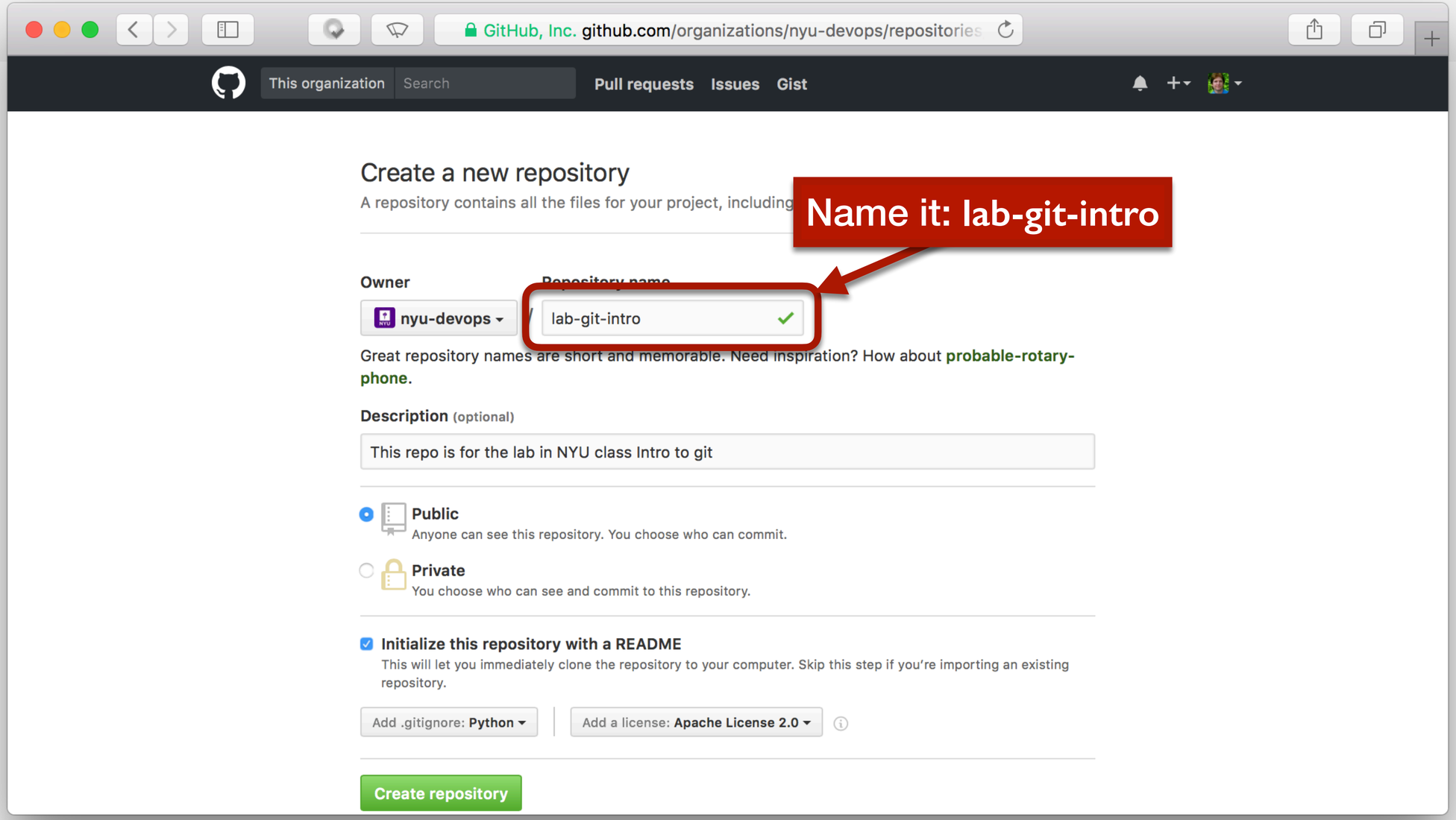
New Repository Page



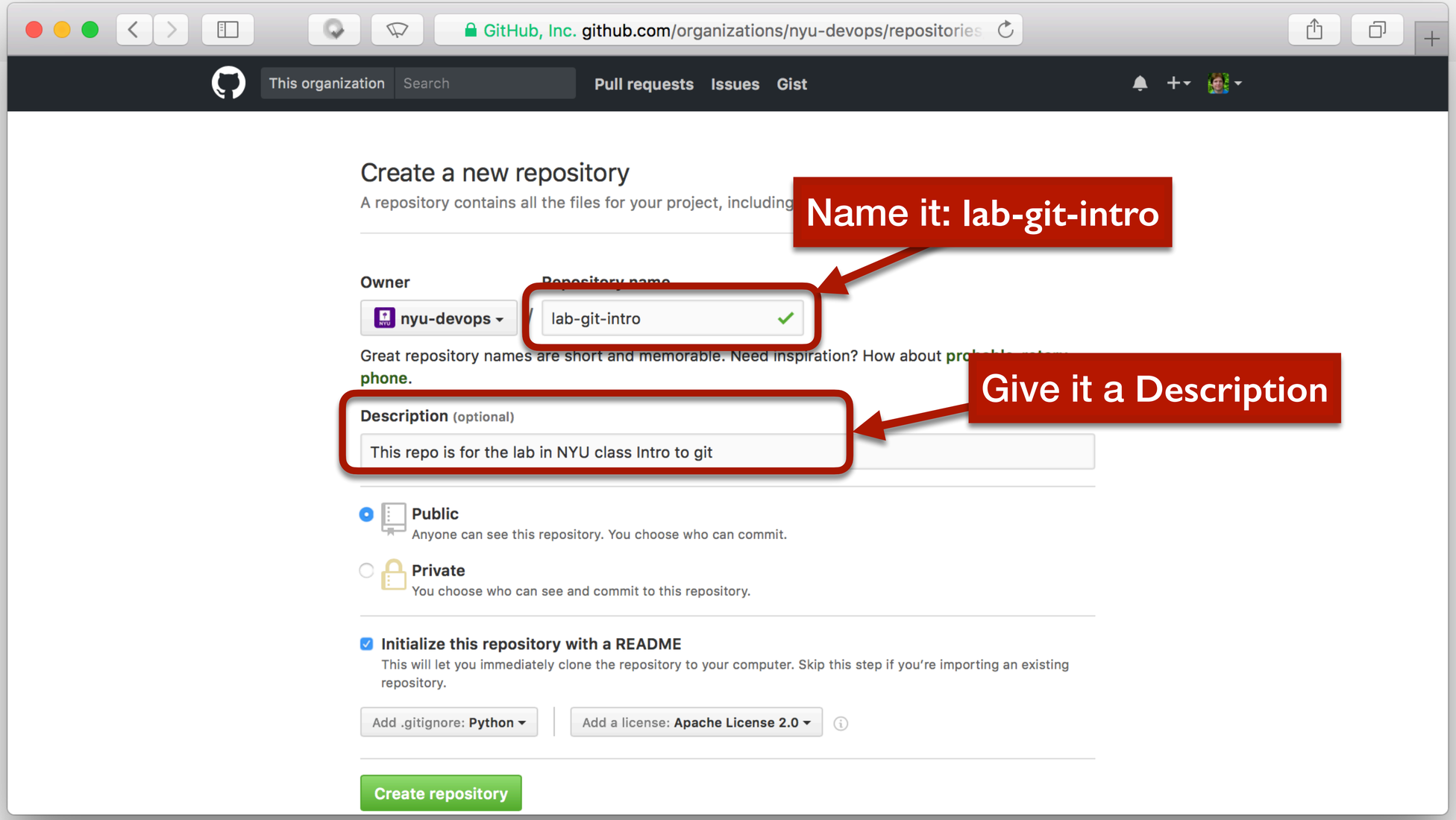
New Repository Filled In



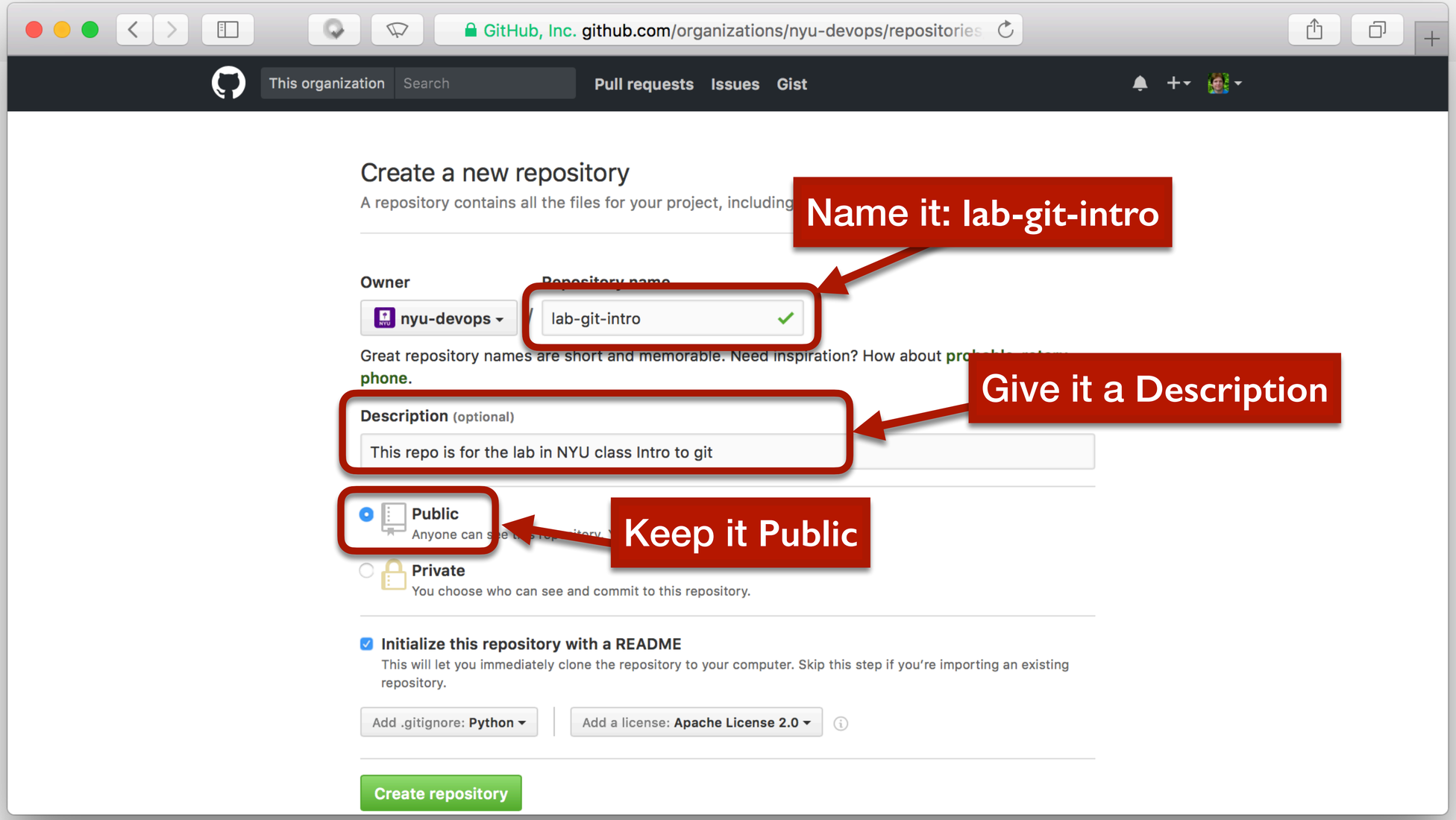
New Repository Filled In



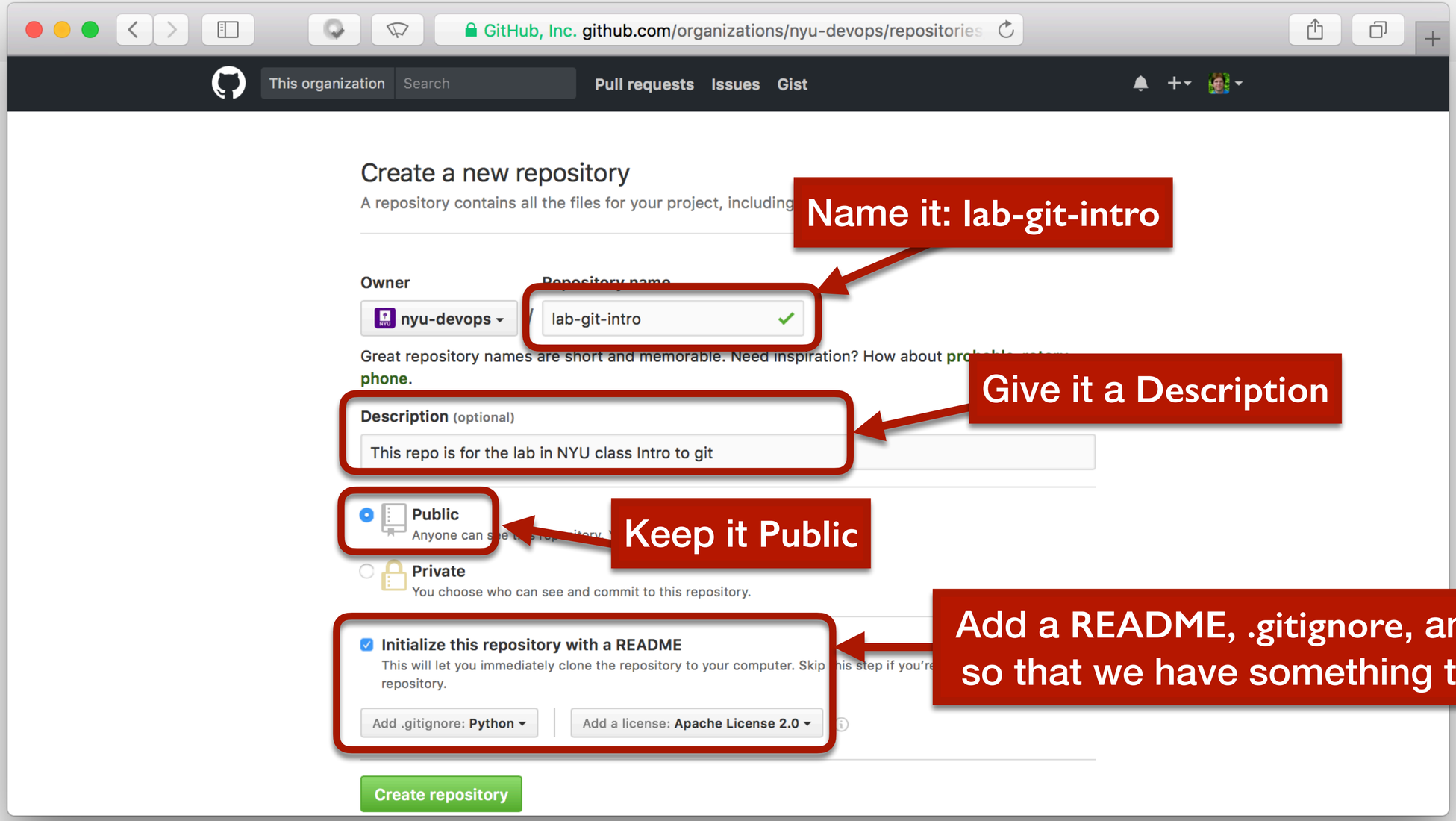
New Repository Filled In



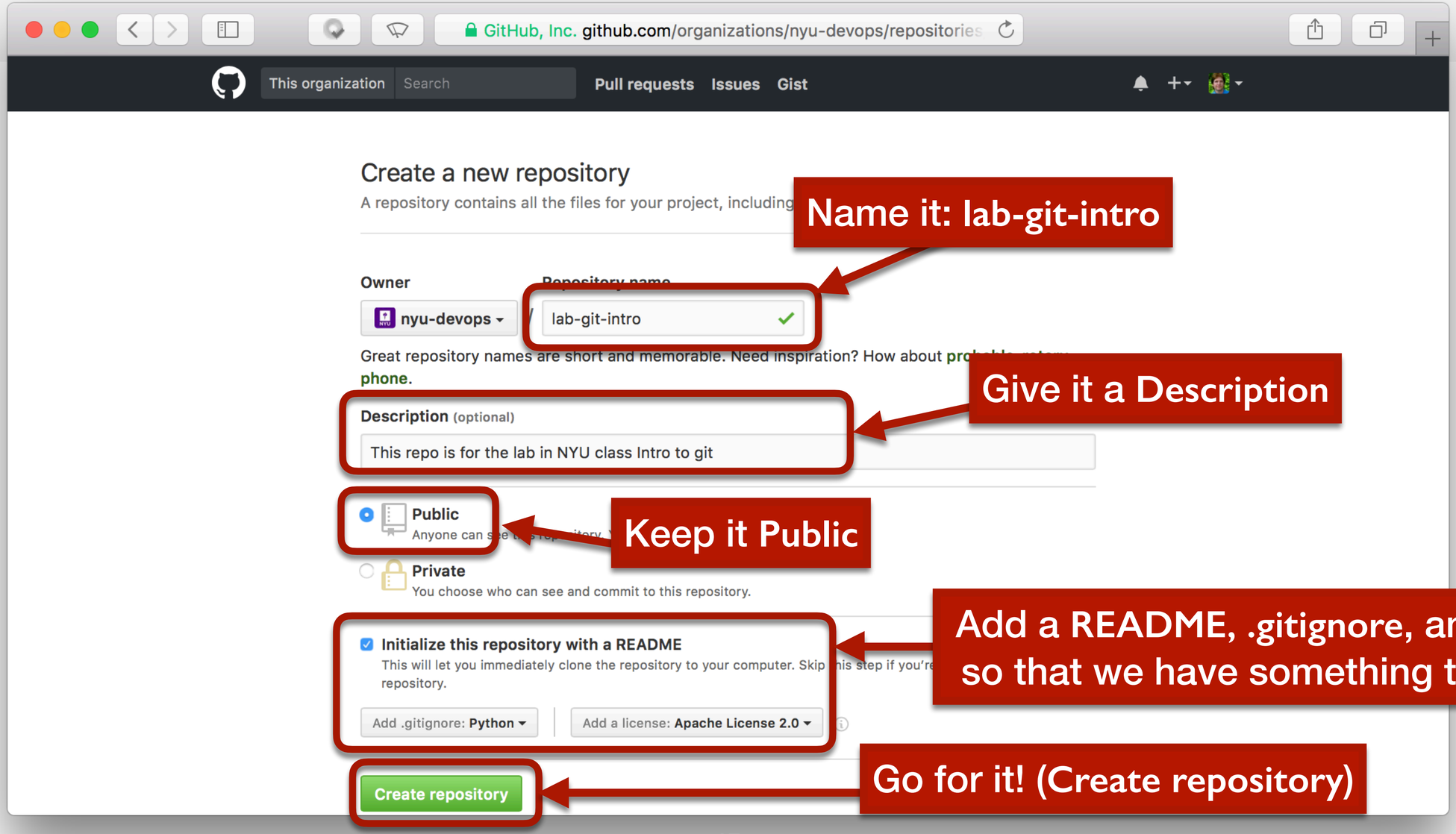
New Repository Filled In



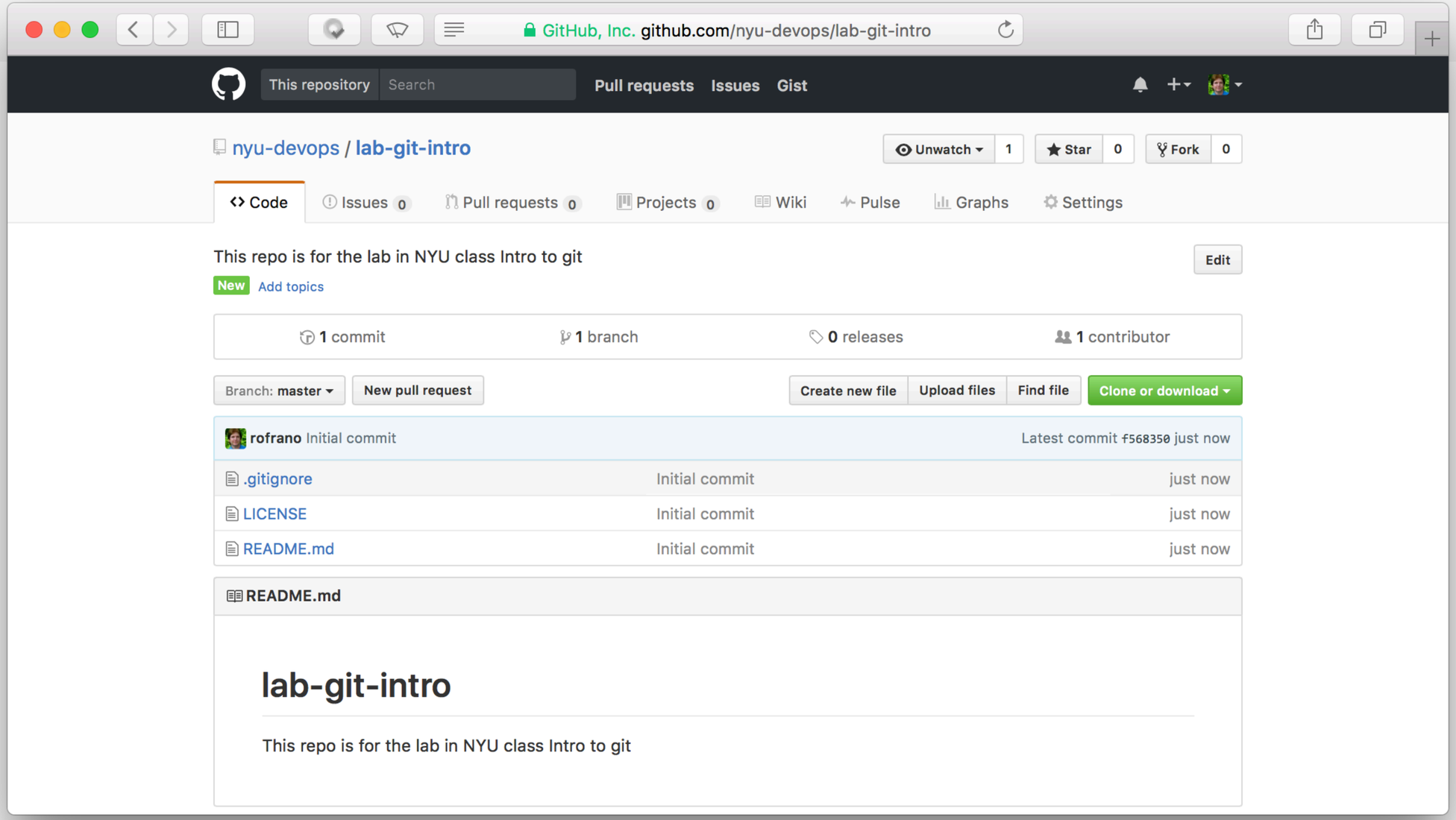
New Repository Filled In



New Repository Filled In



You Now Have A Repository



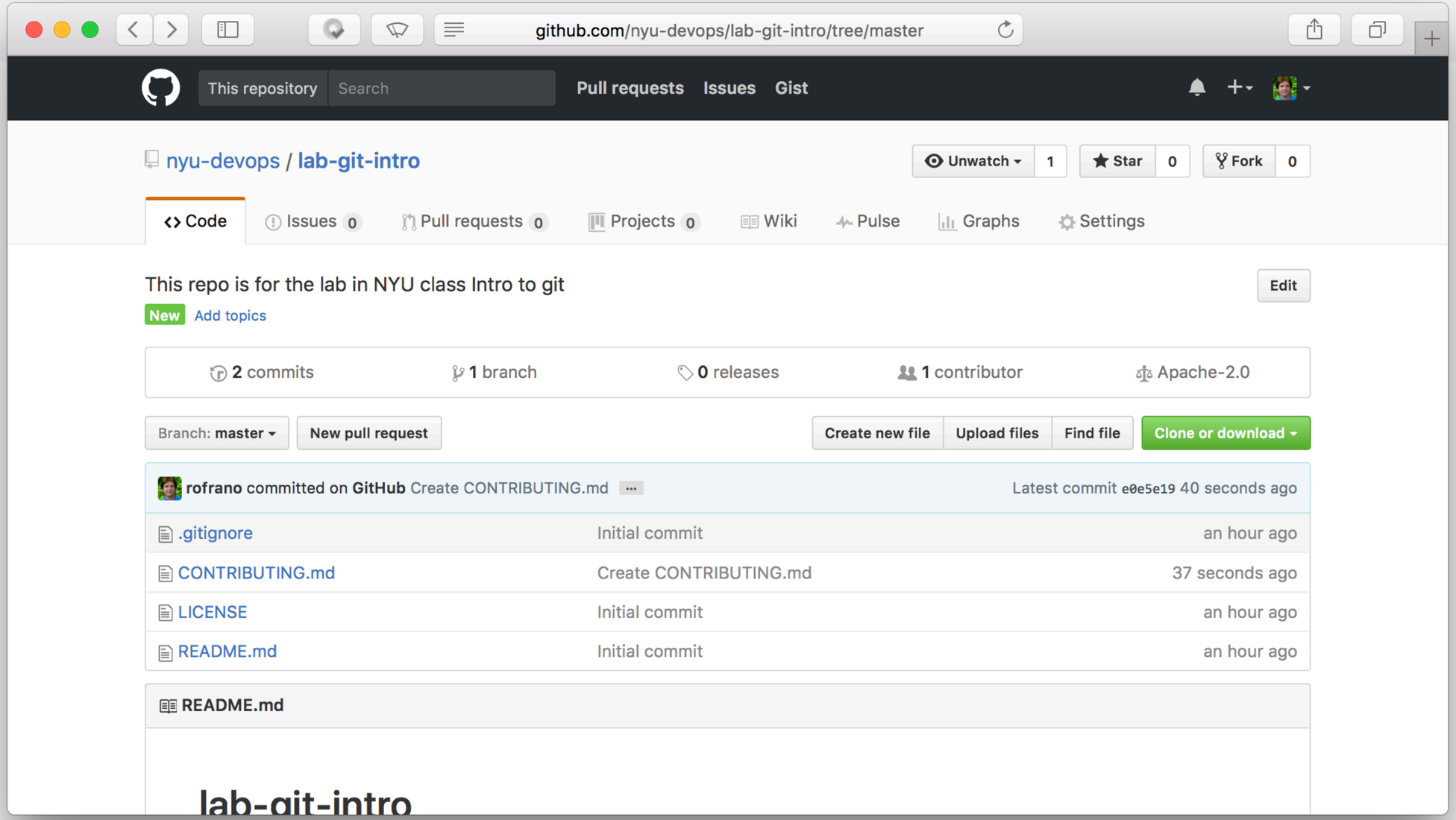
Guidelines For Contributing

- To help your project contributors do good work, you can add a file with contribution guidelines to the root of your project's repository
- Then, whenever someone opens a pull request or creates an issue, they will see a link to that file.
- Good Social Coding requires that everyone know and follow the guidelines.

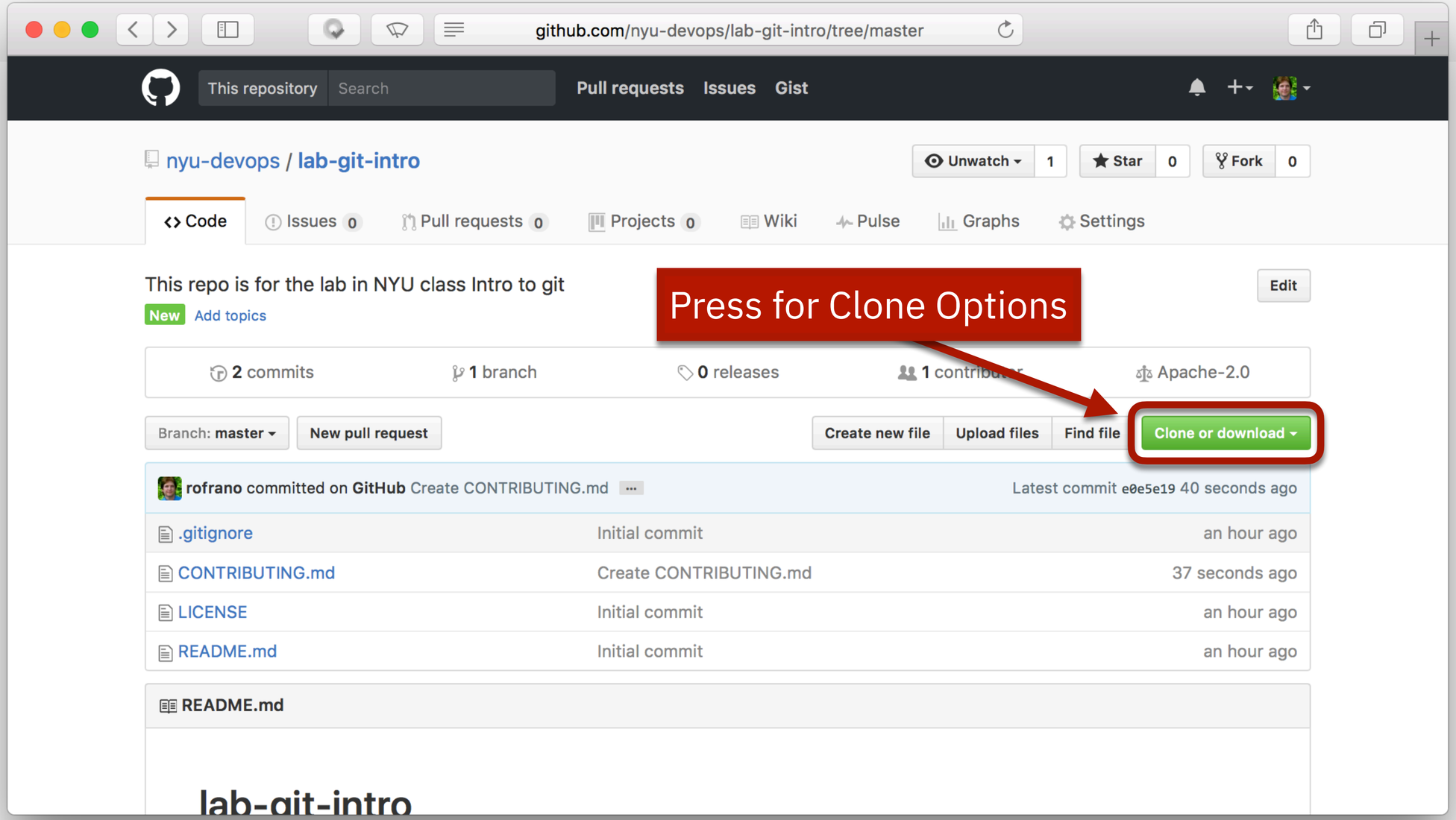
Consistent Contributions

- As you can imagine, with outside contributors there must be coding standards
- Use a `CONTRIBUTION.md` file to document how to contribute
- It’s a good idea to publish your **coding standards** on your GitHub Wiki or other public place and point to them in your `CONTRIBUTION.md` file.
- When a Pull Request is made, you should also check that the standards have been followed

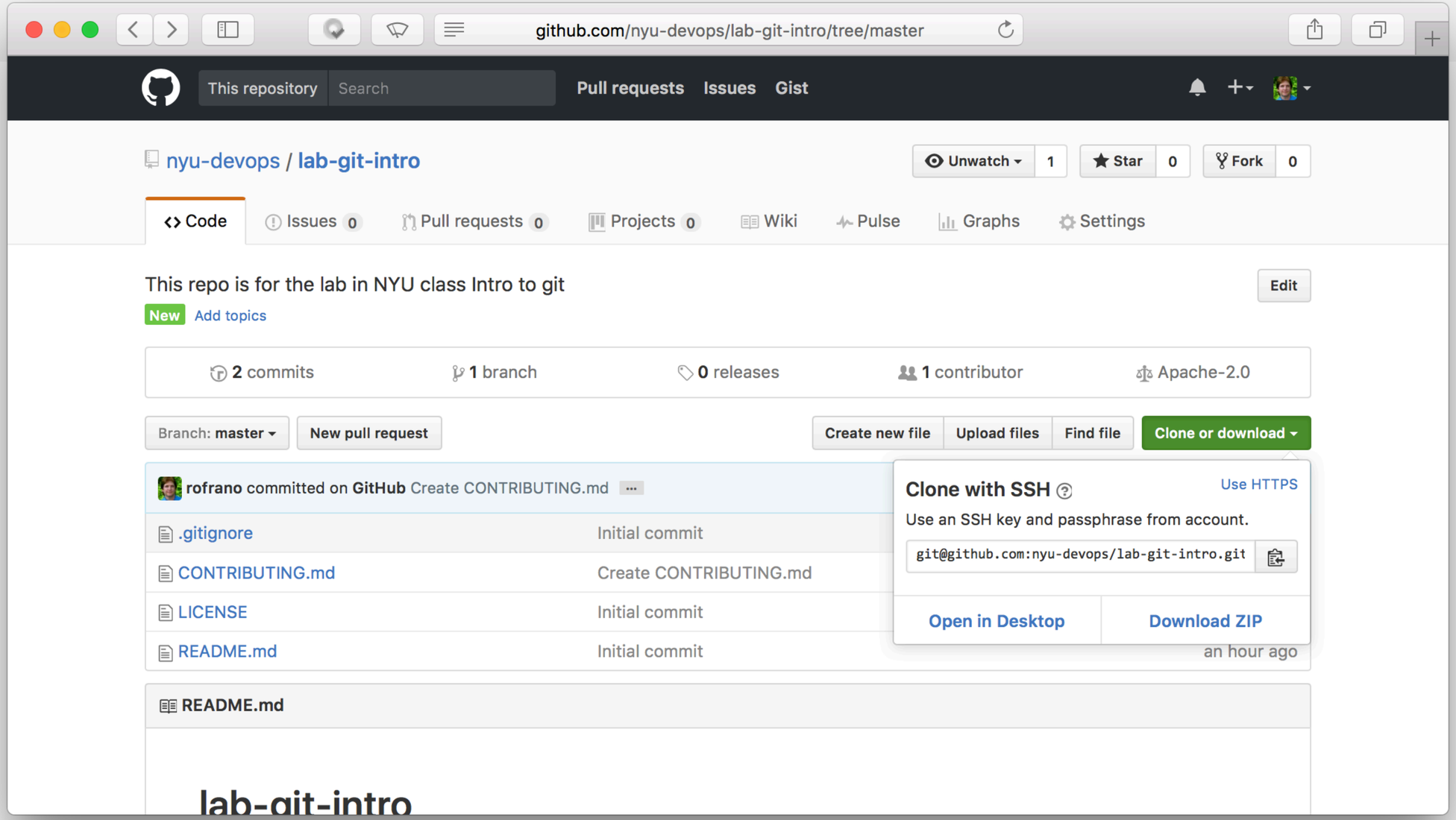
Clone Options



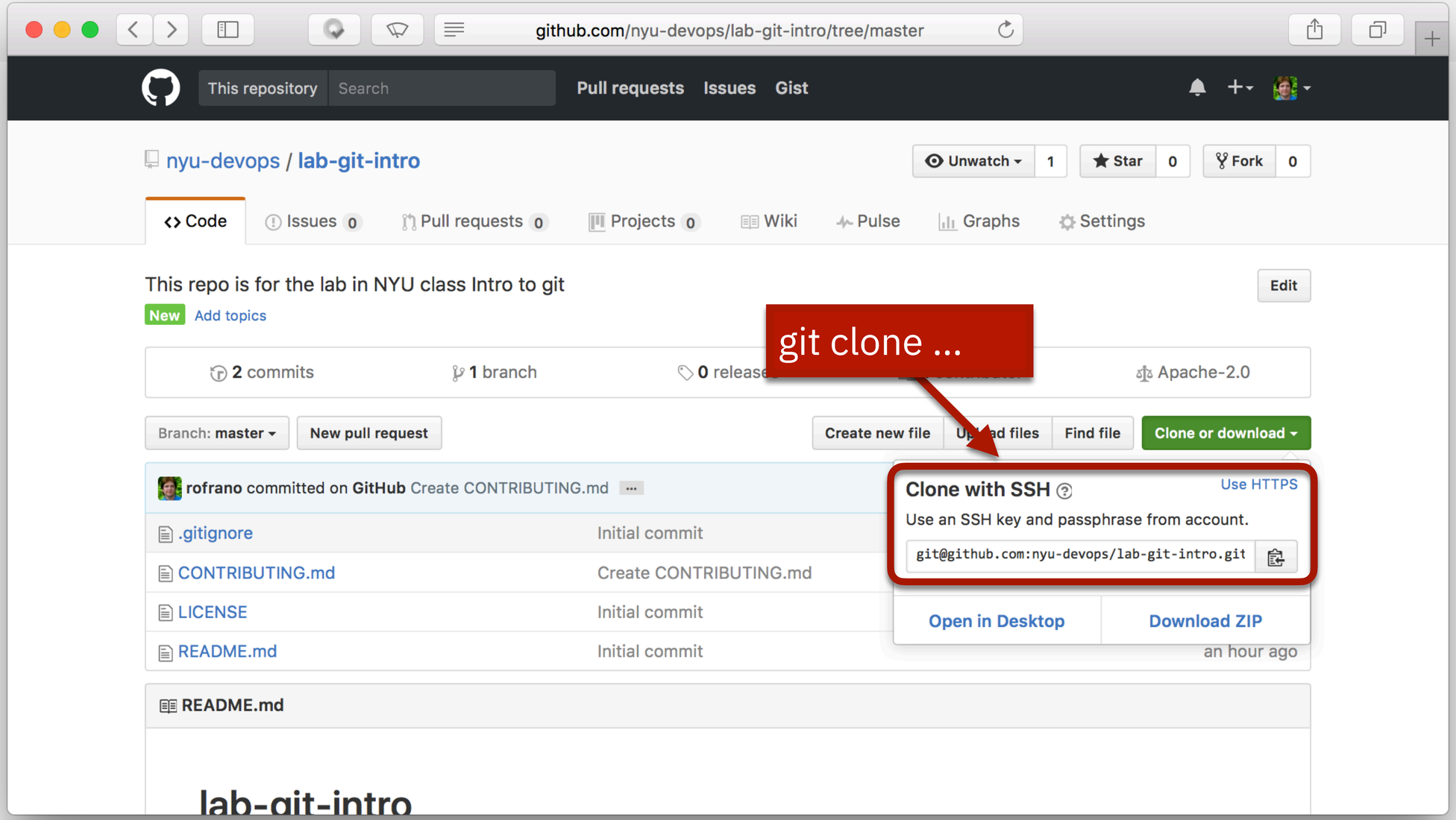
Clone Options



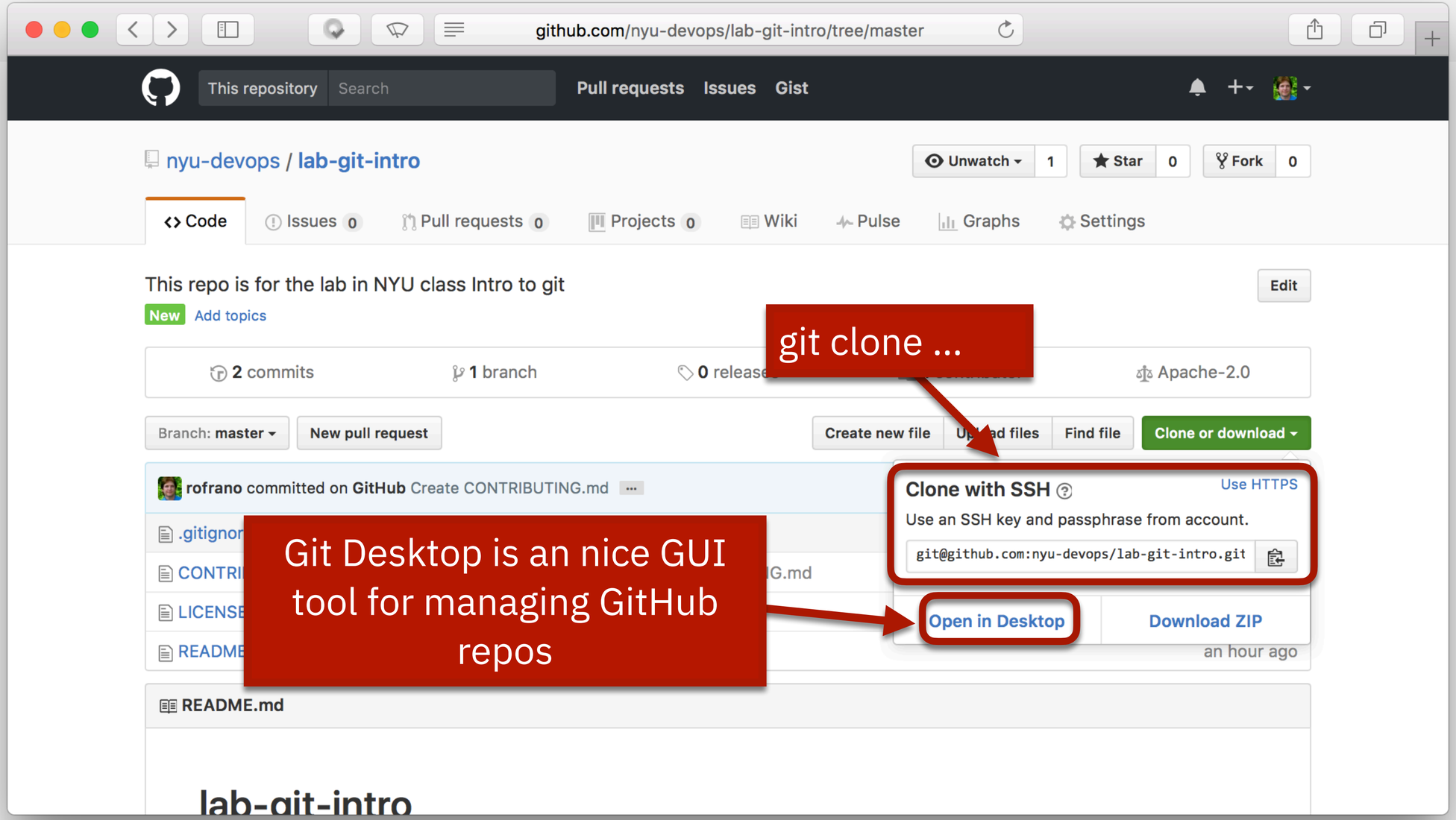
Clone Your Repo



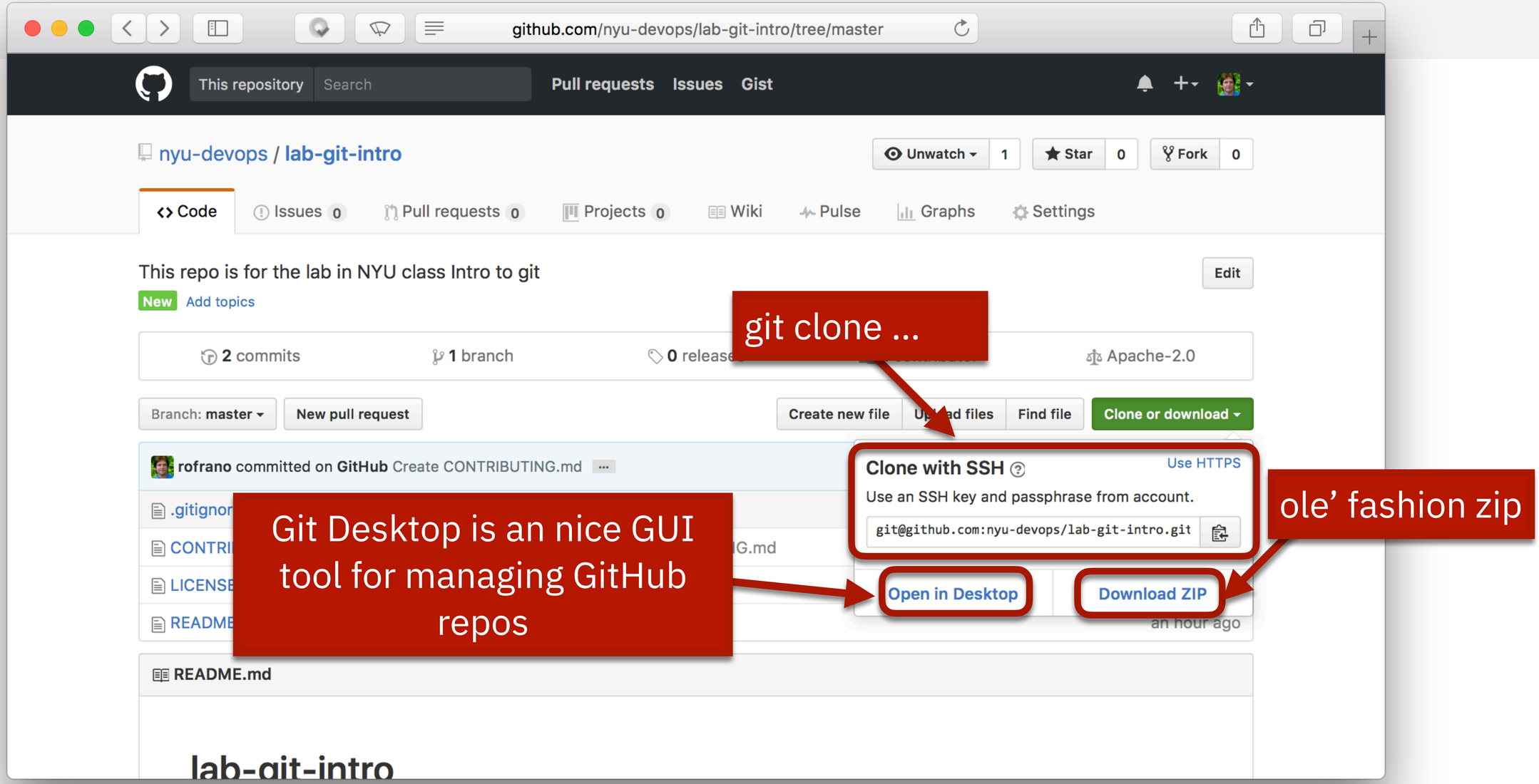
Clone Your Repo



Clone Your Repo



Clone Your Repo



CLONE

- Let’s CLONE the repository to get a local copy

```
$ git clone git@github.com:rofrano/nyu-lab-git-intro.git

\
```

CLONE

- Cloning copies the entire repository locally

```
$ git clone git@github.com:rofrano/nyu-lab-git-intro.git
Cloning into 'nyu-lab-git-intro'...
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), 4.79 KiB | 0 bytes/s, done.
Checking connectivity... done.
$
```

CLONE

- Change into the cloned directory

```
$ git clone git@github.com:rofrano/nyu-lab-git-intro.git
Cloning into 'nyu-lab-git-intro'...
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), 4.79 KiB | 0 bytes/s, done.
Checking connectivity... done.
$ cd nyu-lab-git-intro/
$
```

CLONE

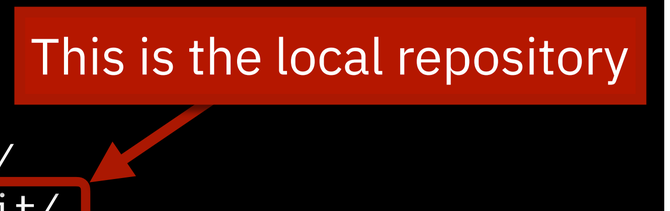
- List the files to see what was cloned

```
$ git clone git@github.com:rofrano/nyu-lab-git-intro.git
Cloning into 'nyu-lab-git-intro'...
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), 4.79 KiB | 0 bytes/s, done.
Checking connectivity... done.
$ cd nyu-lab-git-intro/
$ ls -la
total 40
drwxr-xr-x   6 rofrano  staff   204B Sep 12 15:46 ./
drwxr-xr-x   3 rofrano  staff   102B Sep 12 15:46 ../
drwxr-xr-x  13 rofrano  staff   442B Sep 12 15:46 .git/
-rw-r--r--   1 rofrano  staff   1.0K Sep 12 15:46 .gitignore
-rw-r--r--   1 rofrano  staff    11K Sep 12 15:46 LICENSE
-rw-r--r--   1 rofrano  staff    71B Sep 12 15:46 README.md
$
```

CLONE

- List the files to see what was cloned

```
$ git clone git@github.com:rofrano/nyu-lab-git-intro.git
Cloning into 'nyu-lab-git-intro'...
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), 4.79 KiB | 0 bytes/s, done.
Checking connectivity... done.
$ cd nyu-lab-git-intro/
$ ls -la
total 40
drwxr-xr-x   6 rofrano  staff   204B Sep 12 15:46 ./
drwxr-xr-x   3 rofrano  staff   102B Sep 12 15:46 ../
drwxr-xr-x  13 rofrano  staff   442B Sep 12 15:46 .git/
-rw-r--r--   1 rofrano  staff   1.0K Sep 12 15:46 .gitignore
-rw-r--r--   1 rofrano  staff    11K Sep 12 15:46 LICENSE
-rw-r--r--   1 rofrano  staff    71B Sep 12 15:46 README.md
$
```



CLONE

- List the files to see what was cloned

```
$ git clone git@github.com:rofrano/nyu-lab-git-intro.git
Cloning into 'nyu-lab-git-intro'...
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), 4.79 KiB | 0 bytes/s, done.
Checking connectivity... done.
$ cd nyu-lab-git-intro/
$ ls -la
total 40
drwxr-xr-x   6 rofrano  staff   204B Sep 12 15:46 ./
drwxr-xr-x   3 rofrano  staff   102B Sep 12 15:46 ../
drwxr-xr-x  13 rofrano  staff   442B Sep 12 15:46 .git/
-rw-r--r--   1 rofrano  staff   1.0K Sep 12 15:46 .gitignore
-rw-r--r--   1 rofrano  staff    11K Sep 12 15:46 LICENSE
-rw-r--r--   1 rofrano  staff    71B Sep 12 15:46 README.md
$
```

This is the local repository

These are the optional files we asked to have created

GIT STATUS is Your Friend

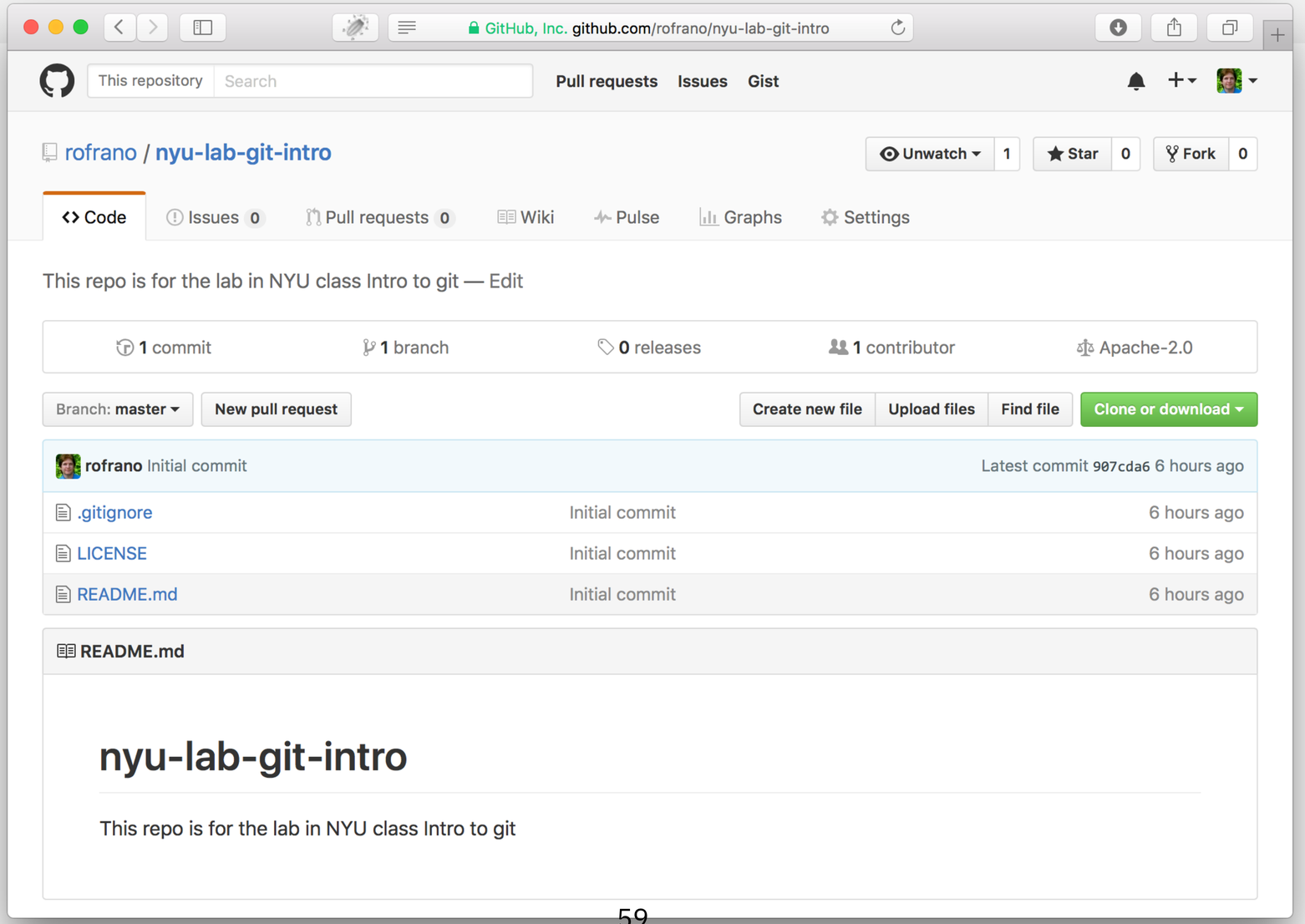
- Git Status will tell you the status of your work
- You'll be using it a lot!

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working directory clean
$
```

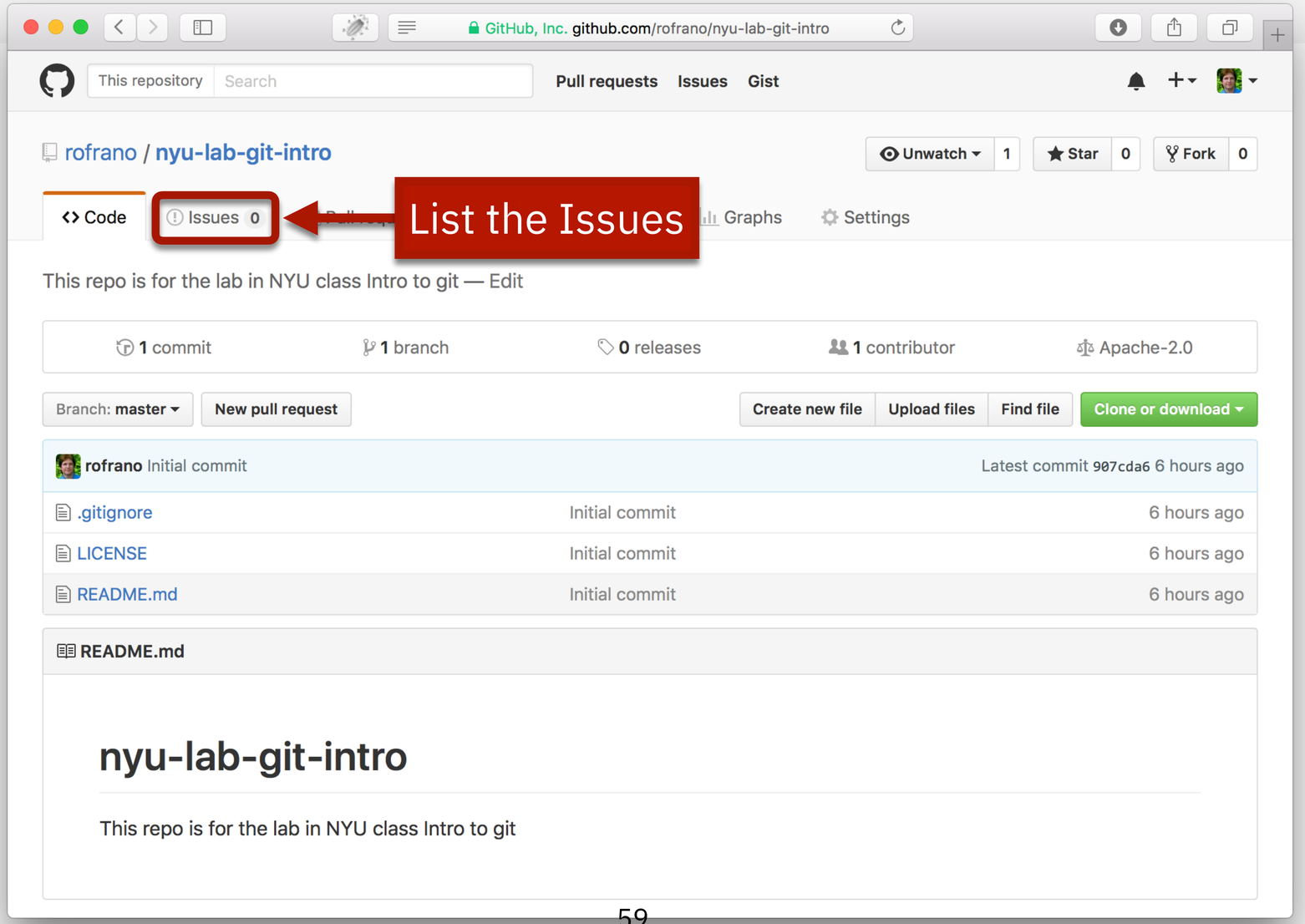
Issues

- ISSUES help you track what everyone is doing
- ISSUES can be features or defects or pull requests or ...
- You should not be working on code without an open ISSUE
- Always assign the ISSUE to yourself before working on it

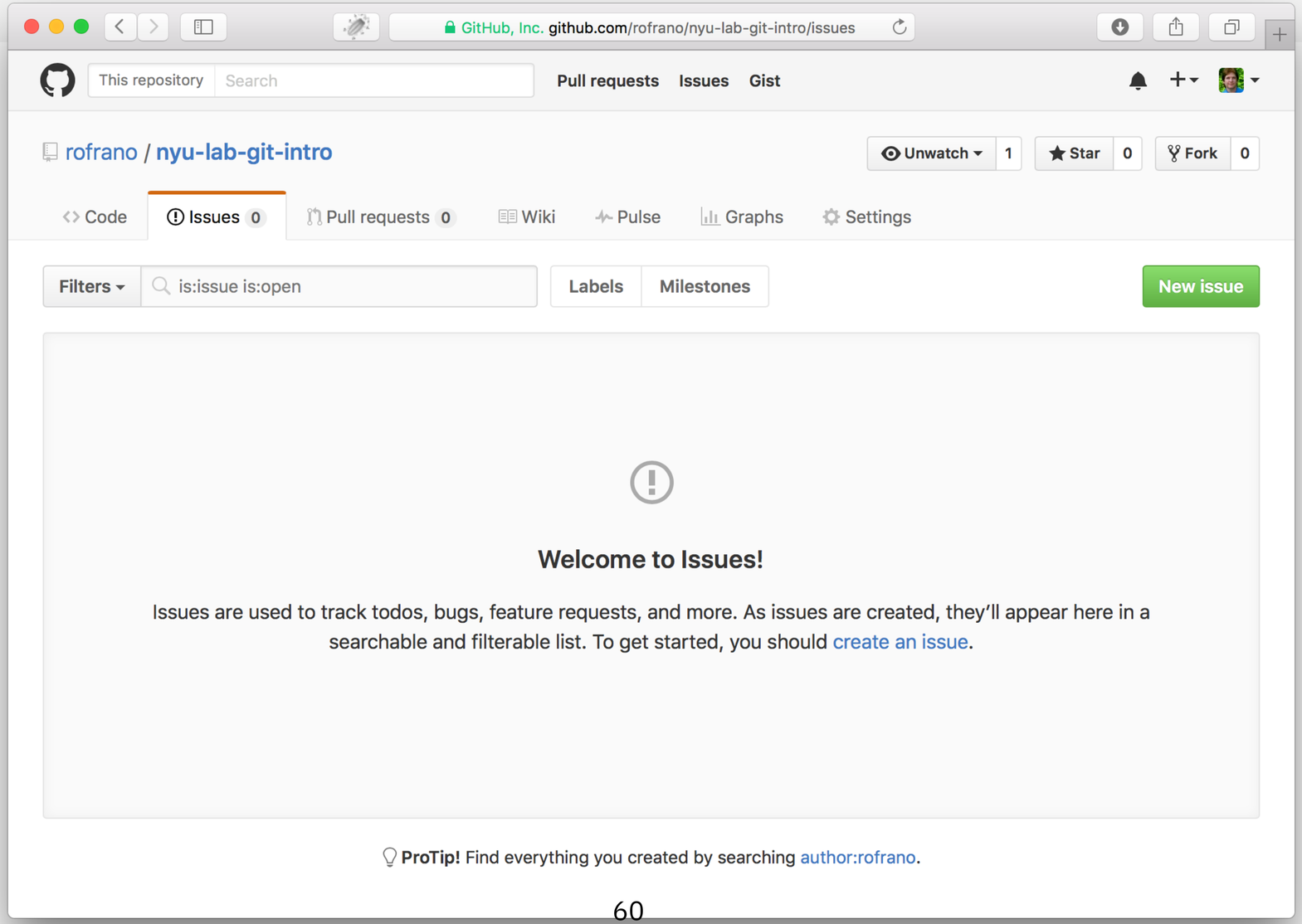
We Need an Issue to Work On



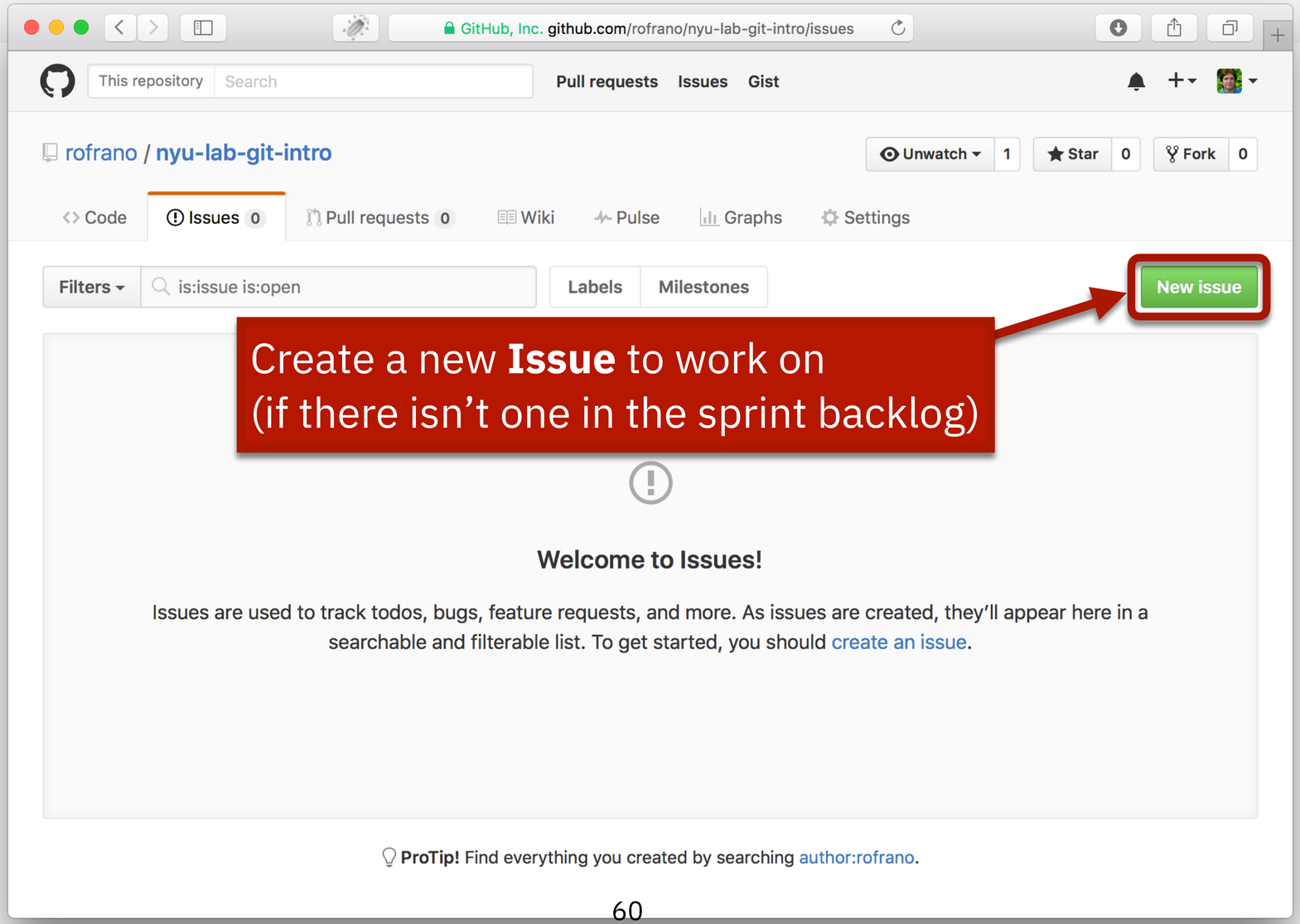
We Need an Issue to Work On



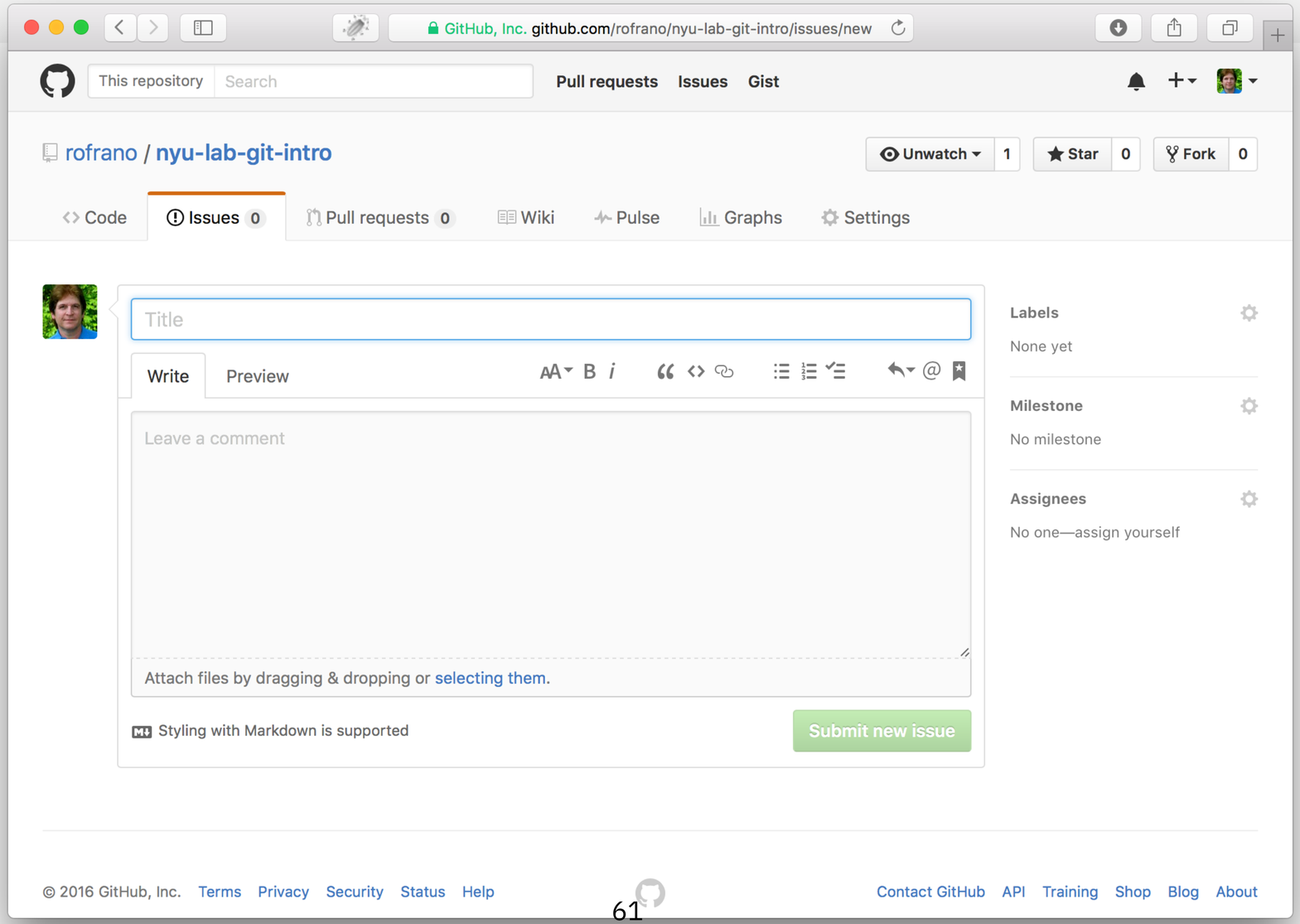
Create a New Issue



Create a New Issue



New Issue Page

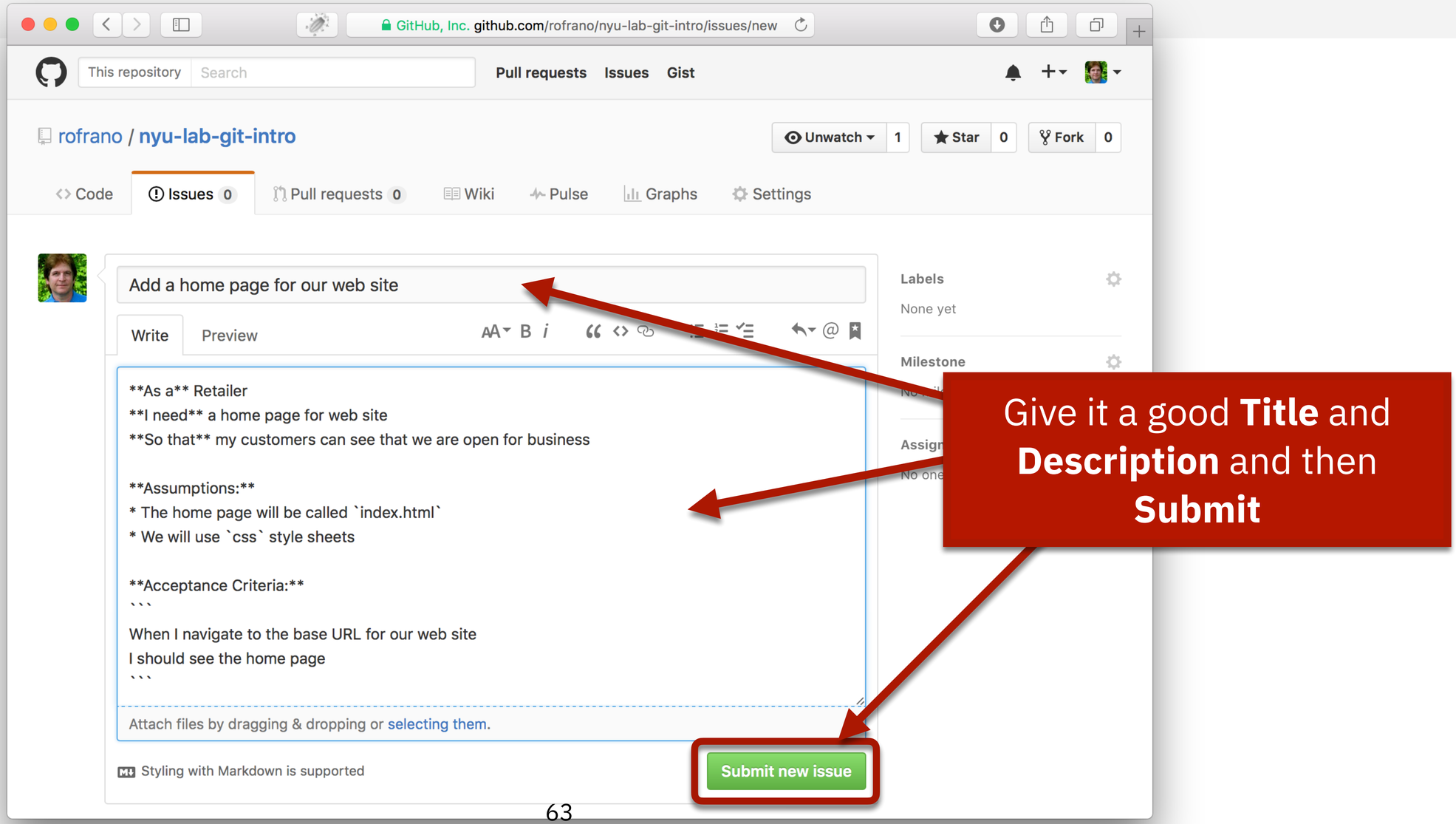


Markdown Reference

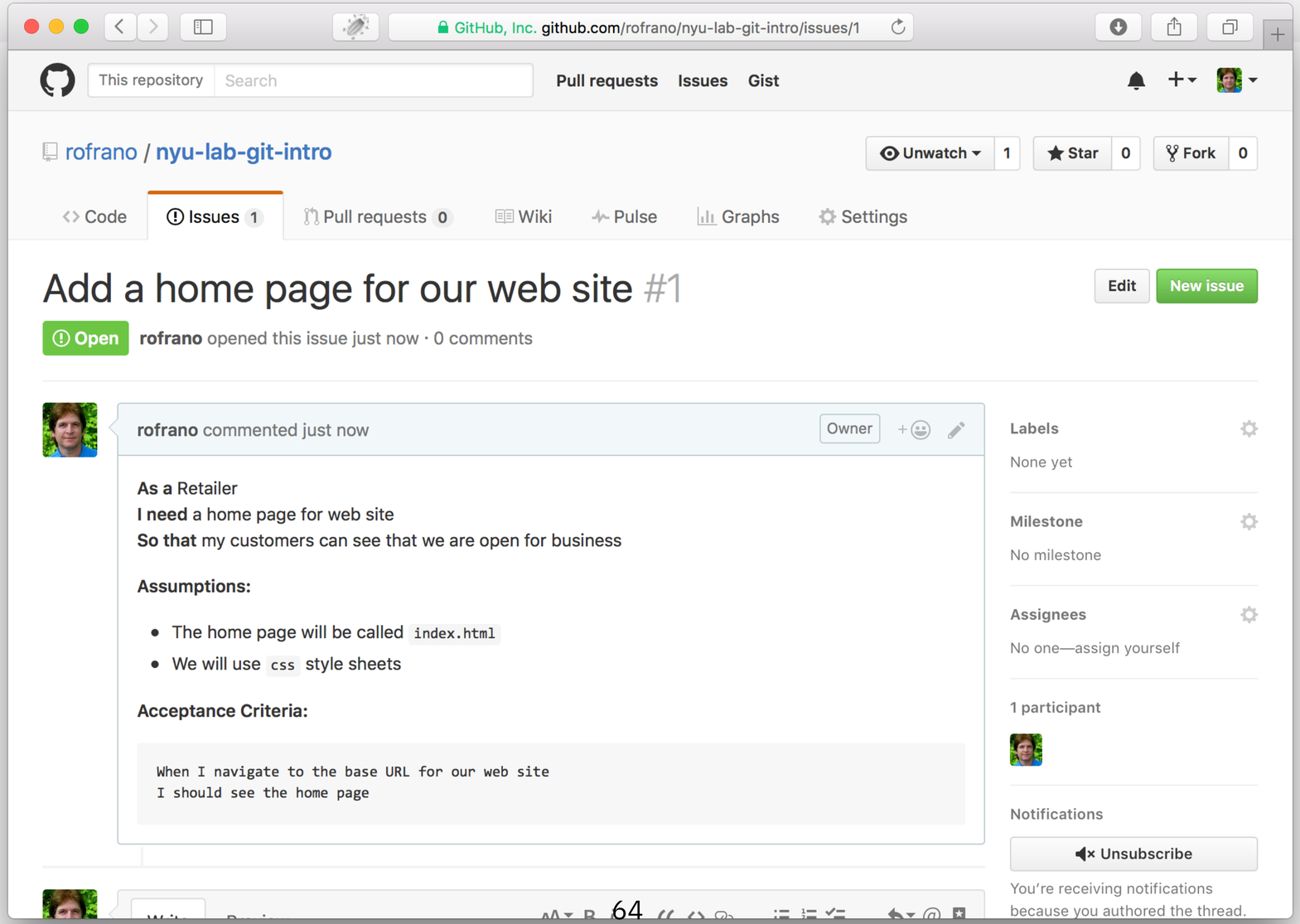
https://guides.github.com/features/mastering-markdown/

Type	Or	… to Get
Italic	_Italic_	*Italic*
Bold	__Bold__	**Bold**
# Heading 1	Heading 1 =========	**Heading 1**
## Heading 2	Heading 2 ---------	Heading 2
[Link](http://a.com)	[Link]	Link
![Image](http://url/a.png)	![Image]	
> Blockquote		I Blockquote
* List * List	- List - List	• List • List
1. One 2. Two	1) One 2) Two	1. One 2. Two

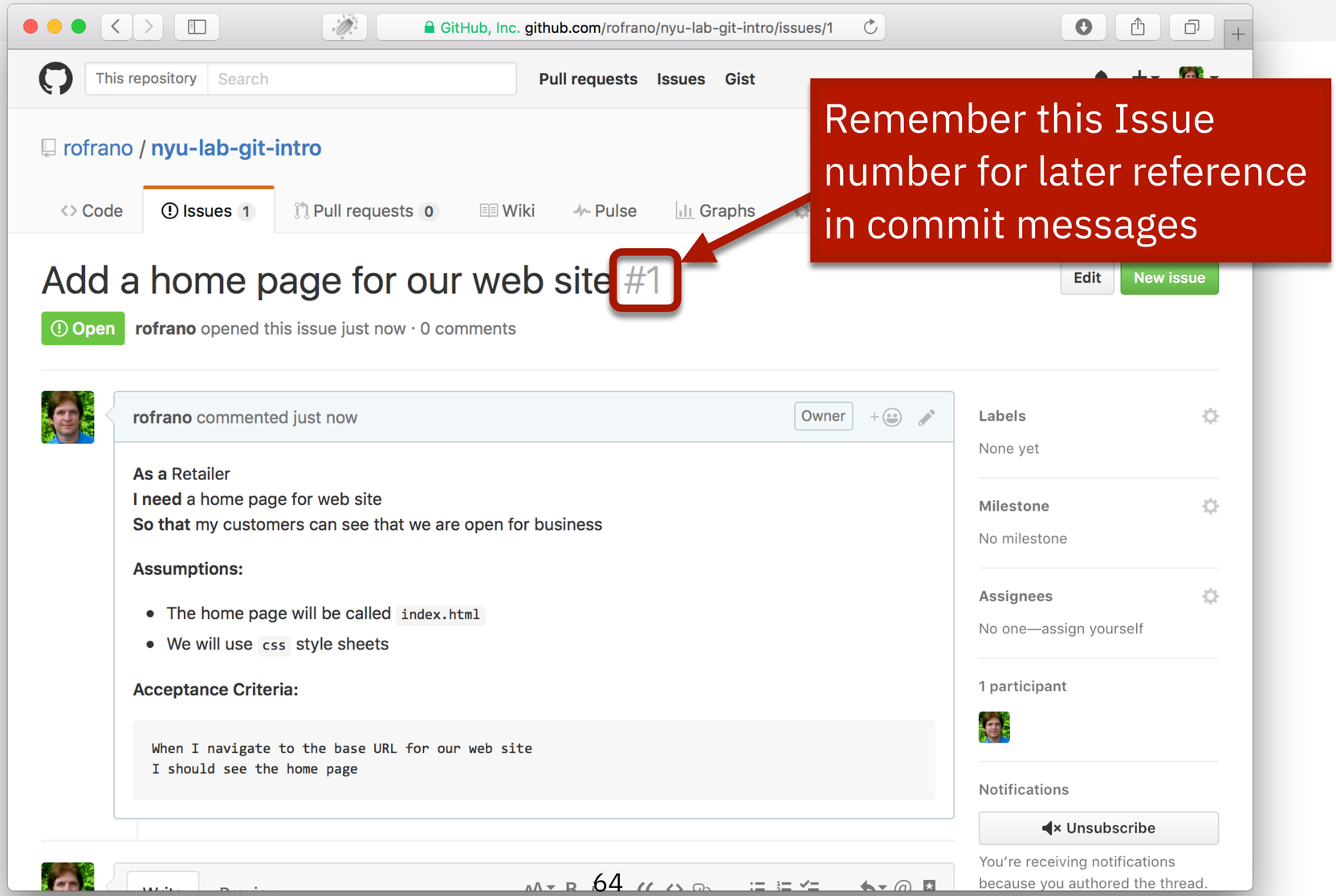
Fill Out The Issue



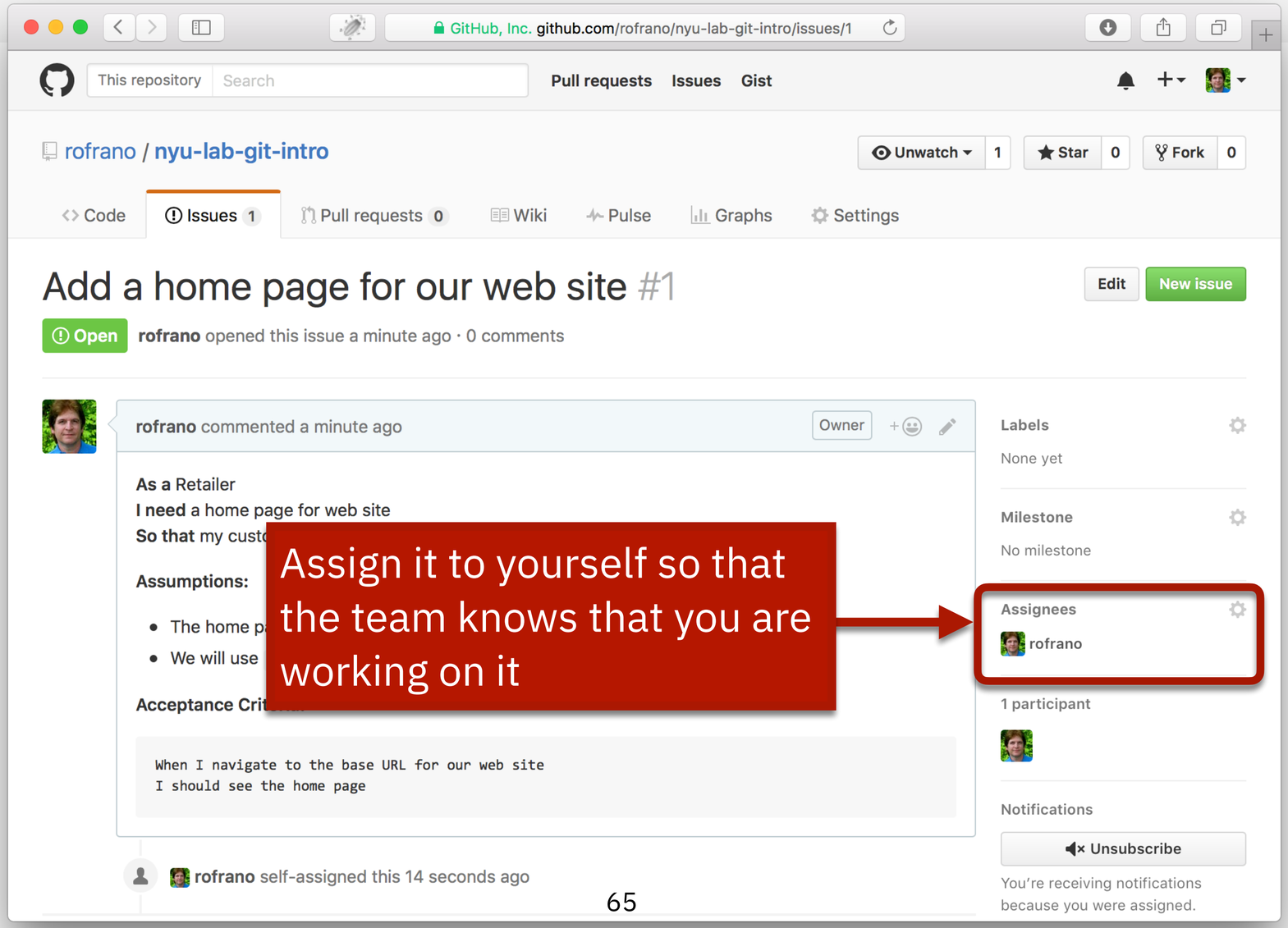
Issues are Assigned Numbers



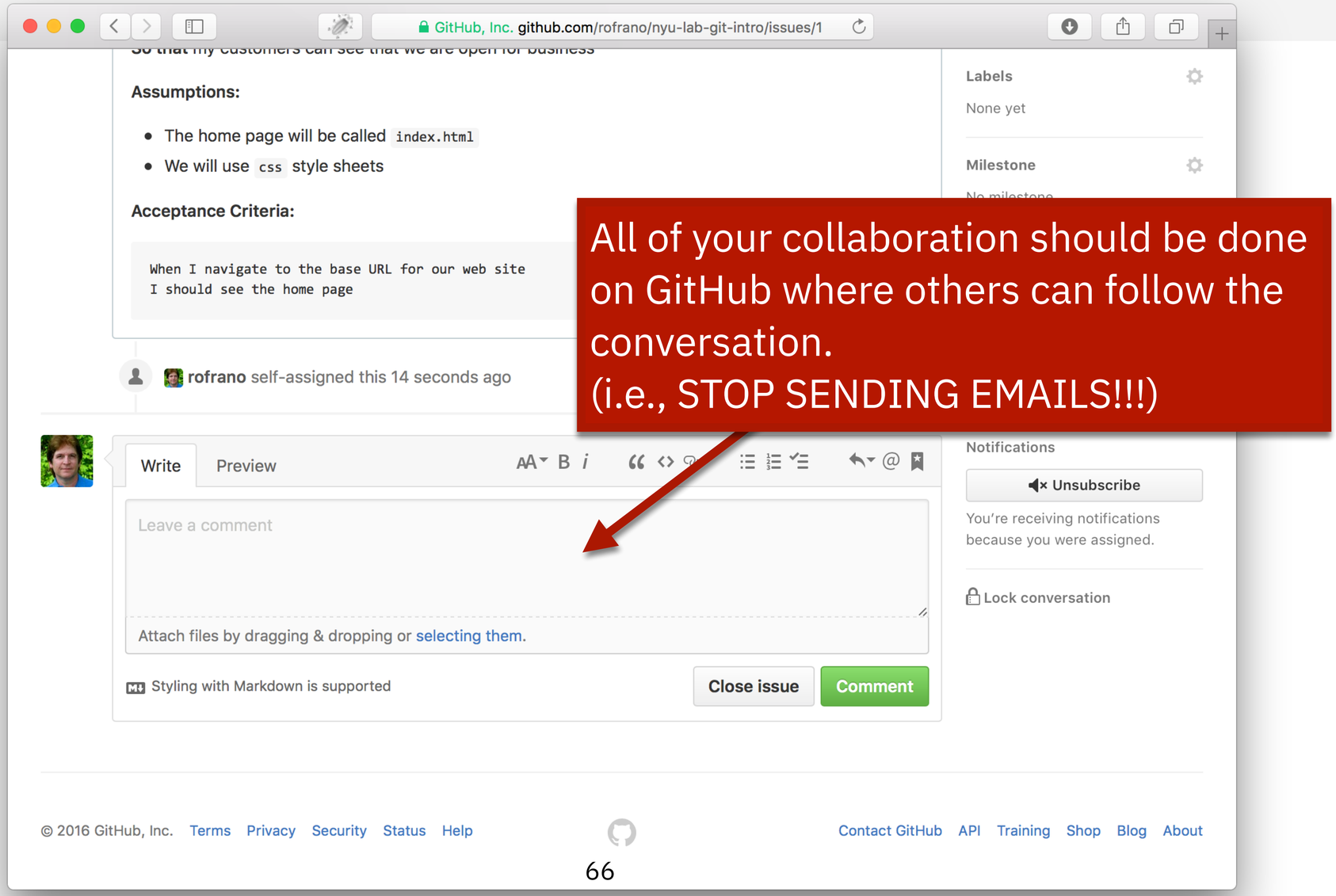
Issues are Assigned Numbers



Assign Issue to Yourself



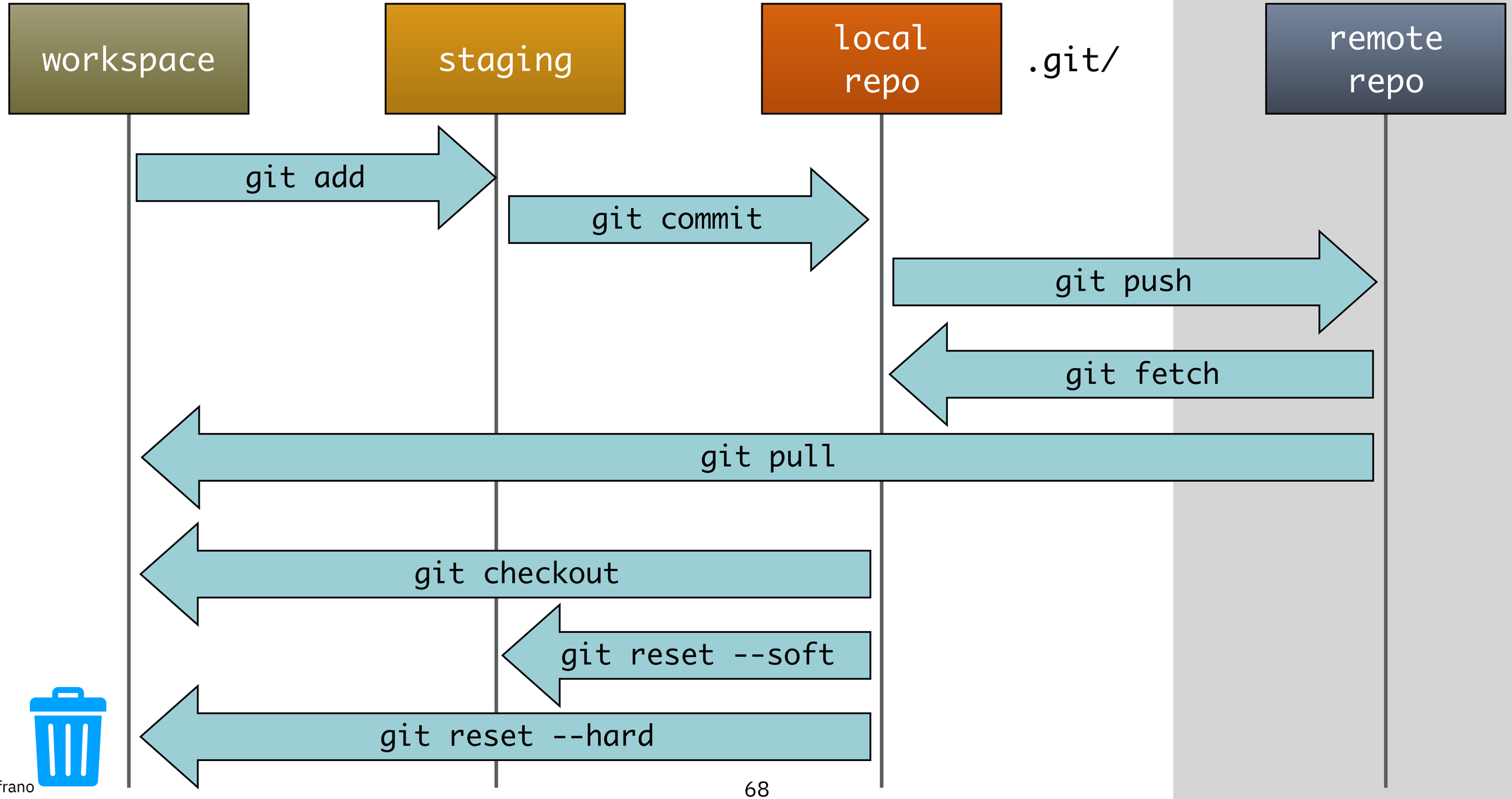
GitHub Collaboration



All of your collaboration should be done on GitHub where others can follow the conversation.
(i.e., STOP SENDING EMAILS!!!)

IT'S TIME TO PLAY

Git Command Workflow



Branches

- Branching is a core concept in Git
  - The entire GitHub Flow is based upon branches
- There's only one rule:
  - *Anything in the **master** branch must always be deployable.*
- It's extremely important that your new branch is created off of master when working on a feature or a fix

BRANCH

- All development is done in a BRANCH

```
git checkout -b add-home-page
```

- You can switch between BRANCHES

```
git checkout other-branch
```

- You can see what branches exist (-a returns all both local and remote)

```
git branch -a
```

- You should delete the BRANCH after it has been MERGED

```
git branch -d add-home-page
```

Create A Branch

- Create a BRANCH to add the home page

```
$ git checkout -b add-home-page
Switched to a new branch 'add-home-page'
$
```

Add Some Files

- Add your working files and edit them as needed

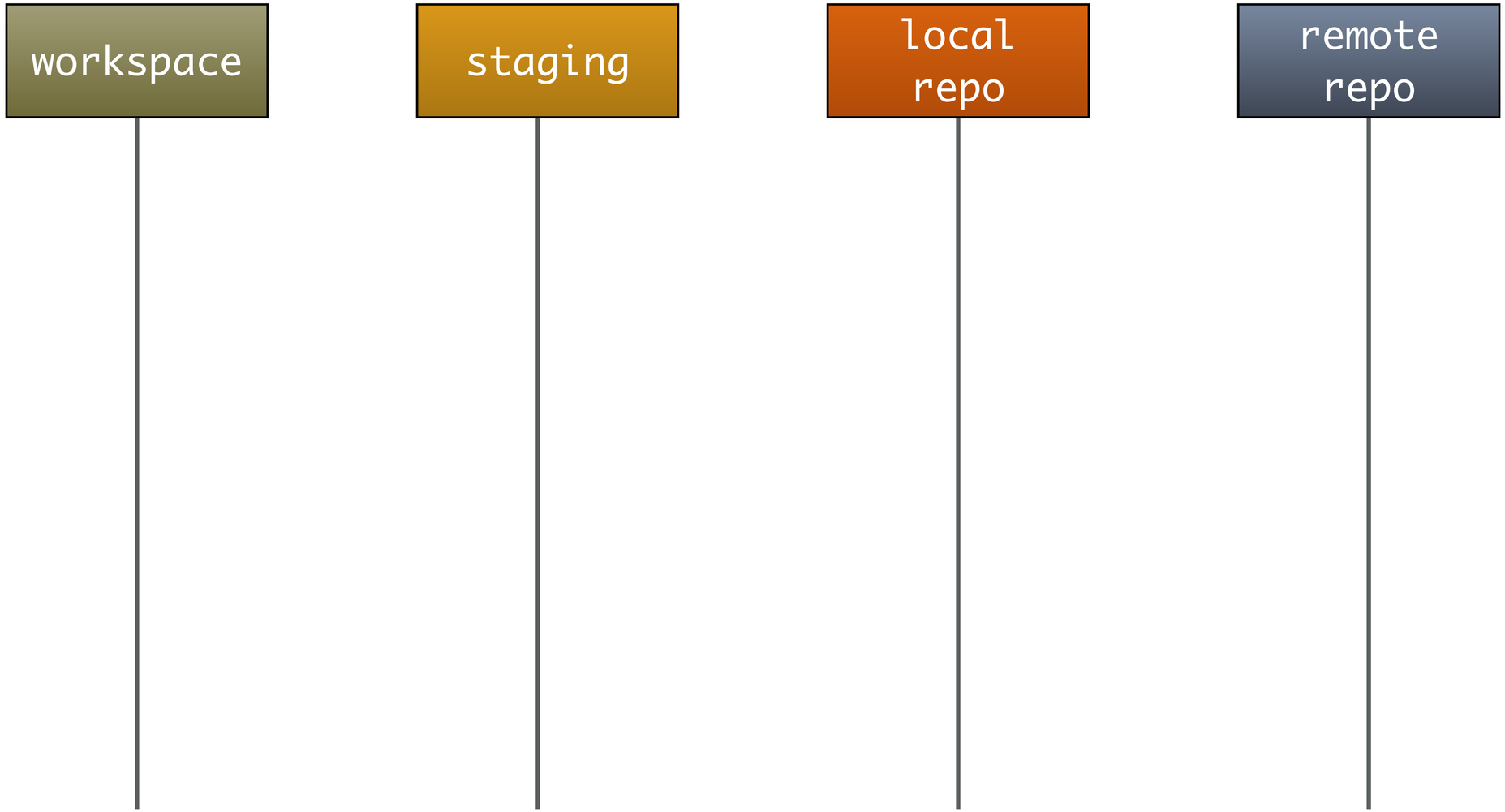
```
$ git checkout -b add-home-page
Switched to a new branch 'add-home-page'
$ touch index.html
$ touch index.css
$
```

Check Your Status

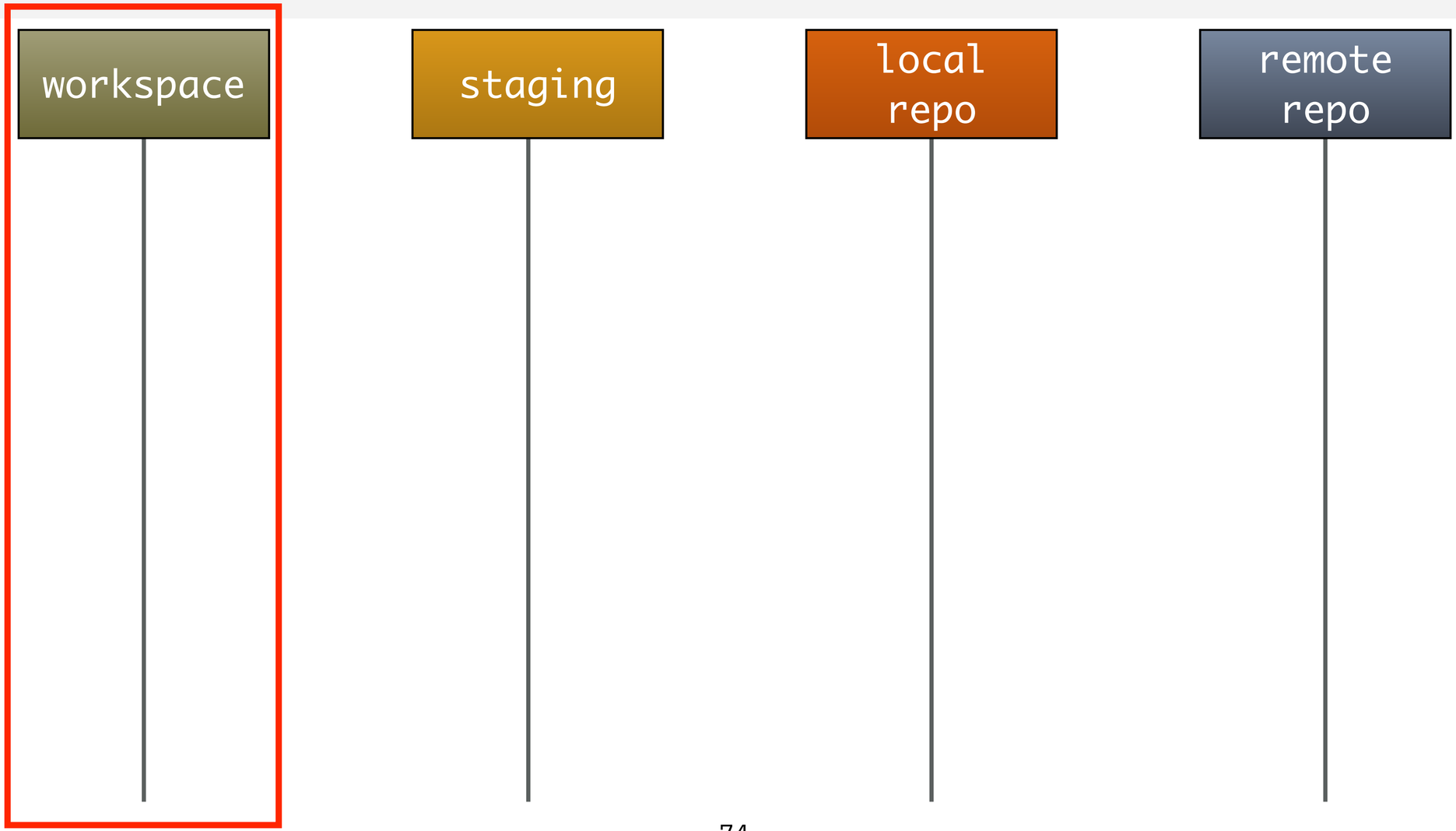
- Git Status shows that the files are not yet Tracked

```
$ git checkout -b add-home-page
Switched to a new branch 'add-home-page'
$ touch index.html
$ touch index.css
$ git status
On branch add-home-page
Untracked files:
  (use "git add <file>..." to include in what will be committed)

  index.css
  index.html

nothing added to commit but untracked files present (use "git add" to track)
$
```

Where Are Your Changes Now?



Where Are Your Changes Now?



ADD

- Use ADD to stage your files for a COMMIT
  - You can use individual filenames or wild cards
  - `git add .` <— will add everything! (use with extreme caution)

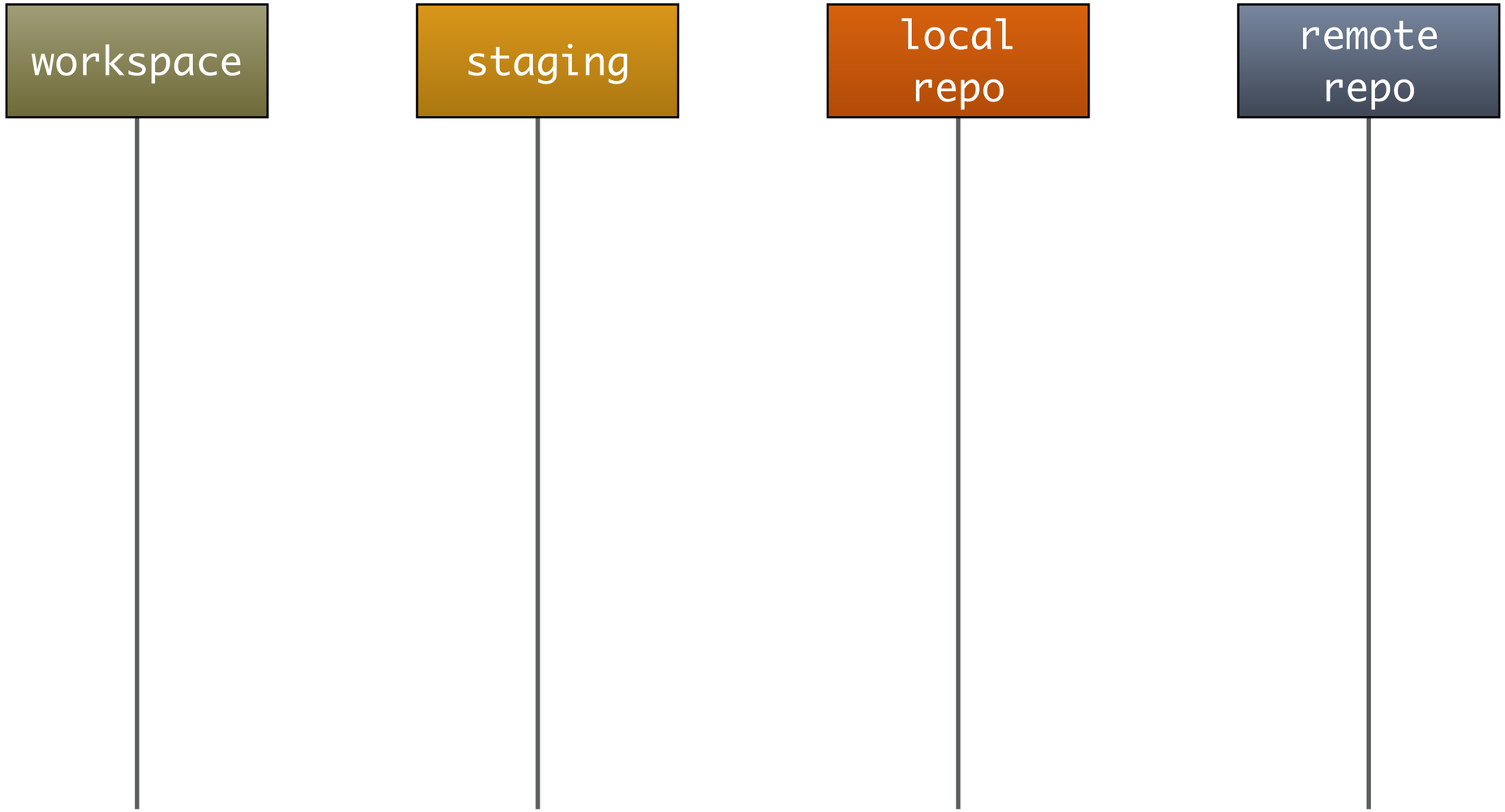
```
$ git add index.*
$
```

Check Your Status

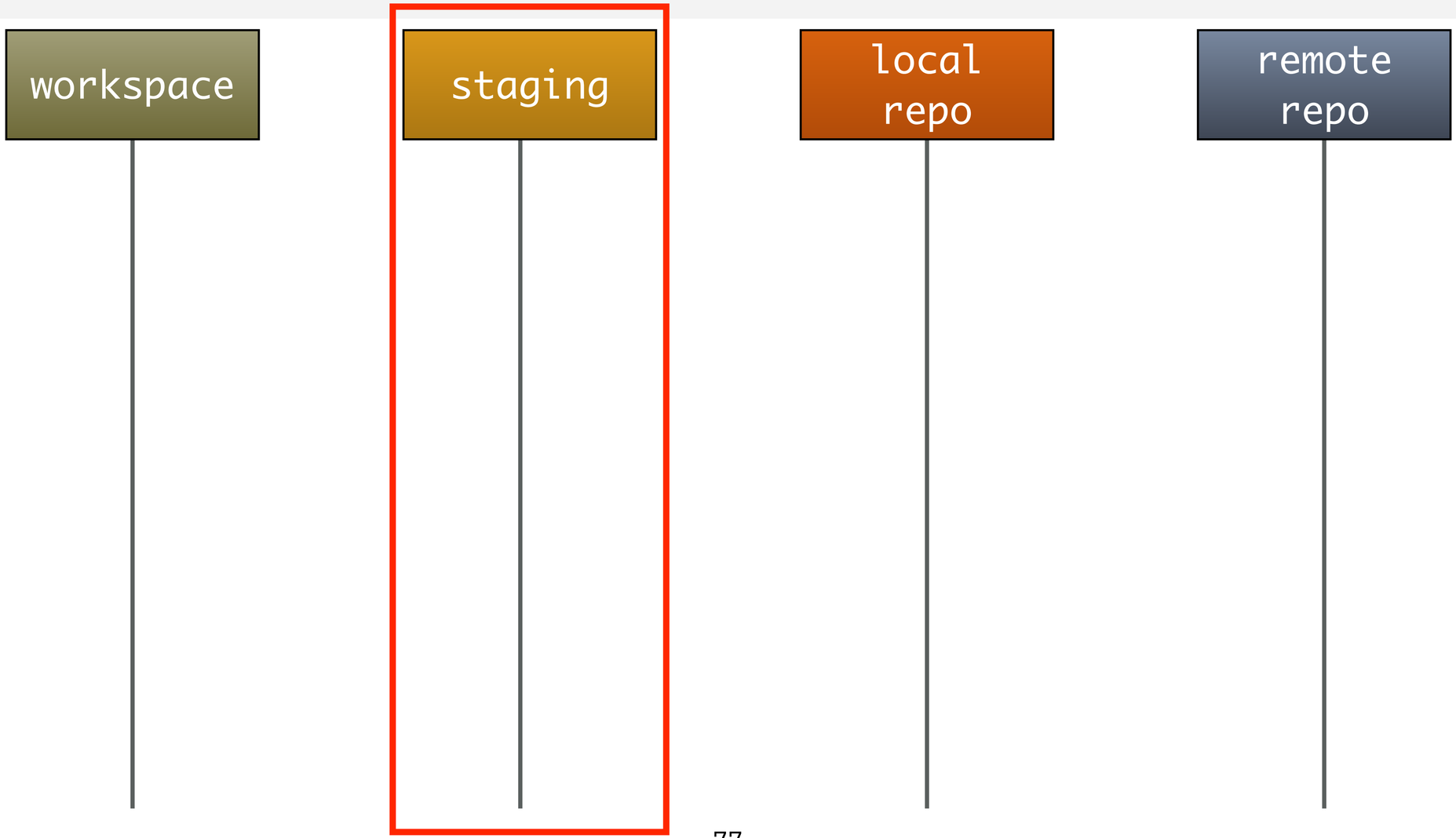
- Check your STATUS
  - You now have two new files staged for the next commit

```
$ git add index.*
$ git status
On branch add-home-page
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

  new file:   index.css
  new file:   index.html

$
```

Where Are Your Changes Now?



Where Are Your Changes Now?



Commit

- COMMIT your changes adding a meaningful message

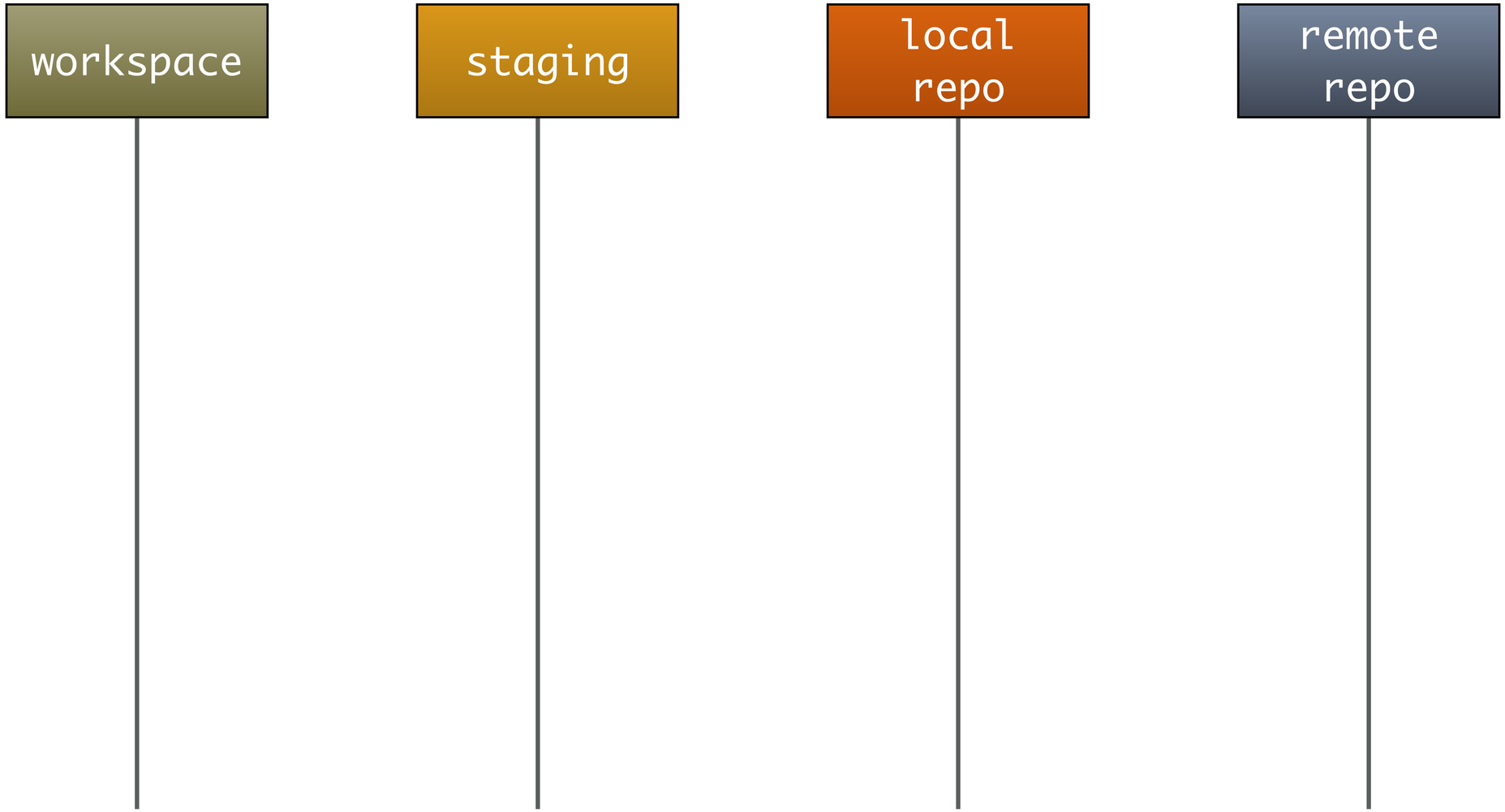
```
$ git commit -m 'added home page and style sheet'
[add-home-page ab9d8be] added home page and style sheet
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 index.css
 create mode 100644 index.html
$
```

Commit

- COMMIT your changes adding a meaningful message
- Check your STATUS and note your clean directory

```
$ git commit -m 'added home page and style sheet'
[add-home-page ab9d8be] added home page and style sheet
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 index.css
 create mode 100644 index.html
$ git status
On branch add-home-page
nothing to commit, working directory clean
$
```

Where Are Your Changes Now?



Where Are Your Changes Now?

CHECKOUT

- Now that you have committed your changes you have a safe place to return to
- If you do more with on `index.html` and you want to revert back to the last commit, just use:

```
git checkout -- index.html
```

LOG

- We can use `GIT LOG` to see where we are
- But the defaults are pretty useless :(

```
$ git log
commit ab9d8be47269d76c319246bebb17f73d7dffebda
Author: John Rofrano <johnnyroy@johnrofrano.com>
Date:   Mon Sep 12 22:26:03 2016 -0400

    added home page and style sheet

commit 907cda61614c761945124582247fc59a8f696d5b5
Author: John Rofrano <johnnyroy@johnrofrano.com>
Date:   Mon Sep 12 15:35:05 2016 -0400

    Initial commit
```

Much Improved LOG

- **--oneline** makes the log compact
- **--decorate** adds branch information
- **--all** shows all branches
- **--graph** creates a nice graph of the branches and merges

```
$ git log --oneline
ab9d8be added home page and style sheet
907cda6 Initial commit

$ git log --oneline --decorate --all --graph
* ab9d8be (HEAD -> add-home-page) added home page and style sheet
* 907cda6 (origin/master, origin/HEAD, master) Initial commit

$
```

You Can Make Aliases

- You can make an alias for any git command to make typing easier
- The syntax is: `git config --global alias.<name> "<command>"`
- To make an alias called '`lg`' for this log command use:

```
$ git config --global alias.lg "log --oneline --decorate --all --graph"
```

You Can Make Aliases

- You can make an alias for any git command to make typing easier
- The syntax is: `git config --global alias.<name> "<command>"`
- To make an alias called '`lg`' for this log command use:

```
$ git config --global alias.lg "log --oneline --decorate --all --graph"
```

This is the name of the alias

You Can Make Aliases

- You can make an alias for any git command to make typing easier
- The syntax is: `git config --global alias.<name> "<command>"`
- To make an alias called '`lg`' for this log command use:

DIFF

- We can use `GIT DIFF` to see the differences with another branch
- e.g. `git diff master` will show what is changed in our branch from the master branch

```
$ git diff master
diff --git a/index.css b/index.css
new file mode 100644
index 0000000..e69de29
diff --git a/index.html b/index.html
new file mode 100644
index 0000000..e69de29
```

More Git Diff

`git diff`	Shows what changes you've made from the index. This shows what changes are NOT committed if you
`git diff master`	Shows diff from workspace to master commit
`git diff --staged` `git diff --cached`	Shows what would be git commit'ed
`git diff --name-only <branch-name>`	Show only the names of the files that have changed

PUSH

- To push your COMMITs to your remote BRANCH

```
git push -u origin add-home-page
```

- This saves you if your hard drive crashes!!!
- Do this at the end of every day, or after important changes
  - Note: After initial `-u` you only need to use `git push`

Push to Remote Branch

- PUSH your changes to a new remote branch

```
$ git push -u origin add-home-page
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 368 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To git@github.com:rofrano/nyu-lab-git-intro.git
 * [new branch]      add-home-page -> add-home-page
Branch add-home-page set up to track remote branch add-home-page from origin.
$
```

Push to Remote Branch

- PUSH your changes to a new remote branch
- Check your STATUS and note you are up-to-date

```
$ git push -u origin add-home-page
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 368 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To git@github.com:rofrano/nyu-lab-git-intro.git
 * [new branch]      add-home-page -> add-home-page
Branch add-home-page set up to track remote branch add-home-page from origin.
$ git status
On branch add-home-page
Your branch is up-to-date with 'origin/add-home-page'.
nothing to commit, working directory clean
$
```

How Has Status Changed?

- Status now has information about the remote branch

Before

```
$ git status
On branch add-home-page
nothing to commit, working directory clean
$
```

After

```
$ git status
On branch add-home-page
Your branch is up-to-date with 'origin/add-home-page'.
nothing to commit, working directory clean
$
```

How Has Status Changed?

- Status now has information about the remote branch

Before

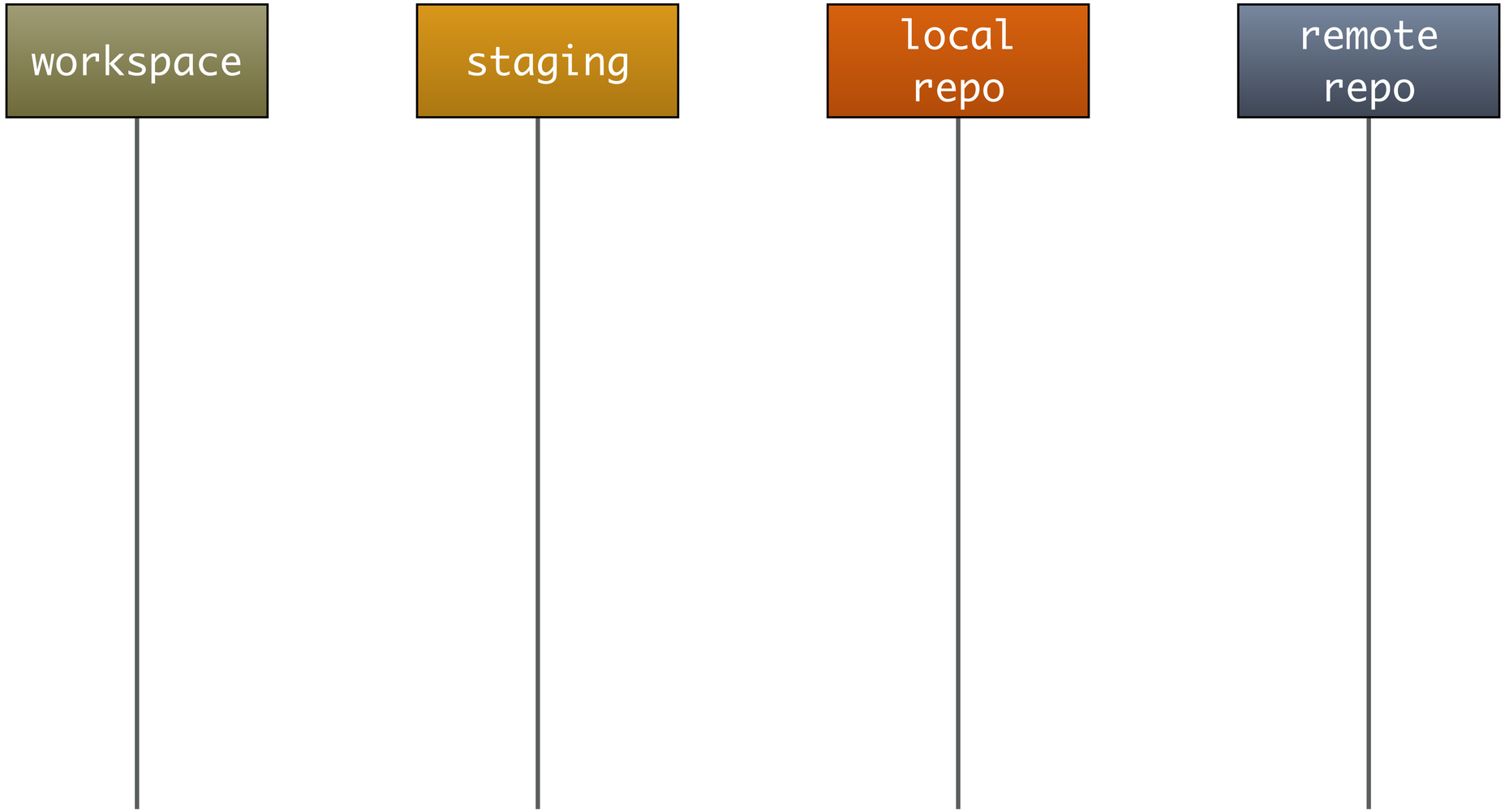
```
$ git status
On branch add-home-page
nothing to commit, working directory clean
$
```

After

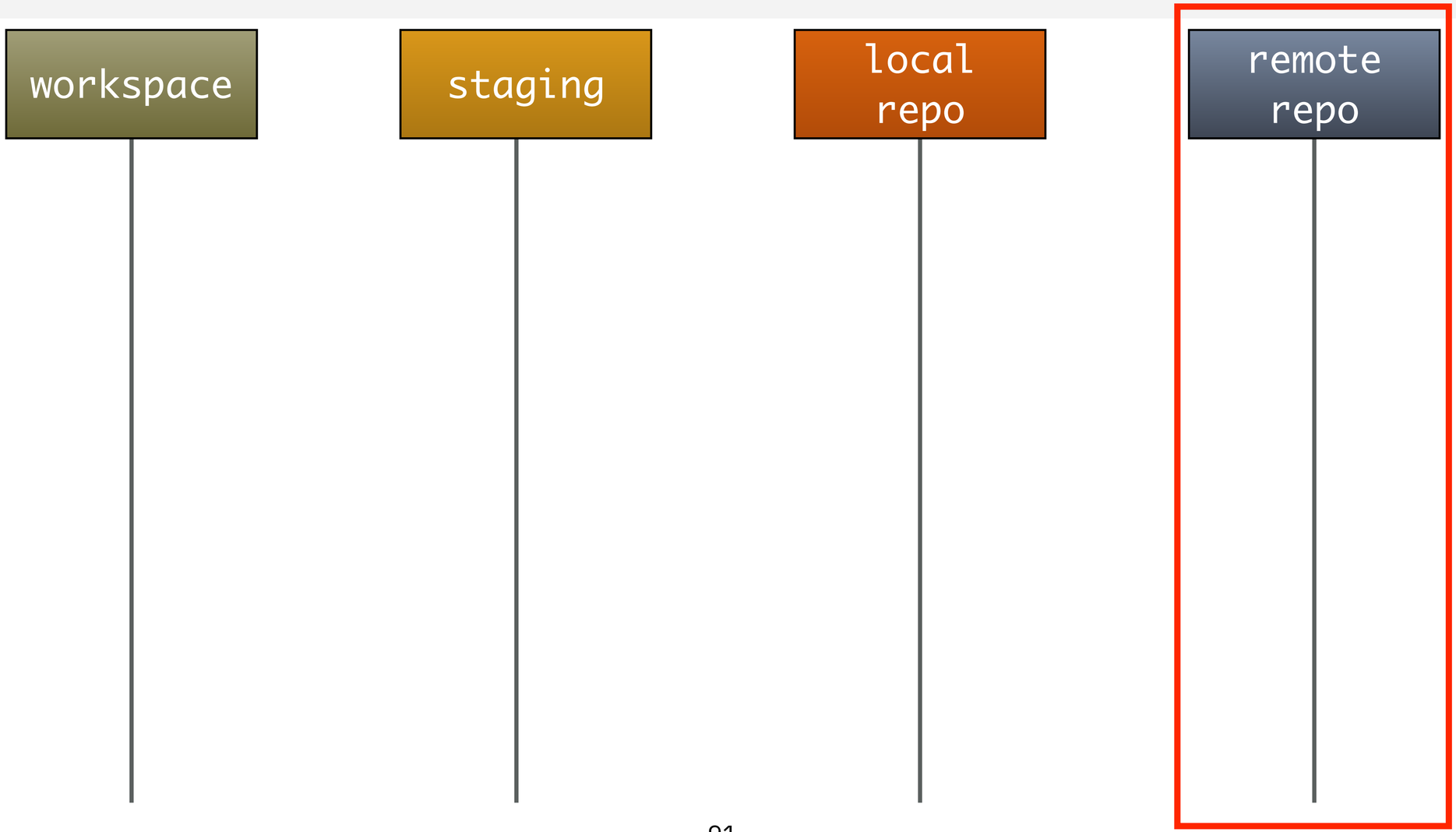
```
$ git status
On branch add-home-page
Your branch is up-to-date with 'origin/add-home-page'.
nothing to commit, working directory clean
$
```

Information about the remote branch

Where Are Your Changes Now?



Where Are Your Changes Now?



Check the LOG

- We can view the logs to see where we are at now as well
- HEAD points to both the remote and local branch add-home-page

```
$ git log --oneline --decorate --all --graph
* ab9d8be (HEAD -> add-home-page, origin/add-home-page) added home page and style sheet
* 907cda6 (origin/master, origin/HEAD, master) Initial commit
$
```

Check the LOG

- We can view the logs to see where we are at now as well
- HEAD points to both the remote and local branch add-home-page

```
$ git log --oneline --decorate --all --graph
* ab9d8be (HEAD -> add-home-page, origin/add-home-page) added home page and style sheet
* 907cda6 (origin/master, origin/HEAD, master) Initial commit
$
```

HEAD is the most recent commit in the history

Pull request workflow

- The pull request workflow helps us to share information. It gives the following benefits:
- **A second set of eyes:** Sometimes you may have solved a problem, but when someone reviews it they pointed out a subtle flaw, or perhaps a way you could solve it more elegantly. It also helps identify things like security exposures. This results in a much cleaner codebase.
- **Knowledge sharing:** It is dangerous to only have one developer that understands the code. Having another developer review the code allows for shared understanding and protects against one developer leaving the project.
- **Allows automation of unit testing:** You setup your github project to run unit tests on pull requests so that we can verify that everything works before merging.

Prepare for Pull Request

- You want to make sure that you are using the latest code before you make a pull request
- The remote master branch has probably moved on since you created your branch
- Therefore it is necessary "rebase to" or "merge with" the current master

Fetch and Pull

- Fetch will bring down all of the changes from the remote repository
- Pull will merge the changes from the current branch with your local workspace
  - Note: Pull does a fetch so if you just want to pull there is no need to fetch
- We do this on the master branch

Merge

- Before you issue a **Pull Request**, it’s a good idea to merge from master
- Merge joins two branches together, usually your branch and master but it can be any two branches only changing the checked out branch
- If there are no conflicts and the current HEAD is an ancestor of the merge branch, git will fast-forward the merge by moving the pointer forward
- Merge retains all the history as it happened chronologically

Refreshing Master Before PR

- Switch to the master branch
- Pull the latest changes
- In this case we were already up to date

```
$ git checkout master
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.
$ git pull
Already up-to-date.
$
```

Merge from Current Master

- Change to your branch to prepare to merge

```
$ git checkout add-home-page
Switched to branch 'add-home-page'
Your branch is up-to-date with 'origin/add-home-page'.
$
```

Merge from Master

- Issue the `merge` command to merge any the new code

```
$ git checkout add-home-page
Switched to branch 'add-home-page'
Your branch is up-to-date with 'origin/add-home-page'.
$ git merge master
Current branch add-home-page is up to date.
$
```

Push to Remote Branch

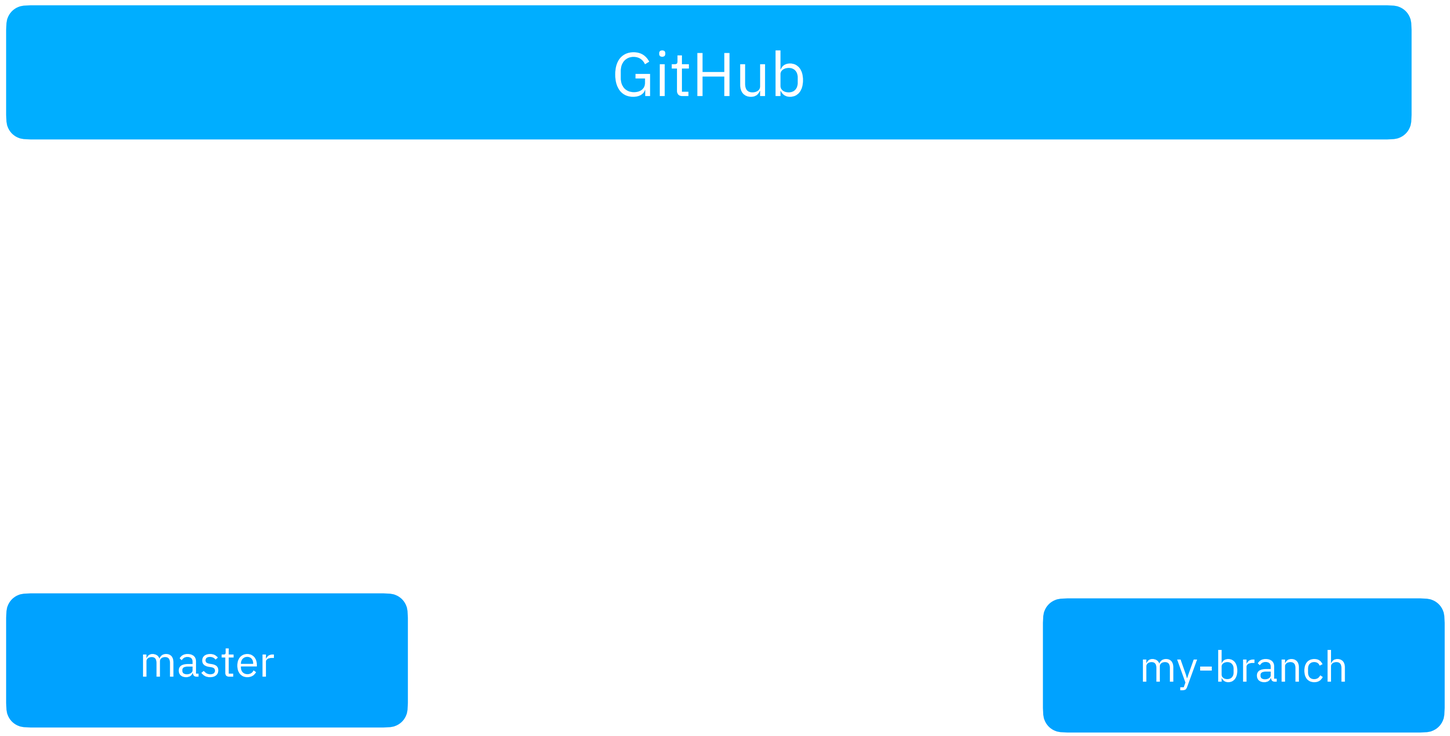
- One the merge is complete, push the results to your remote branch

```
$ git checkout add-home-page
Switched to branch 'add-home-page'
Your branch is up-to-date with 'origin/add-home-page'.
$ git merge master
Current branch add-home-page is up to date.
$ git push
Everything up-to-date
$
```

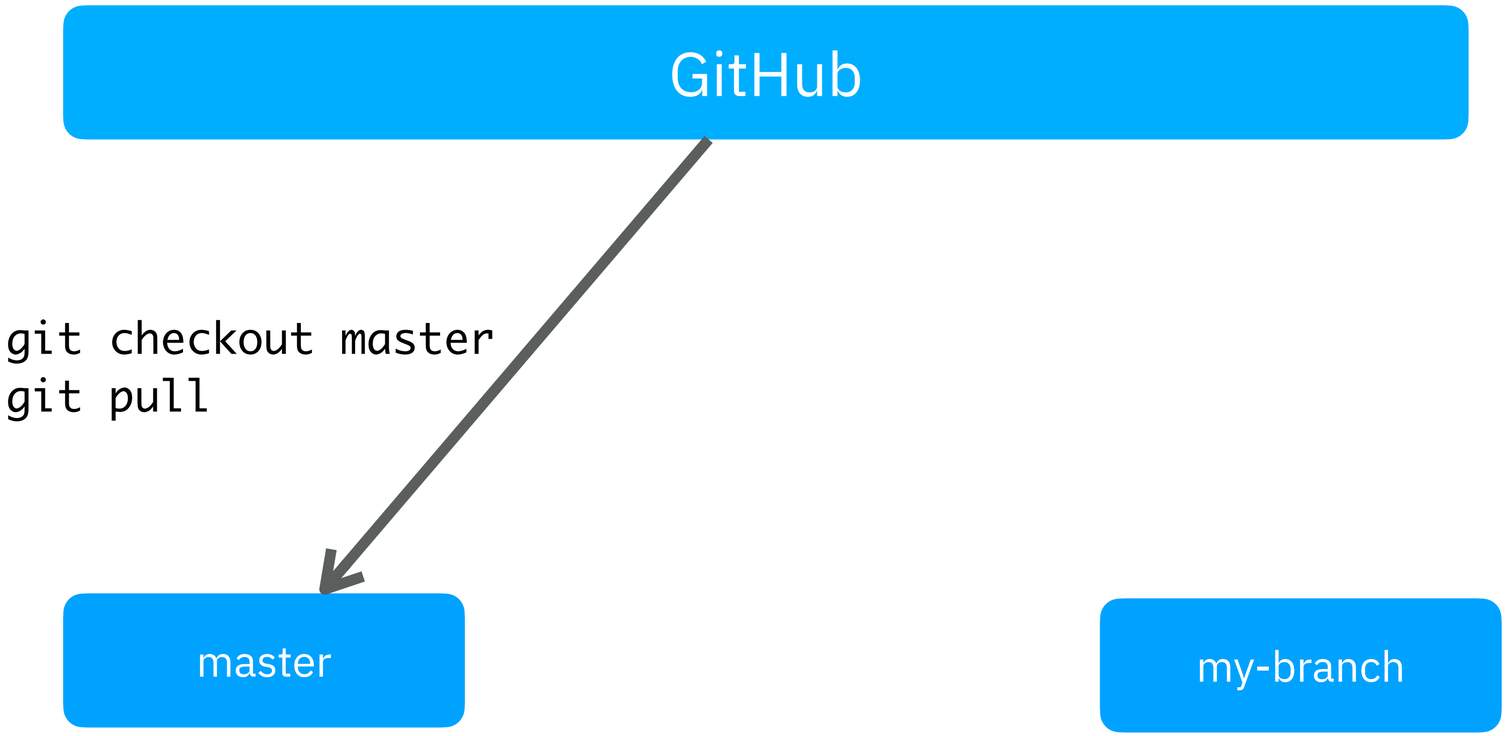
REBASE

- Rebase will replay your changes on top of the current version of master
- It will look as if your branch was created from the current state of master
- You can optionally do this before pushing/creating to remote branch otherwise use merge
- Never rebase once you have pushed to a remote. It will mess everyone else up because it changes history in a way that is incompatible

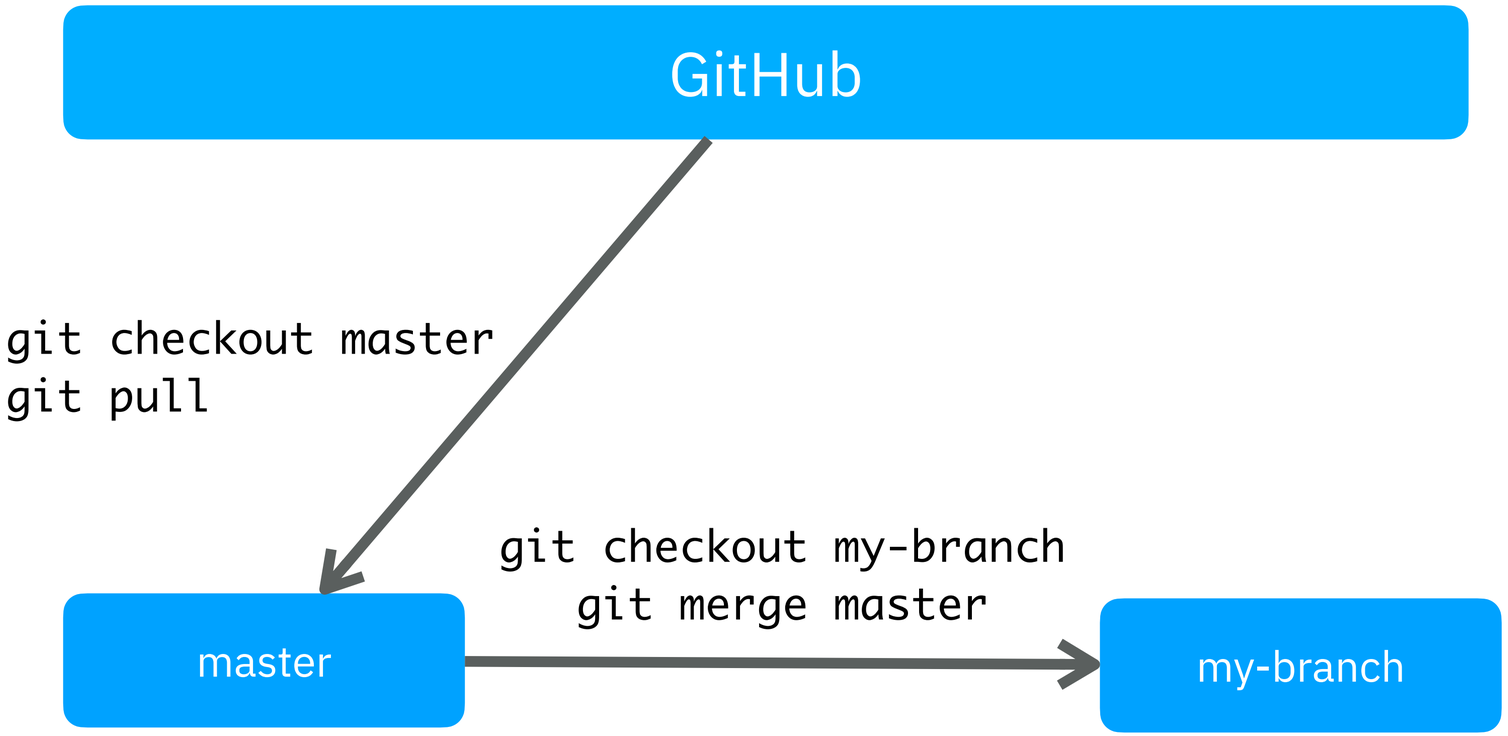
Pull Request Prep Summary



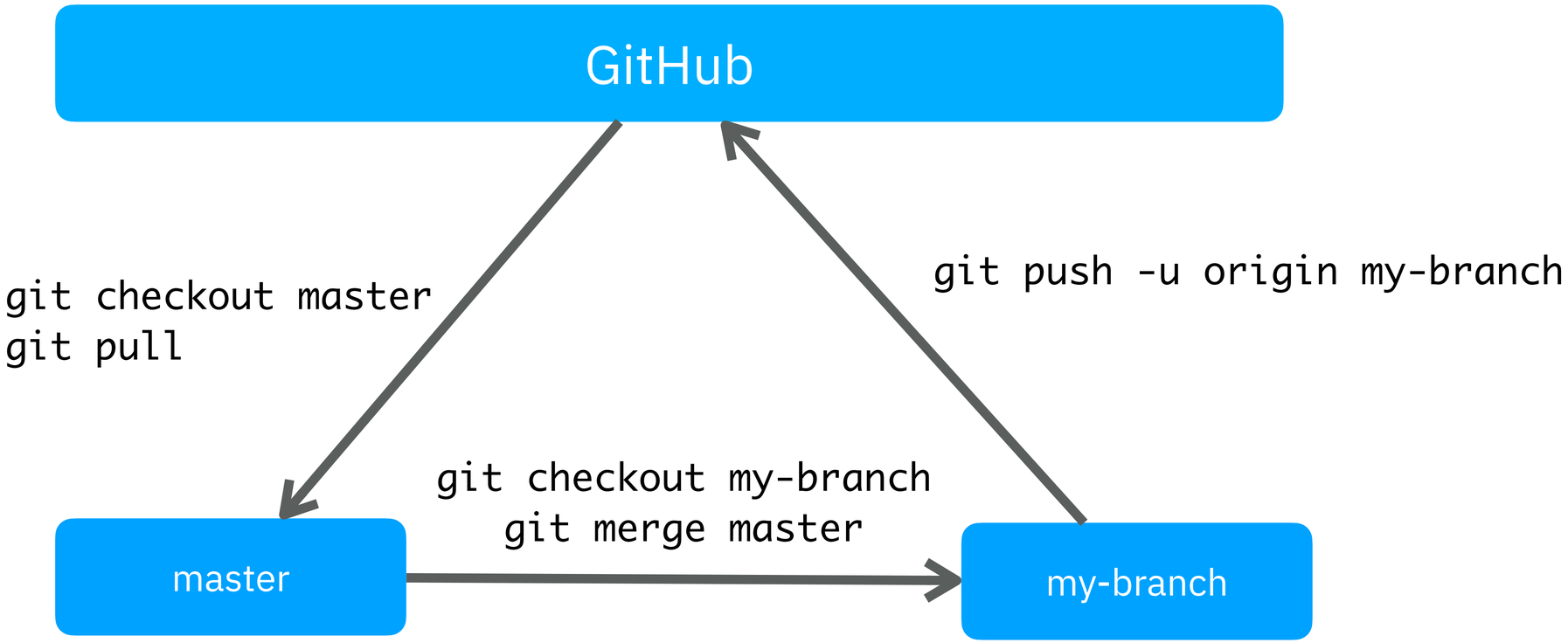
Pull Request Prep Summary



Pull Request Prep Summary

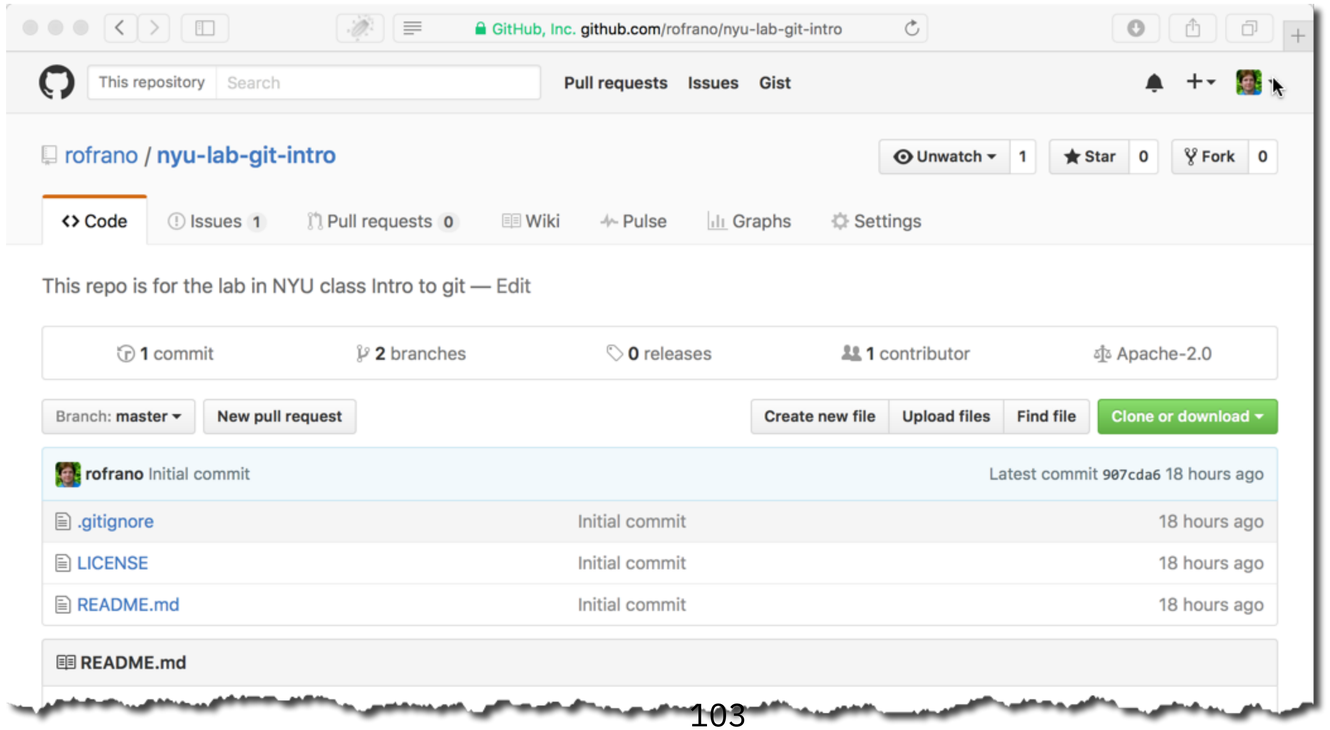


Pull Request Prep Summary



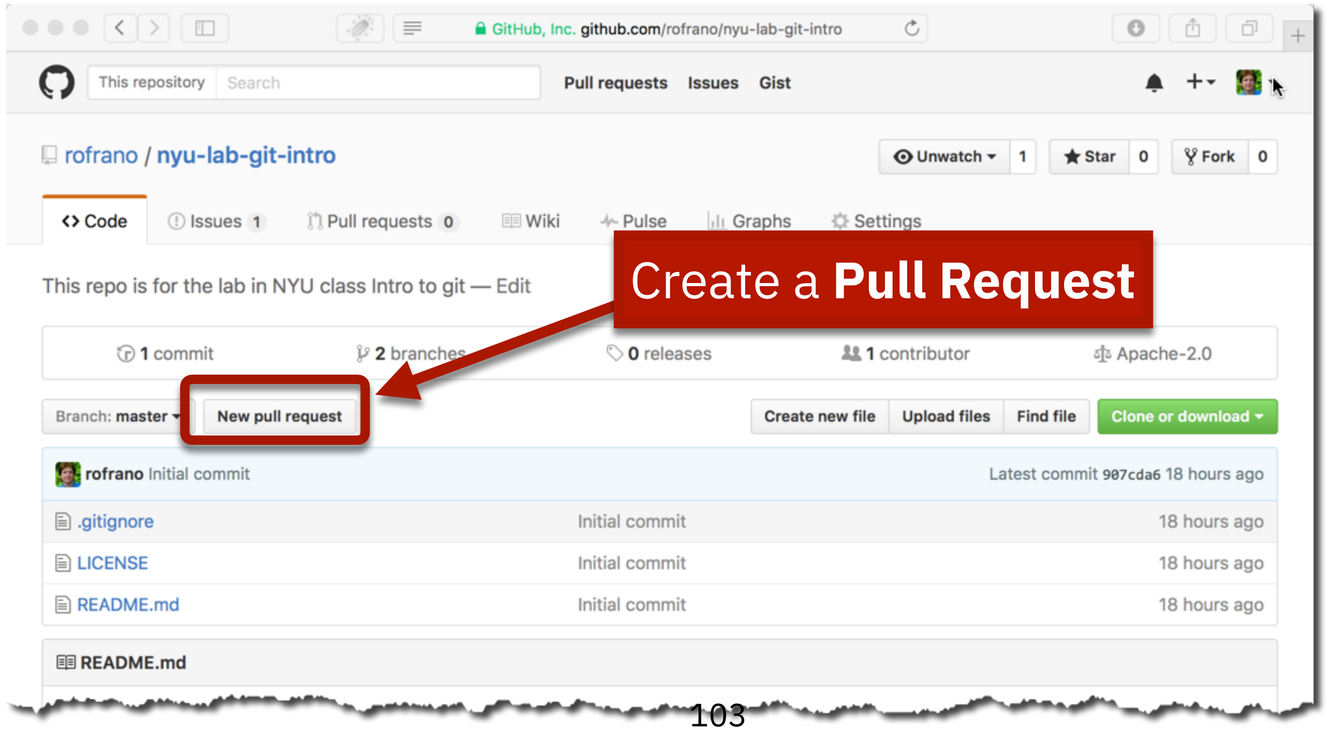
Pull Request

- Use a PULL REQUEST to notify the team that you changes are ready to be reviewed and merged into the MASTER branch
- If you are working on a Fork, initiate the Pull Request from the Forked repository

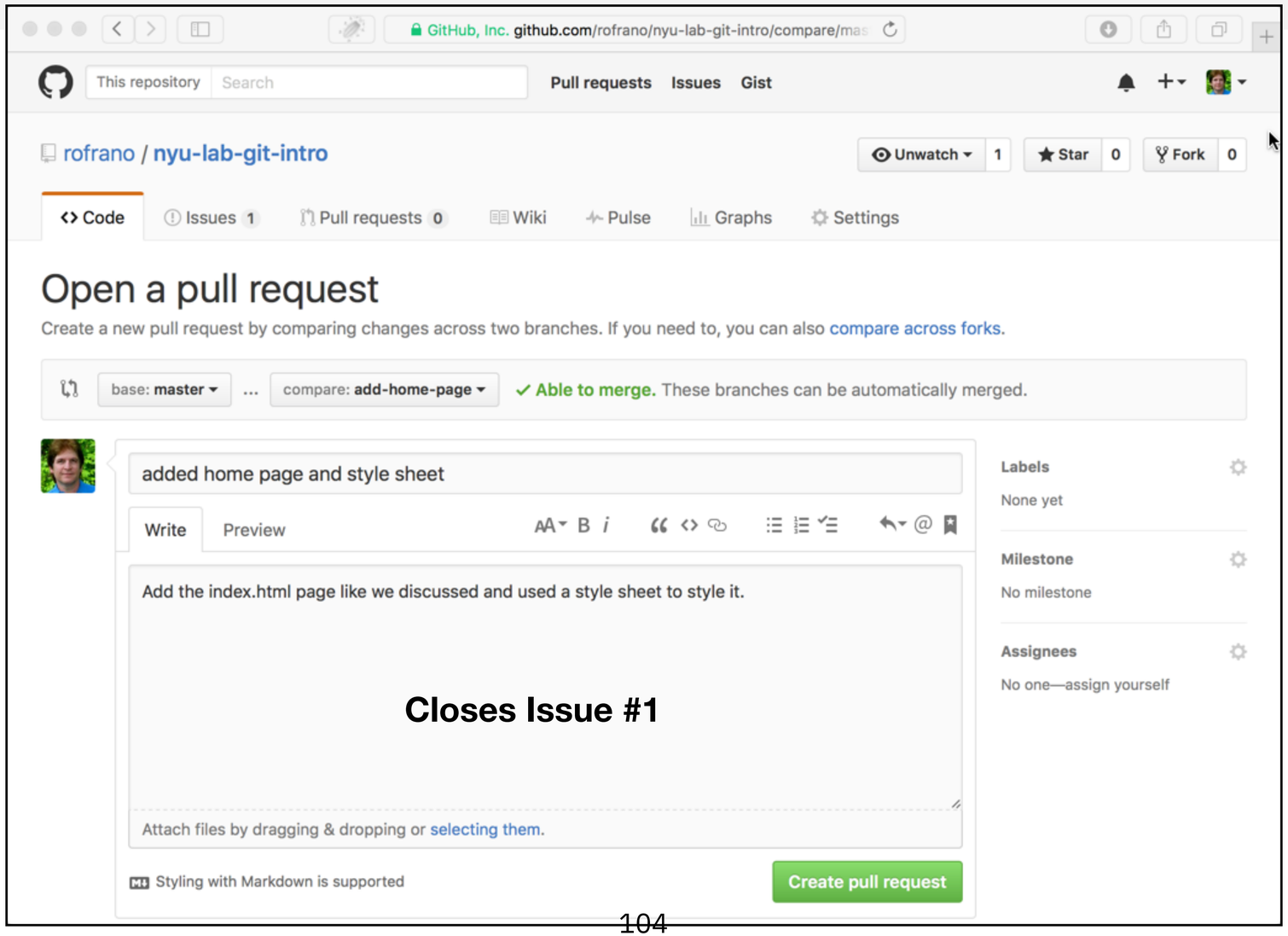


Pull Request

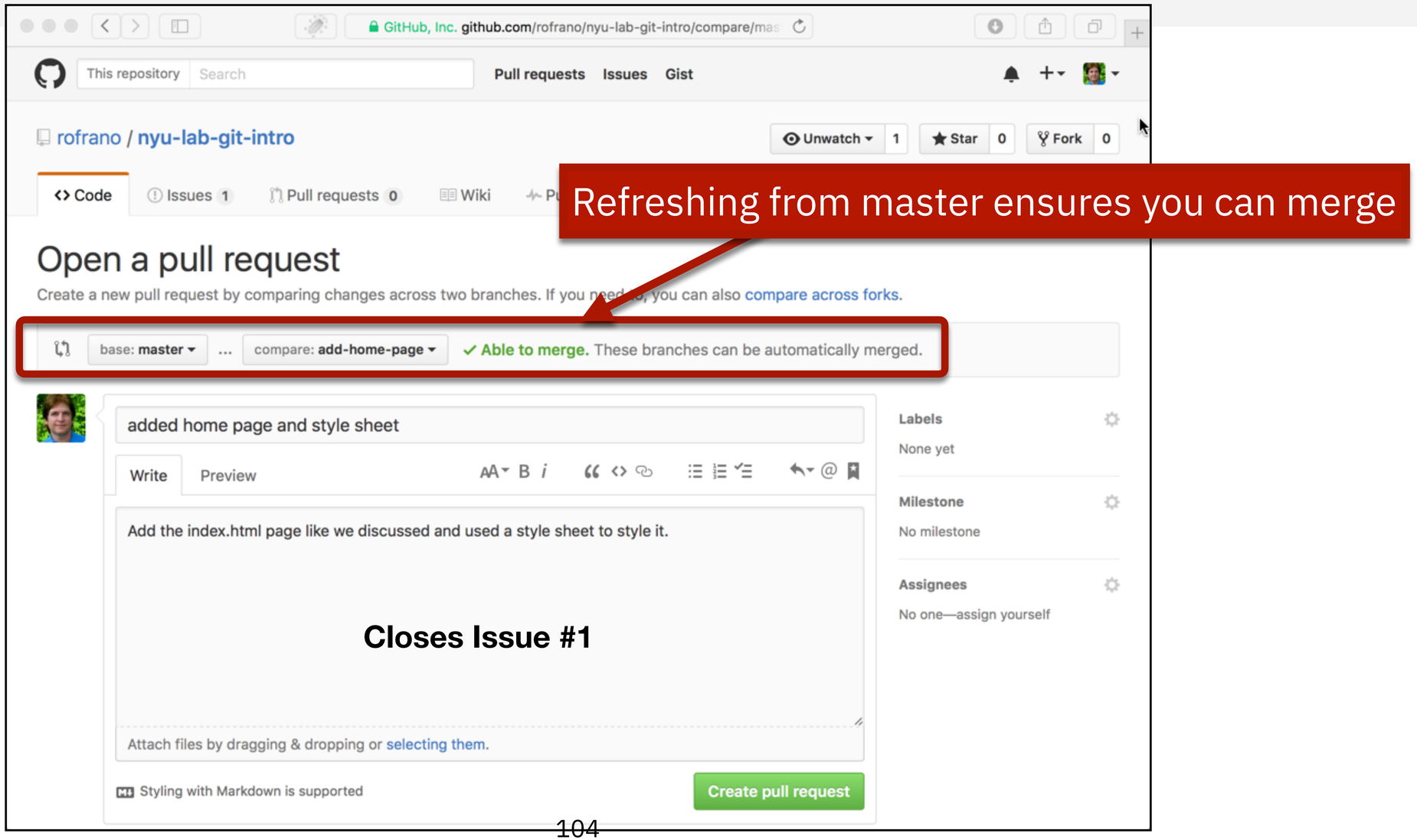
- Use a PULL REQUEST to notify the team that you changes are ready to be reviewed and merged into the MASTER branch
- If you are working on a Fork, initiate the Pull Request from the Forked repository



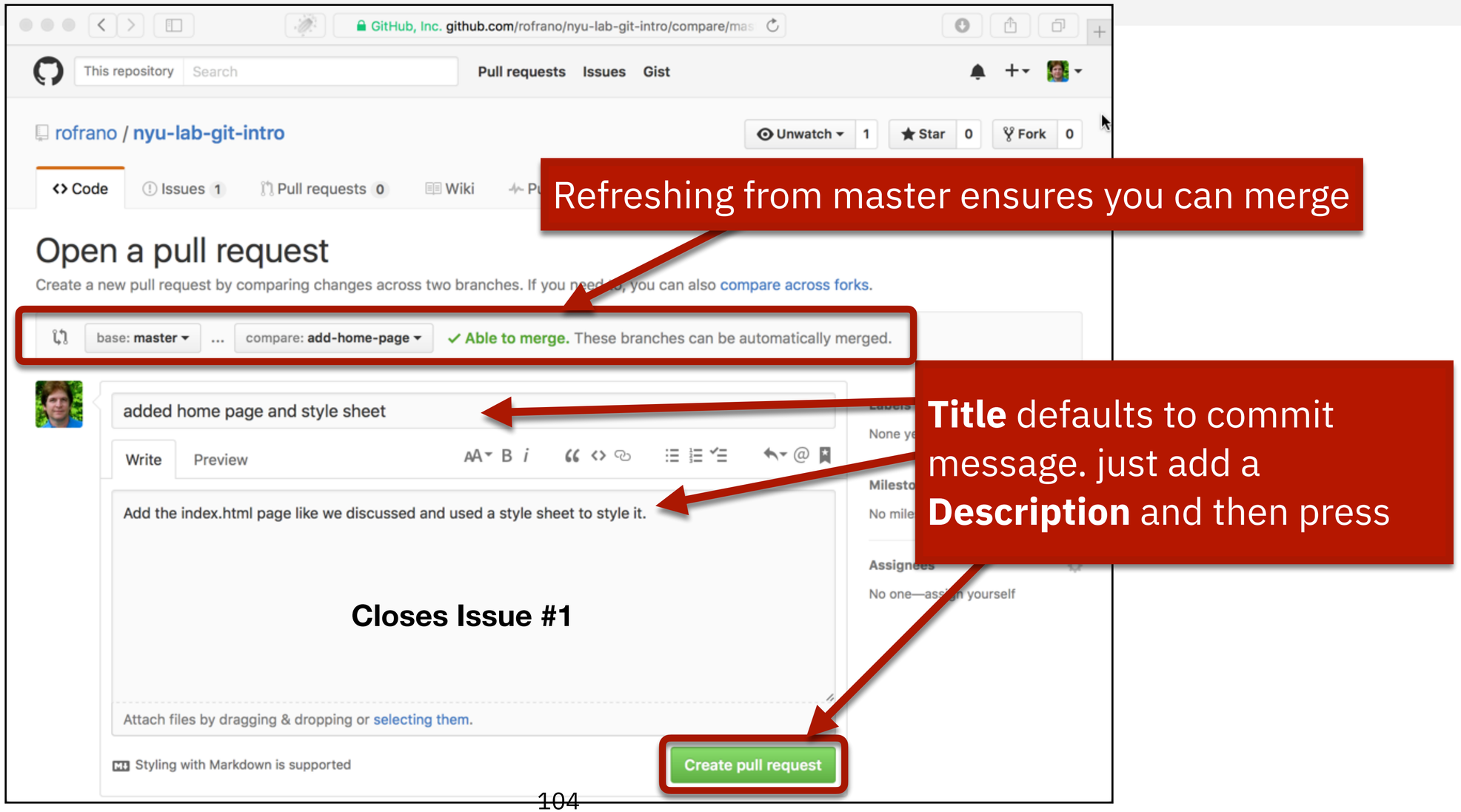
Create Pull Request



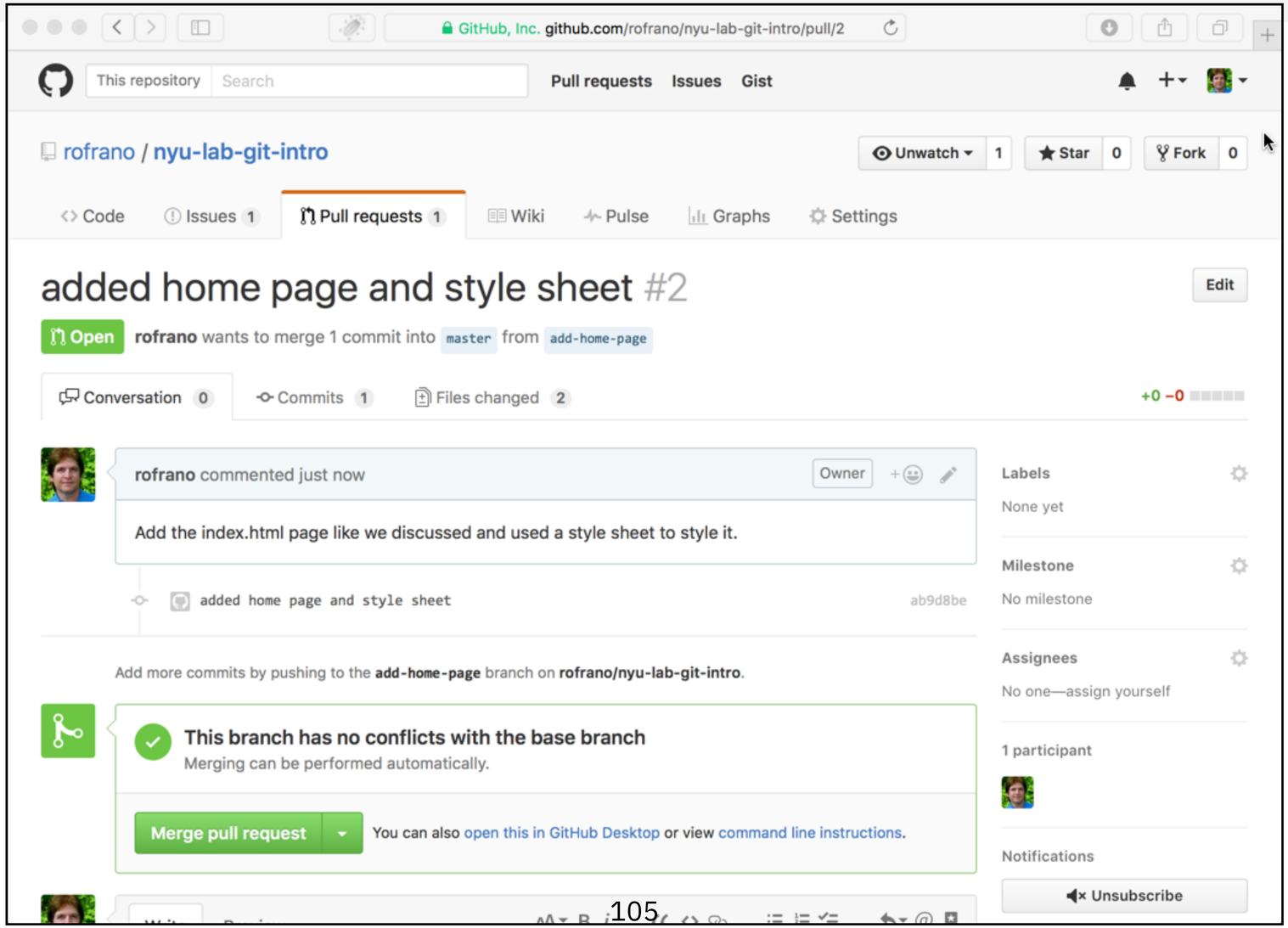
Create Pull Request



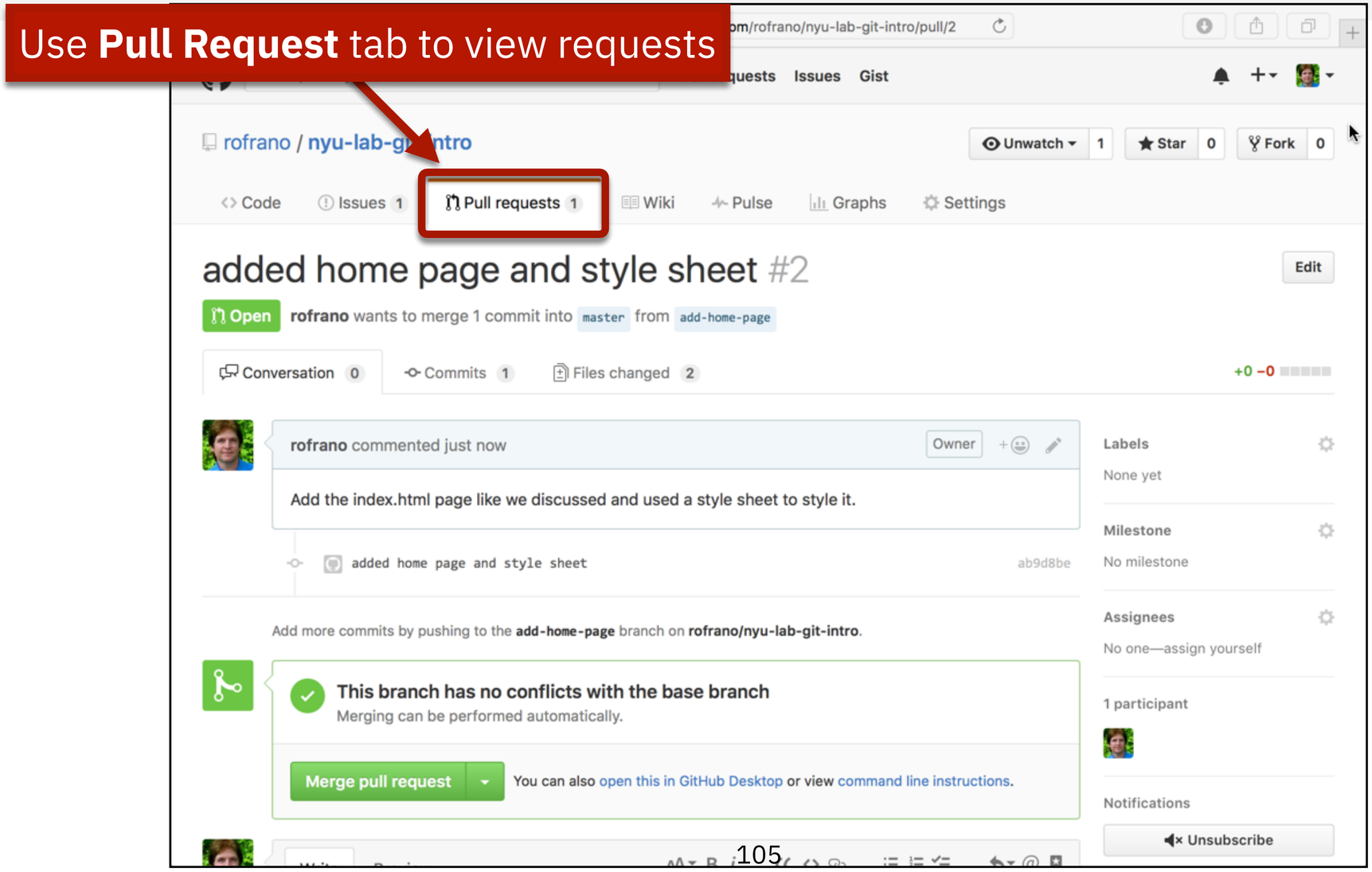
Create Pull Request



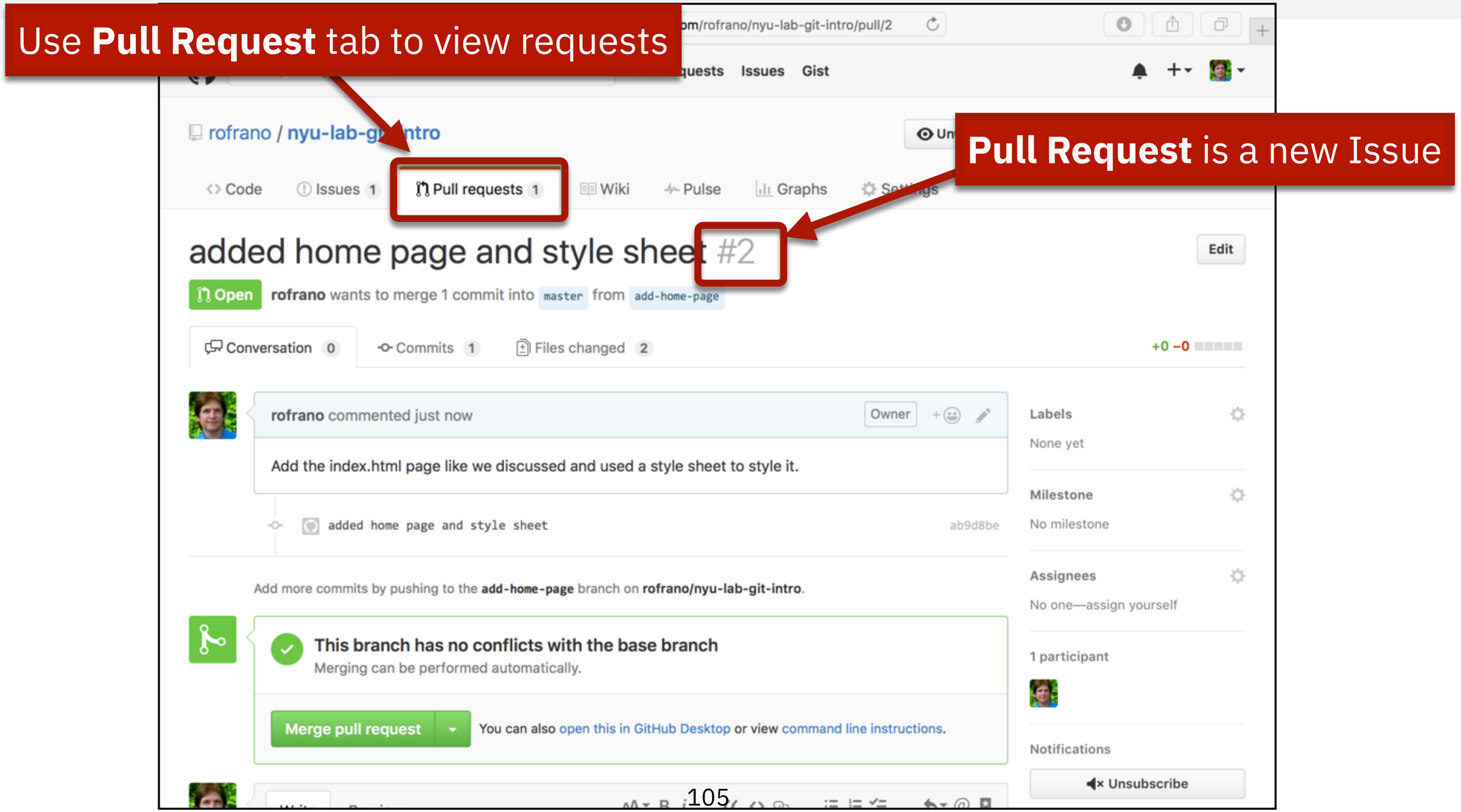
Pull Request Details



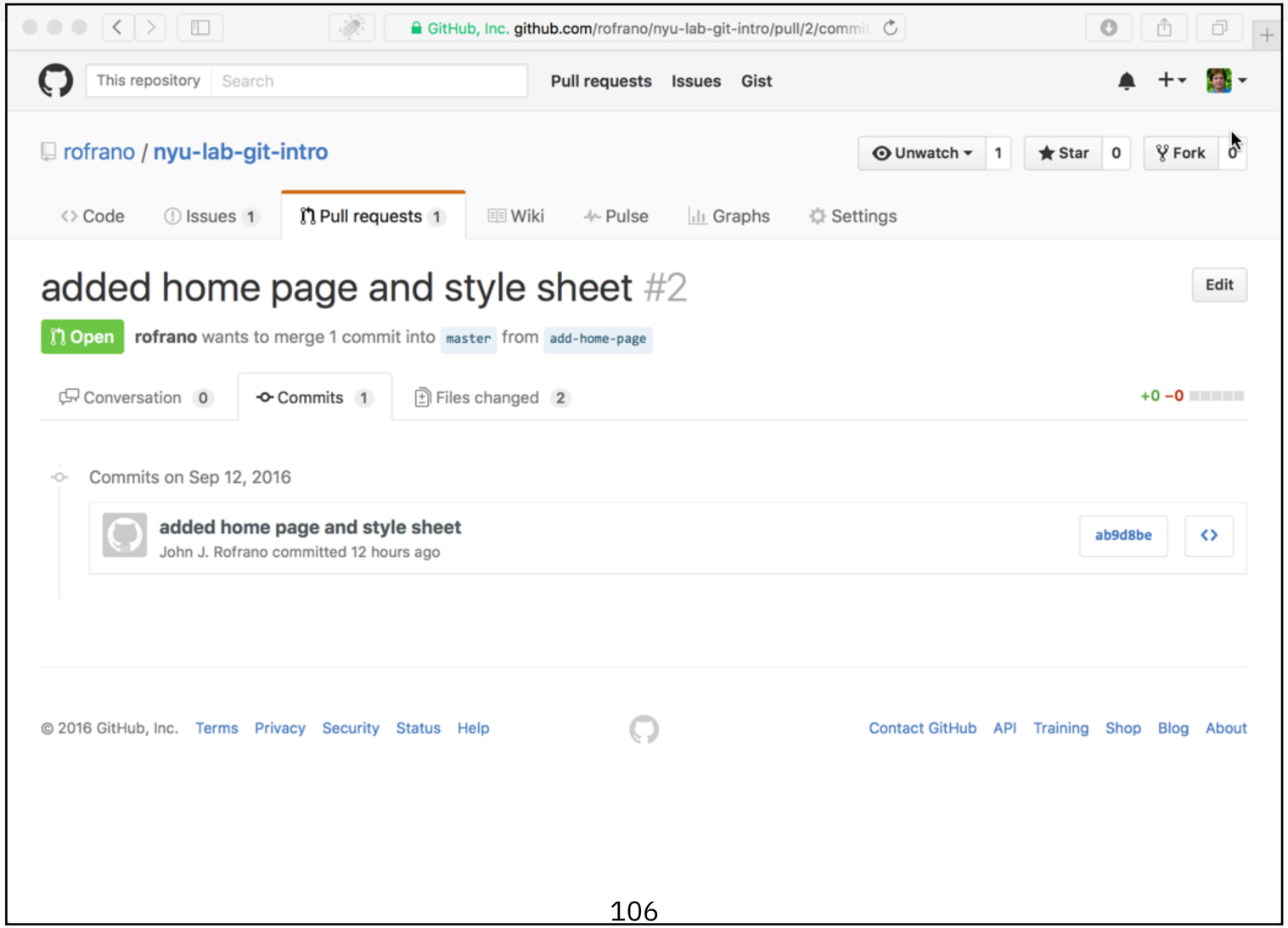
Pull Request Details



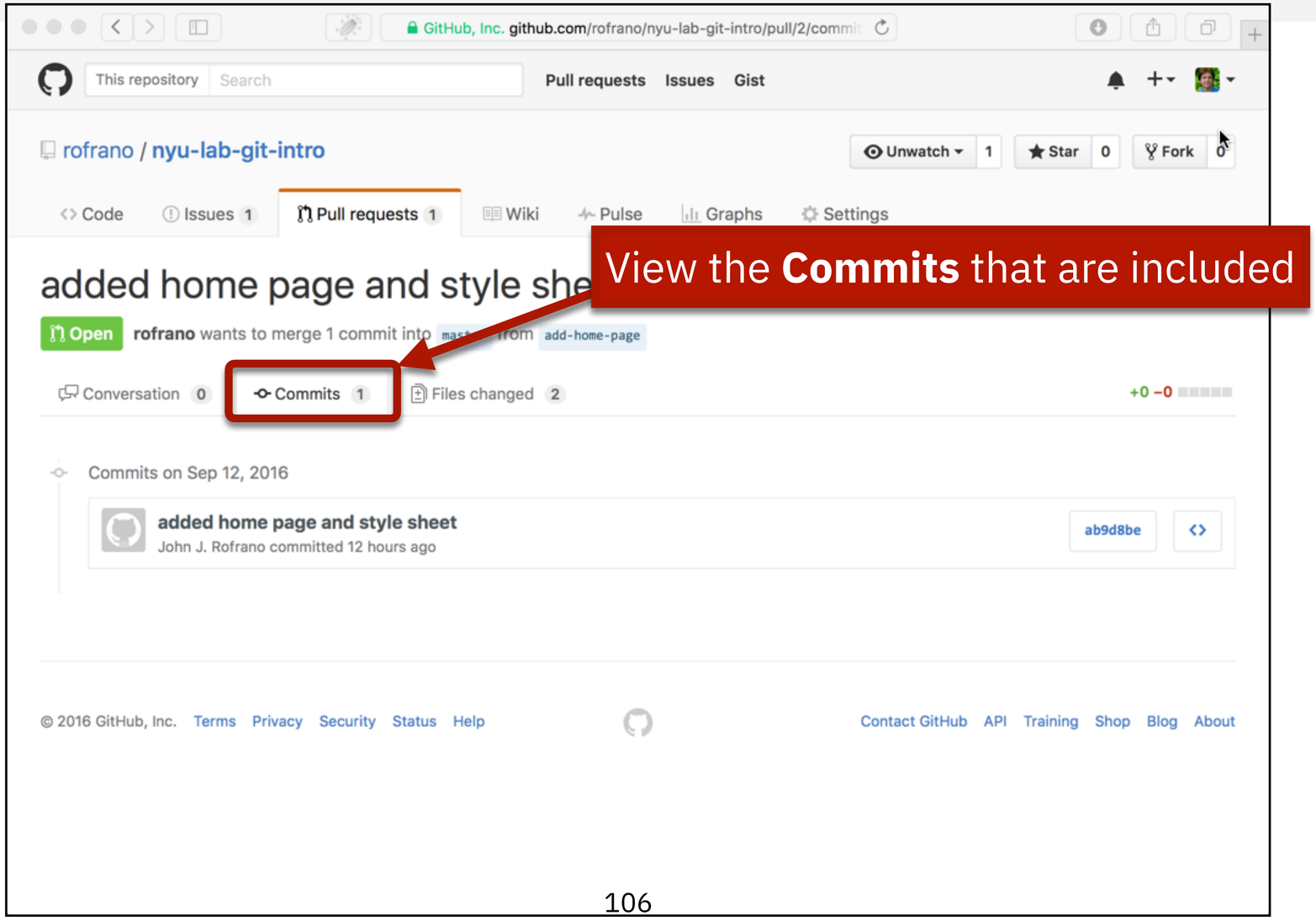
Pull Request Details



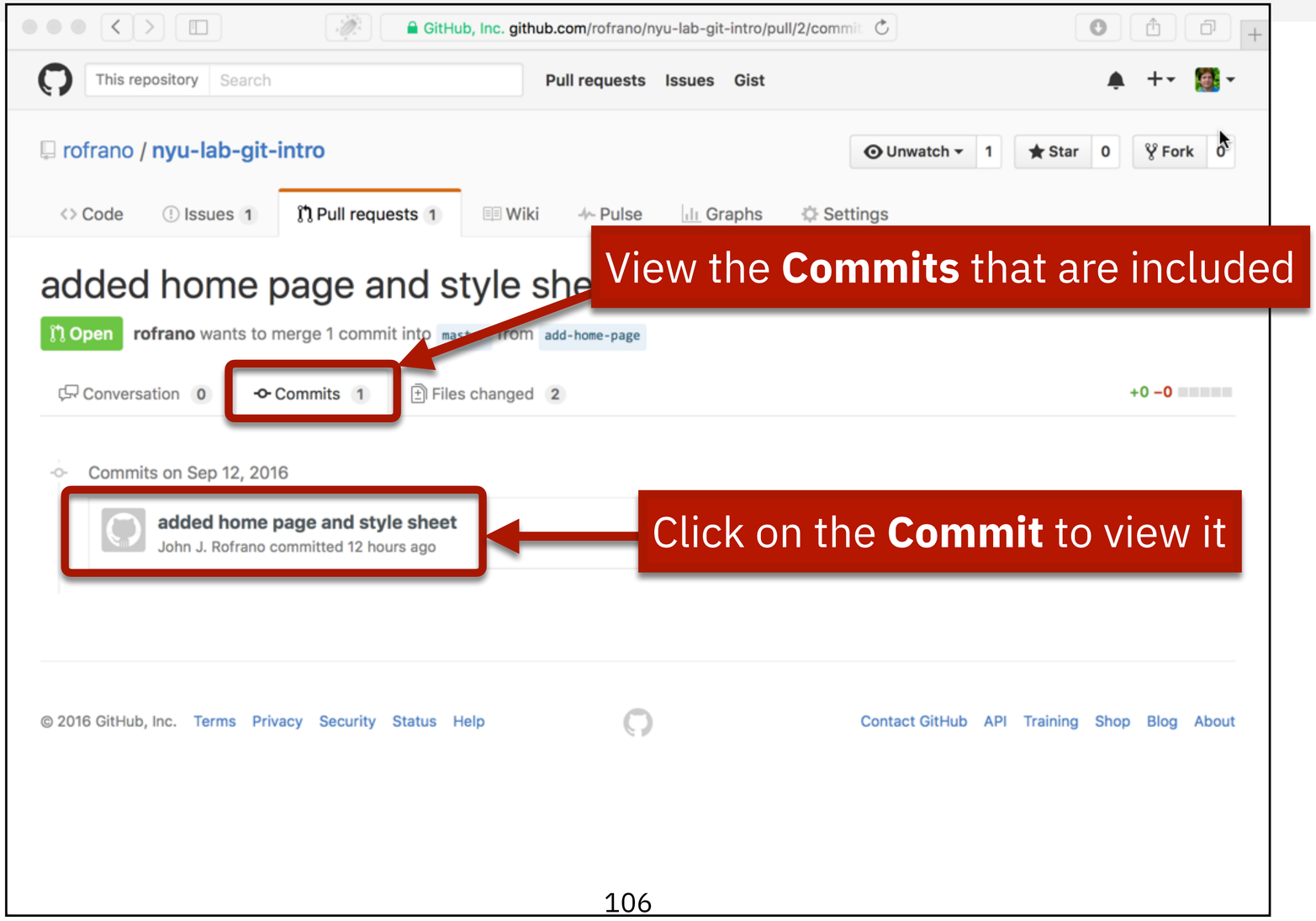
View Commits



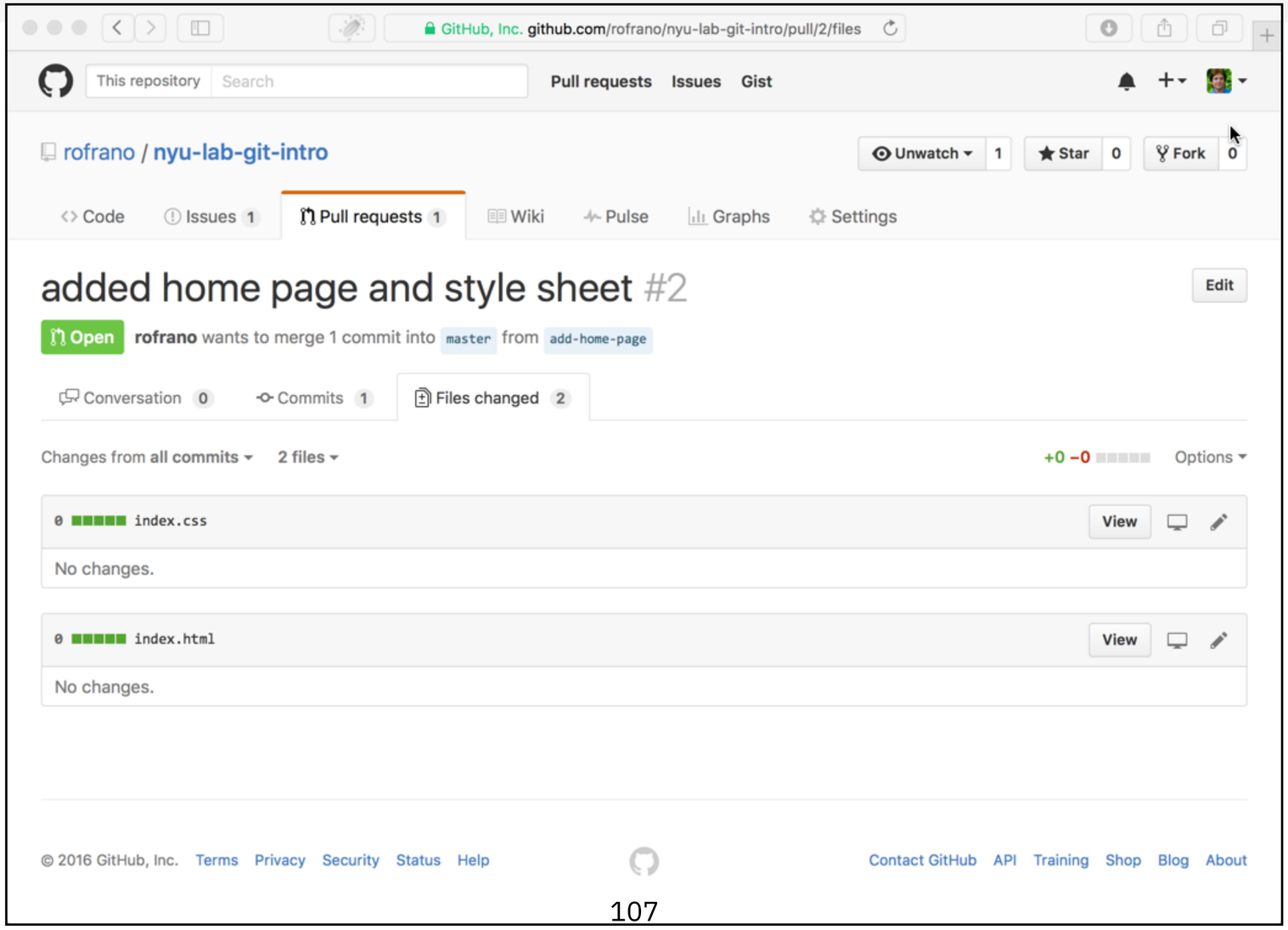
View Commits



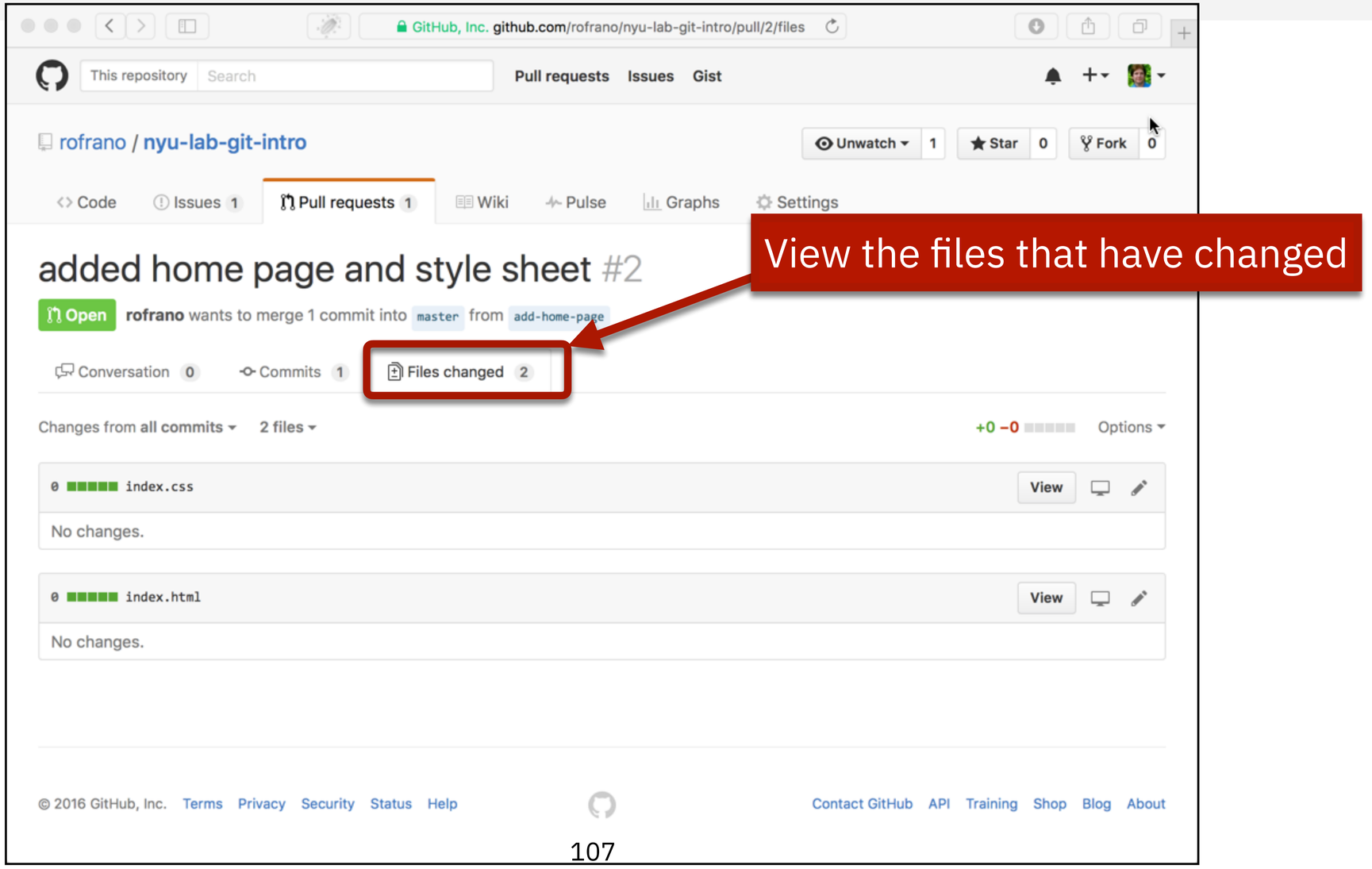
View Commits



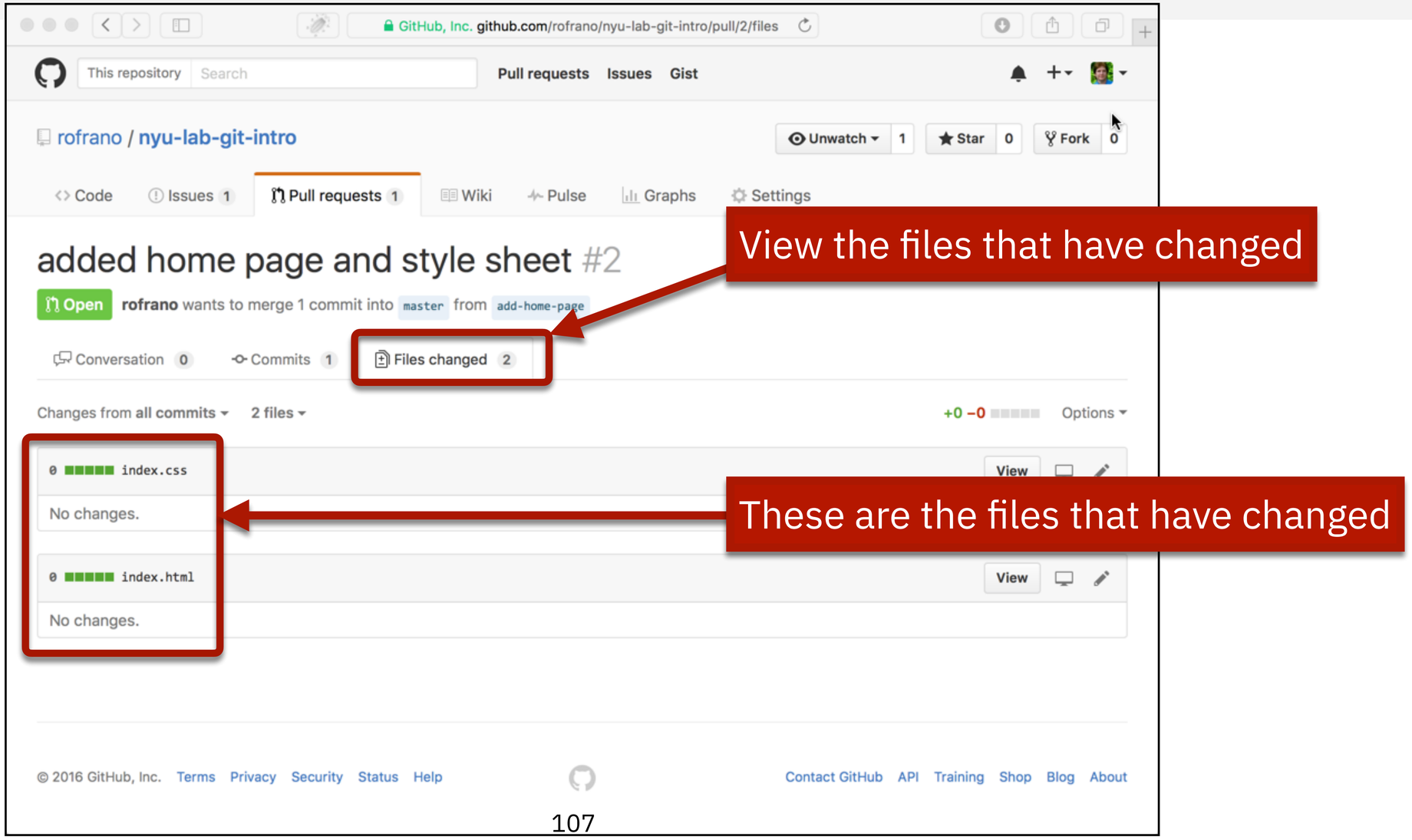
Files Changed



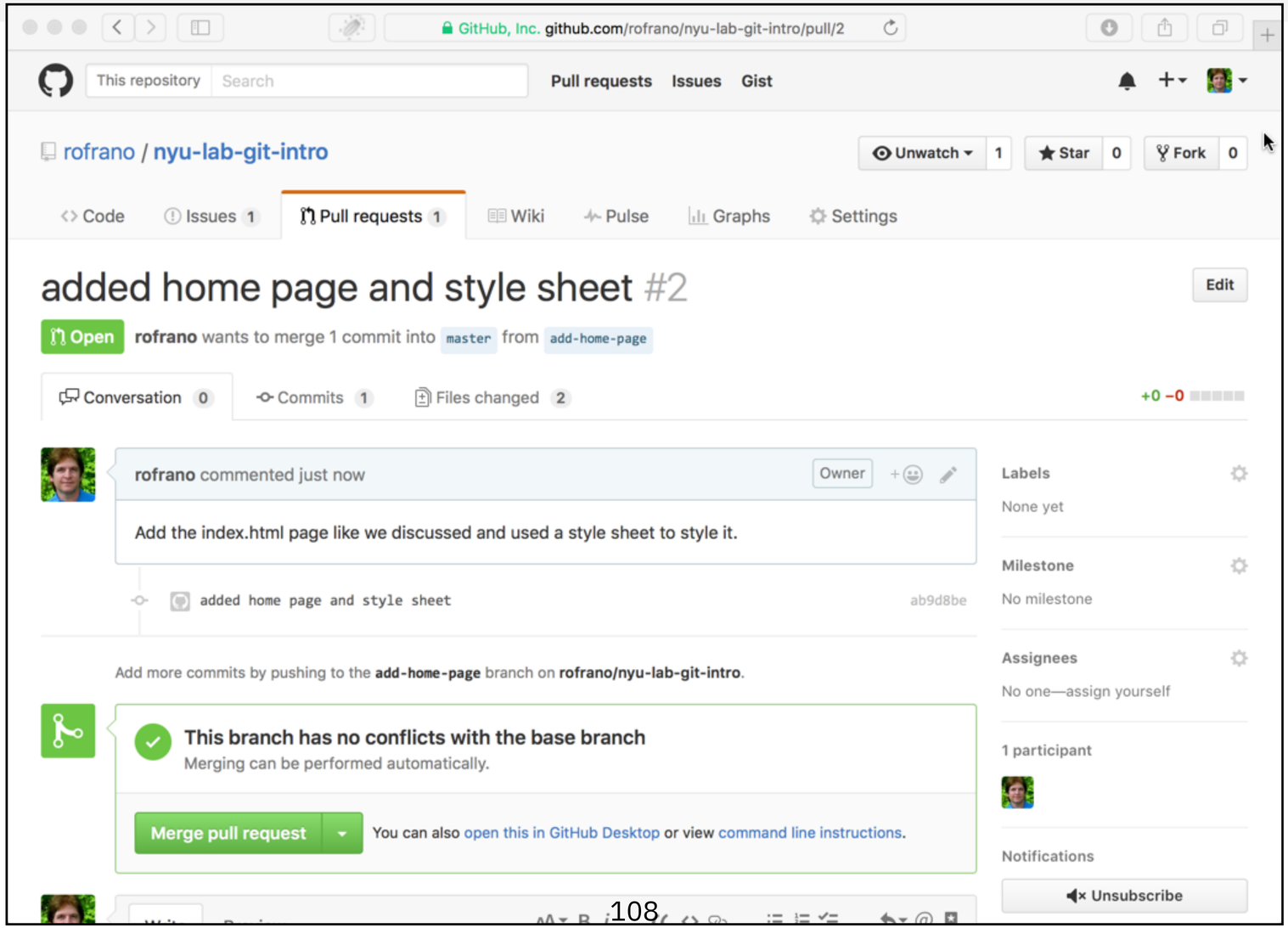
Files Changed



Files Changed

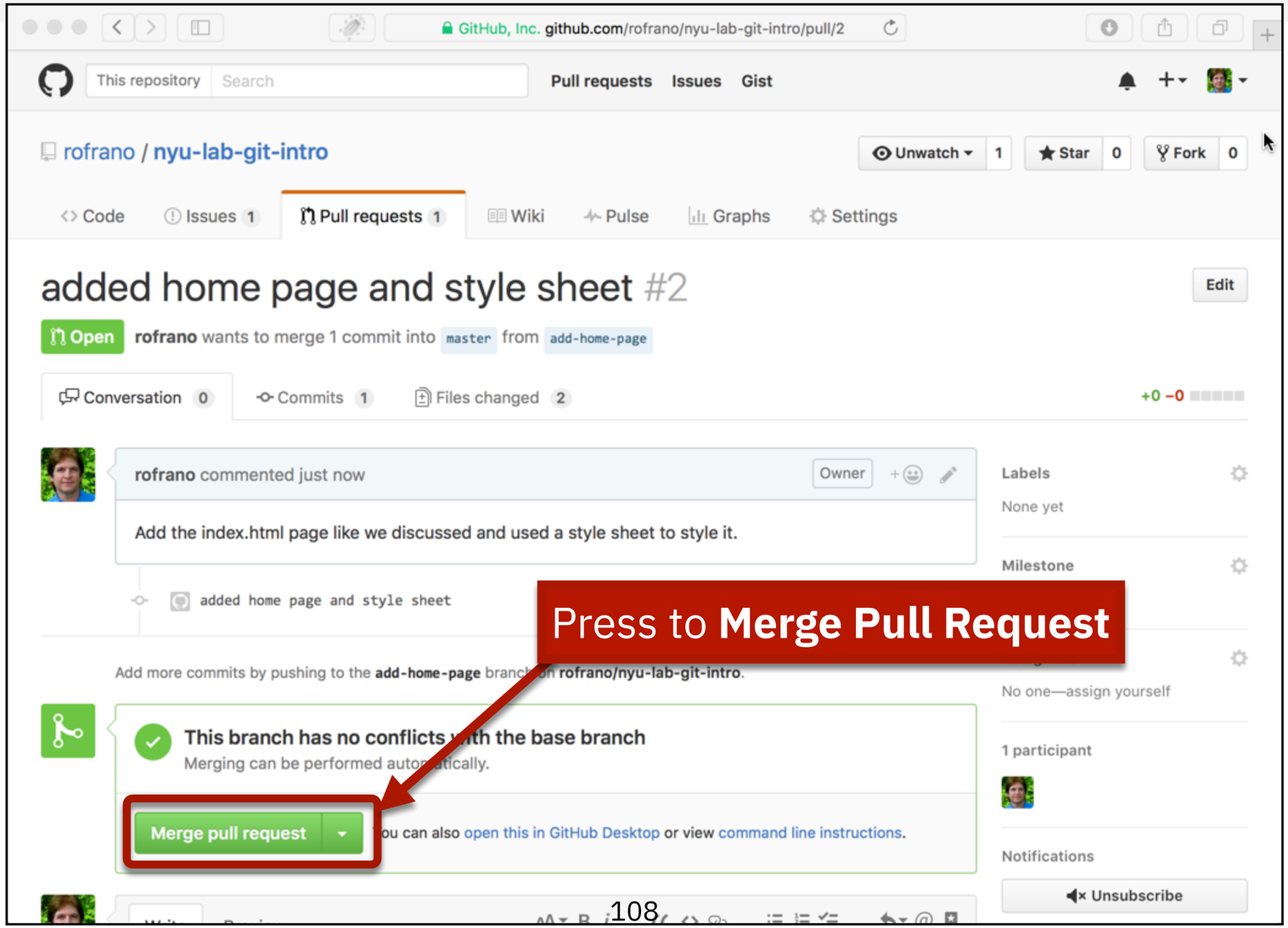


Merge Pull Request



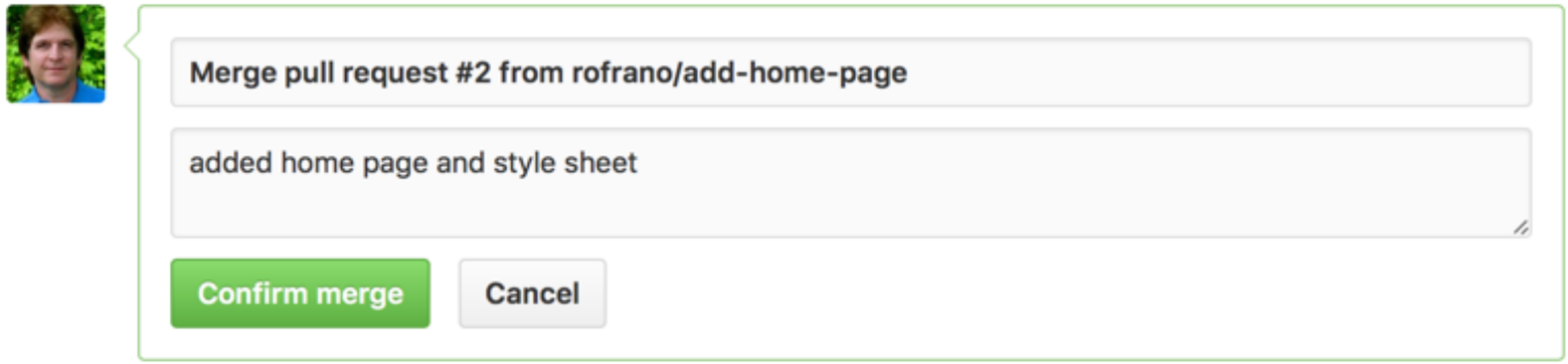
Note: Testing can be automated with tools like Travis CI, Jenkins, etc.

Merge Pull Request

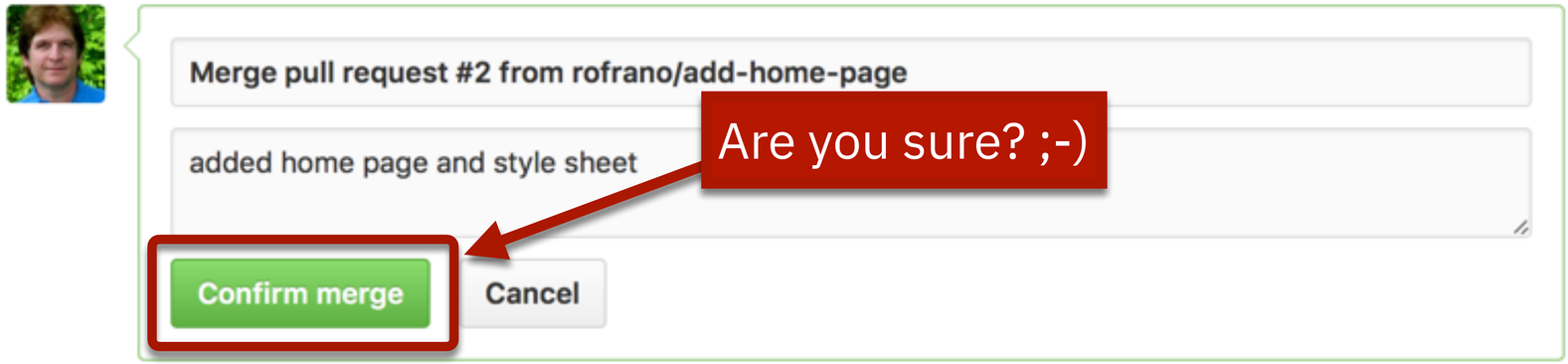


Note: Testing can be automated with tools like Travis CI, Jenkins, etc.

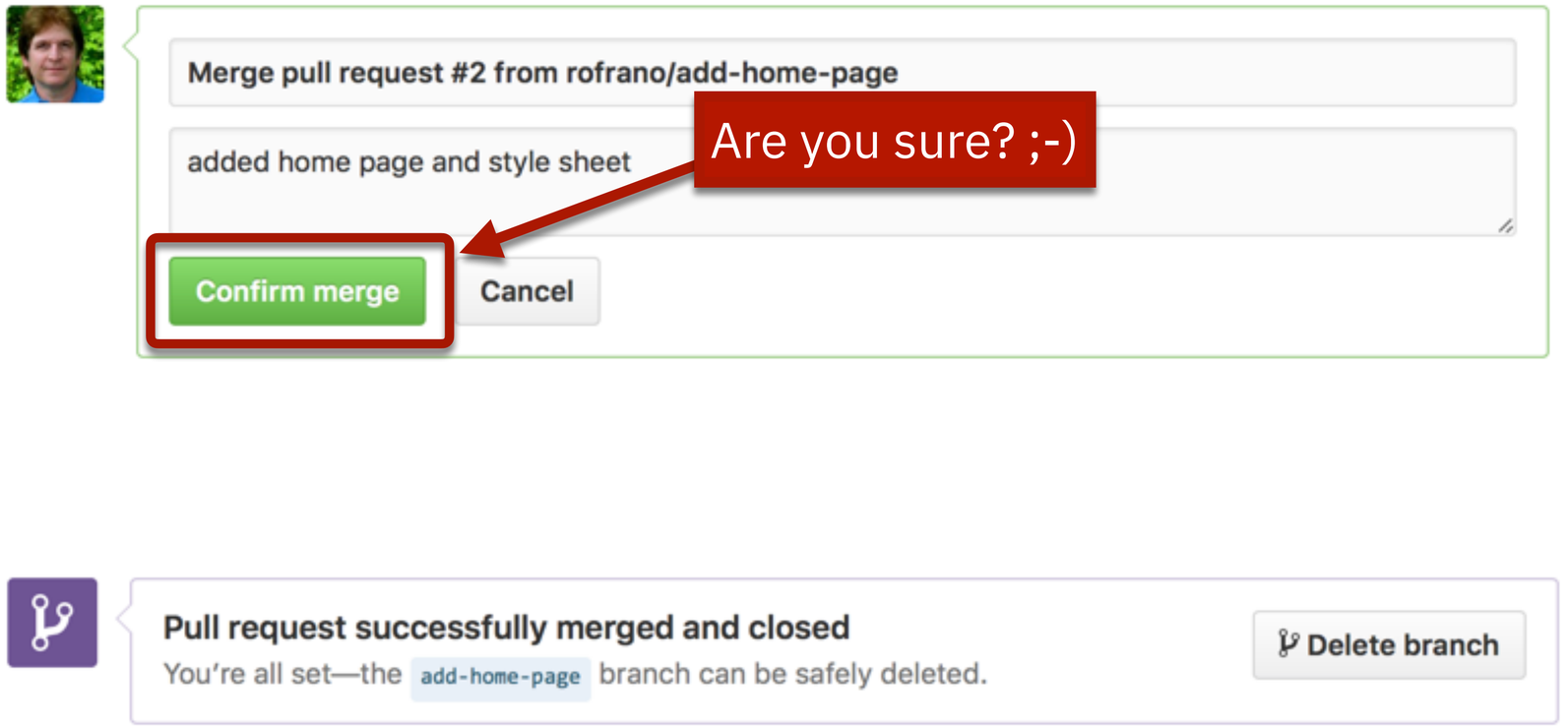
Complete the Merge



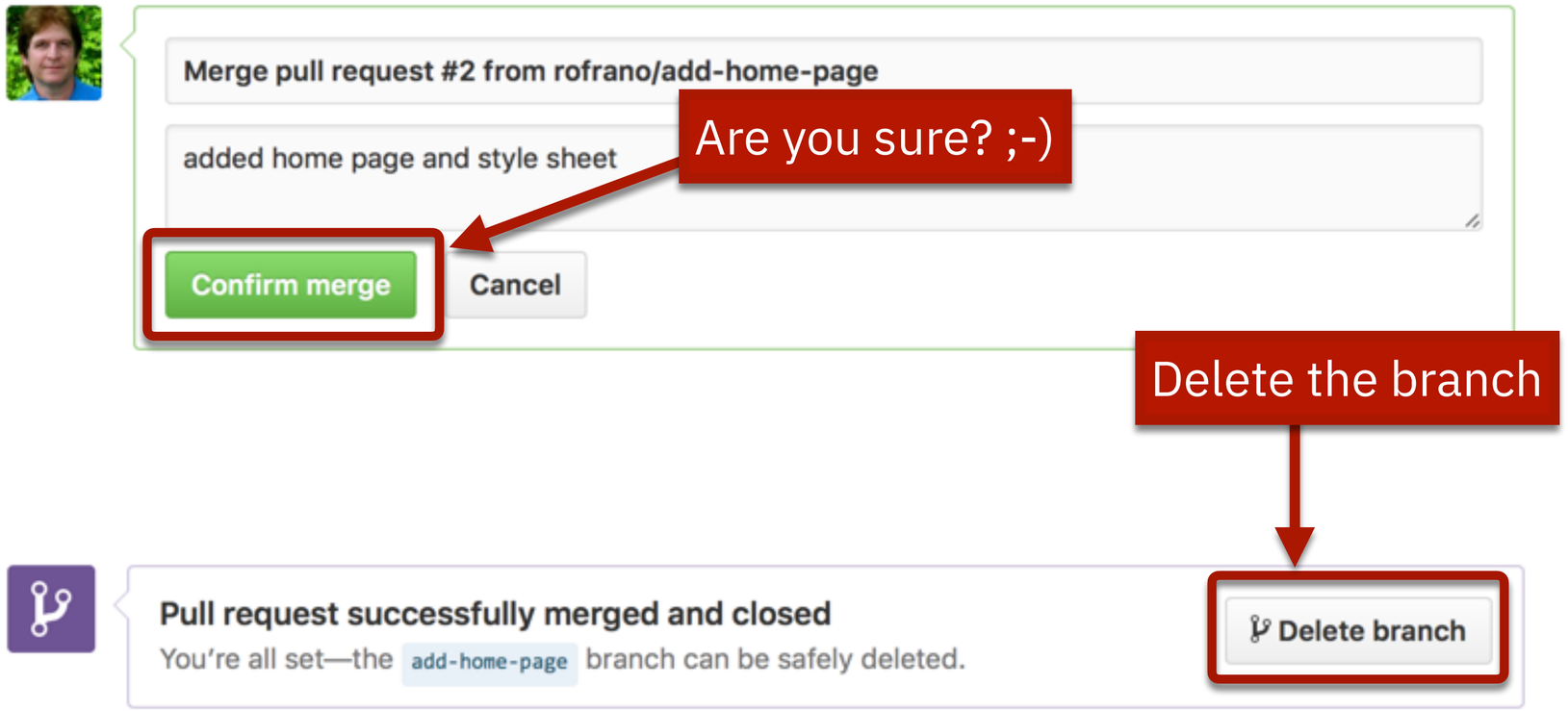
Complete the Merge



Complete the Merge



Complete the Merge



Delete Your Local Branch

- You will need to delete your local branch manually
- Make sure you are not on the branch that you want to delete

```
$ git branch
* add-home-page
  master
$ git checkout master
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.
$ git branch -d add-home-page
Deleted branch add-home-page (was ab9d8be).
$ git branch
* master
$
```

Delete Your Local Branch

- You will need to delete your local branch manually
- Make sure you are not on the branch that you want to delete

```
$ git branch
* add-home-page
  master
$ git checkout master
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.
$ git branch -d add-home-page
Deleted branch add-home-page (was ab9d8be).
$ git branch
* master
$
```

Why does it think we are up-to-date?
(we just changed master via pull request)

Prune Missing Branches

- You can list all branches that can be removed because the remote upstream branch does not exist anymore.
- Removing `--dry-run` will delete the remote branches in your local repo that don't exist anymore on the remote repo

```
$ git branch -a
* master
  remotes/origin/HEAD -> origin/master
  remotes/origin/add-home-page
  remotes/origin/master

$ git remote prune origin --dry-run
Pruning origin
URL: git@github.com:rofrano/nyu-lab-git-intro.git
 * [would prune] origin/add-home-page
$
```

Refresh from the new Master

- Don’t forget to fetch the latest changes and pull them into master

```
$ git fetch
remote: Counting objects: 1, done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (1/1), done.
From github.com:rofrano/nyu-lab-git-intro
   907cda6..5f7a2de  master     -> origin/master
$
```

Refresh from the new Master

- Switch to the **master** branch so that we can **pull** the changes

```
$ git fetch
remote: Counting objects: 1, done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (1/1), done.
From github.com:rofrano/nyu-lab-git-intro
   907cda6..5f7a2de  master     -> origin/master
$ git checkout master
Switched to branch 'master'
Your branch is behind 'origin/master' by 2 commits, and can be fast-forwarded.
  (use "git pull" to update your local branch)
$
```

Refresh from the new Master

- Issue a `git pull` to bring down the new code

```
$ git fetch -p
remote: Counting objects: 1, done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (1/1), done.
From github.com:rofrano/nyu-lab-git-intro
   907cda6..5f7a2de  master     -> origin/master
$ git checkout master
Switched to branch 'master'
Your branch is behind 'origin/master' by 2 commits, and can be fast-forwarded.
  (use "git pull" to update your local branch)
$ git pull
Updating 907cda6..5f7a2de
Fast-forward
 index.css  | 0
 index.html | 0
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 index.css
 create mode 100644 index.html
$
```

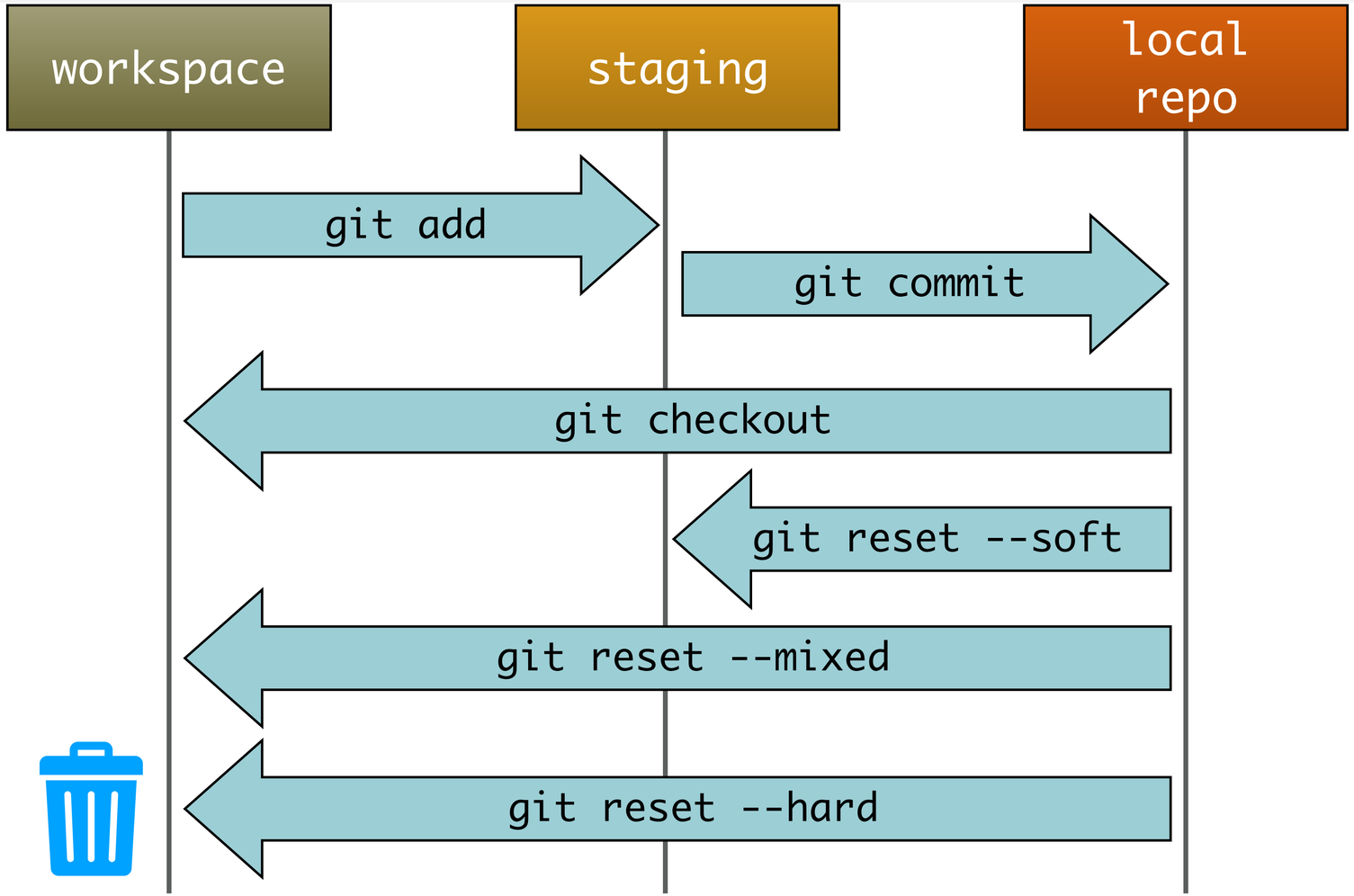
What About Mistakes?

- If you have NOT pushed to a remote branch
- You can use `RESET` to go back to a previous commit
- Reset rewinds the commits as if they never happened

RESET

- To get rid of the last two commits
  - `git reset --soft HEAD~2`
  - `git reset --hard master@{1}` (undo fast forward merge)
- **--soft** will undo the commit history and leave the files staged
- **--mixed** will undo the history and the staging so the files are untracked (default)
- **--hard** will undo and delete your files!!!

Summary of reset Movements



REVERT

- If you made a commit and pushed it to the remote, how to do undo it because other people may have access to the remote?

  – `git revert <sha1>`

- Revert is the only *safe* undo for a commit that has a remote push
- Note: revert will not remove files that are pushed remotely

Reflog and Revert

- You can use `reflog` to see your activities and `revert` them
- Reflog shows the history of all the commands issued in git

```
$ git reflog
ab9d8be HEAD@{0}: commit: added home page and style sheet
907cda6 HEAD@{1}: checkout: moving from master to add-home-page
907cda6 HEAD@{2}: clone: from git@github.com:rofrano/nyu-lab-git-intro.git
$
```

How Reset and Revert are Different

- **Reset** will alter history as if it never existed
  - This will confuse remote repos so once you push a commit you should not reset it
- **Revert** will undo history by adding a commit that reverses the changes
  - This is remote repo friendly

Fun Facts about this Class...

- You will not be working alone in GitHub
- Someone on your team will edit the same file that you do
- There will be merge conflicts!
- These are not the conflicts you're looking for
- There is no fear...
- There is only the FORCE

Merge and Conflicts

- Git can usually figure out how to merge code as long as different lines are changed
- Git cannot merge code that has the same line of code changed in two different branches
- This requires manual intervention and probably collaboration with the developer who made the other changes

Conflict Scenario

- All it takes to create a merge conflict is for two branches to change the same line in a common file
- For this scenario we will change the file `index.html`
- One developer is working on a story to make the index say 'hello'
- Another developer is working on a story to make the index say 'welcome'

Let's Create Some Conflicts!

😈😈😈

✓ Create a local git repo called `merge-conflict`

```
$ mkdir merge-conflict
$ cd merge-conflict
$ git init
Initialized empty Git repository in /home/vagrant/merge-conflict/.git/
$ git status
On branch master

Initial commit

nothing to commit (create/copy files and use "git add" to track)
$
```

Add A File

✓ Add an `index.html` file and commit the change as 'add index'

```
$ touch index.html
$ git add index.html
$ git commit -m 'add index'
[master (root-commit) 1e796aa] add index
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 index.html
$ git reflog
1e796aa HEAD@{0}: commit (initial): add index
$ git log --oneline --decorate --all --graph
* 1e796aa (HEAD -> master) add index
$ git status
On branch master
nothing to commit, working directory clean
$
```

Create the Hello Branch

✓ Create a branch called 'hello' and edit the 1st line of index.html and commit

```
$ git checkout -b hello
Switched to a new branch 'hello'
$ echo 'Hello' > index.html
$ cat index.html
Hello
$ git status
On branch hello
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

  modified:   index.html

no changes added to commit (use "git add" and/or "git commit -a")
$ git commit -am 'added hello to index'
[hello 92b526a] added hello to index
 1 file changed, 1 insertion(+)
$
```

Create the Welcome Branch

✓ Switch back to master

✓ Create a branch called 'welcome' and edit the 1st line of index.html and commit

```
$ git checkout master
Switched to branch 'master'
$ cat index.html
$ git checkout -b welcome
Switched to a new branch 'welcome'
$ echo 'Welcome' > index.html
$ cat index.html
Welcome
$ git commit -am 'added welcome'
[welcome 5e8197b] added welcome
 1 file changed, 1 insertion(+)
$
```

Where Are We At?

✓ Let's see where we're at with `log` and `reflog`

```
$ git log --oneline --decorate --all --graph
* 5e8197b (HEAD -> welcome) added welcome
| * 92b526a (hello) added hello to index
|/
* 1e796aa (master) add index

$ git reflog
5e8197b HEAD@{0}: commit: added welcome
1e796aa HEAD@{1}: checkout: moving from master to welcome
1e796aa HEAD@{2}: checkout: moving from hello to master
92b526a HEAD@{3}: commit: added hello to index
1e796aa HEAD@{4}: checkout: moving from master to hello
1e796aa HEAD@{5}: commit (initial): add index
```

Customize Log Command

- You can create a nice looking log command my using git aliases

```
git config --global alias.lg "log --oneline --decorate --all --graph"
```

- Or a really fancy one with formatting

```
git config --global alias.lg "log --decorate --all --graph --
pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold
blue)<%an>%Creset'"
```

- Then just use `git lg` to use it

Let's Simulate a Pull Request

√ To simulate a pull request lets merge welcome into master

```
$ git checkout master
Switched to branch 'master'
$ cat index.html
$ git merge welcome
Updating 1e796aa..5e8197b
Fast-forward
 index.html | 1 +
 1 file changed, 1 insertion(+)

$ cat index.html
Welcome
$ git log --oneline --decorate --all --graph
* 5e8197b (HEAD -> master, welcome) added welcome
| * 92b526a (hello) added hello to index
|/
* 1e796aa add index
$
```

How Did The Log Change?

Before

```
$ git log --oneline --decorate --all --graph
* 5e8197b (HEAD -> welcome) added welcome
| * 92b526a (hello) added hello to index
|/
* 1e796aa (master) add index
```

After

```
$ git log --oneline --decorate --all --graph
* 5e8197b (HEAD -> master, welcome) added welcome
| * 92b526a (hello) added hello to index
|/
* 1e796aa add index
$
```

- Notice that **master** simply moved forward to join **welcome**
- The commit **sha1** is the same as welcome (**5e8197b**)
- This is called a **fast-forward**

How Did The Log Change?

Before

```
$ git log --oneline --decorate --all --graph
* 5e8197b (HEAD -> welcome) added welcome
| * 92b526a (hello) added hello to index
|/
* 1e796aa (master) add index
```

After

```
$ git log --oneline --decorate --all --graph
* 5e8197b (HEAD -> master, welcome) added welcome
| * 92b526a (hello) added hello to index
|/
* 1e796aa add index
$
```

- Notice that **master** simply moved forward to join **welcome**
- The commit **sha1** is the same as welcome (**5e8197b**)
- This is called a **fast-forward**

Simulate Pull Request With Hello

✓ Try to simulate a pull request by merging hello into master

```
$ git status
On branch master
nothing to commit, working directory clean

$ git merge hello
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.

$ cat index.html
<<<<<<< HEAD
Welcome
=======
Hello
>>>>>>> hello
$ git merge --abort
```

Simulate Pull Request With Hello

✓ Try to simulate a pull request by merging hello into master

```
$ git status
On branch master
nothing to commit, working directory clean

$ git merge hello
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.

$ cat index.html
<<<<<<< HEAD
Welcome
=======
Hello
>>>>>>> hello
$ git merge --abort
```

The dreaded merge conflict 🔥

What Happened?

- The **hello** branch and **welcome** branch were both direct descendants of **master** (*this is key*)
- The **hello** branch modified the same line as the **welcome** branch
- Git needs you to tell it which version of the line to use
- *Hint:* Your pull request would have had a conflict

What's A Possible Fix?

✓ The owner of hello should merge their work on the current version of master before submitting the pull request

```
$ git checkout hello
Switched to branch 'hello'

$ git merge master
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the
result.

$ cat index.html
<<<<<<< HEAD
Hello
=======
Welcome
>>>>>>> master
```

Different Perspectives

- Note how the merge is reflected depending on your current branch

Perspective from master branch

```
$ git checkout master
$ git merge hello
$ cat index.html
<<<<<<< HEAD
Welcome
=======
Hello
>>>>>>> hello
```

Perspective from hello branch

```
$ git checkout hello
$ git merge master
$ cat index.html
<<<<<<< HEAD
Hello
=======
Welcome
>>>>>>> master
```

Different Perspectives

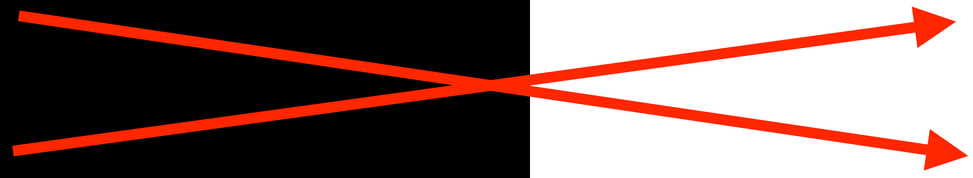
- Note how the merge is reflected depending on your current branch

Perspective from master branch

```
$ git checkout master
$ git merge hello
$ cat index.html
<<<<<<< HEAD
Welcome
=======
Hello
>>>>>>> hello
```

Perspective from hello branch

```
$ git checkout hello
$ git merge master
$ cat index.html
<<<<<<< HEAD
Hello
=======
Welcome
>>>>>>> master
```



Different Perspectives

- Note how the merge is reflected depending on your current branch

Perspective from master branch

```
$ git checkout master
$ git merge hello
$ cat index.html
<<<<<<< HEAD
Welcome
=======
Hello
>>>>>>> hello
```

Perspective from hello branch

```
$ git checkout
$ git merge ma
$ cat index.html
<<<<<<< HEAD
Hello
=======
Welcome
>>>>>>> master
```

HEAD always points to your branch

Resolve The Conflict

✓ You must edit the file in conflict and then tell rebase to continue

```
$ echo "Hello and Welcome" > index.html
$ git status
On branch hello
You have unmerged paths.
  (fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)

  both modified:   index.html

no changes added to commit (use "git add" and/or "git commit -a")
$ git commit -am 'fix merge conflict'
[hello 218eb38] fix merge conflict
$
```

Results of the merge

Before

```
$ git log --oneline --decorate --all --graph
* 5e8197b (HEAD -> master, welcome) added welcome
| * 92b526a (hello) added hello to index
|/
* 1e796aa add index
```

After

```
$ git log --oneline --decorate --all --graph
*   218eb38 (HEAD -> hello) fix merge conflict
|\
| * 5e8197b (welcome, master) added welcome
* | 92b526a added hello to index
|/
* 1e796aa add index
```

- Notice that **hello** is now the HEAD
- We are now ready to merge this back into **master**

Results of the merge

Before

```
$ git log --oneline --decorate --all --graph
* 5e8197b (HEAD -> master, welcome) added welcome
| * 92b526a (hello) added hello to index
|/
* 1e796aa add index
```

After

```
$ git log --oneline --decorate --all --graph
*   218eb38 (HEAD -> hello) fix merge conflict
|\
| * 5e8197b (welcome, master) added welcome
* | 92b526a added hello to index
|/
* 1e796aa add index
```

- Notice that hello is now the HEAD
- We are now ready to merge this back into master

Continue Pull Request Simulation

√ To simulate a pull request lets merge hello into master

```
$ git checkout master
Switched to branch 'master'

$ cat index.html
Welcome

$ git merge hello
Updating 5e8197b..218eb38
Fast-forward
 index.html | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

$ git log --oneline --decorate --all --graph
*   218eb38 (HEAD -> master, hello) fix merge conflict
|\
| * 5e8197b (welcome) added welcome
* | 92b526a added hello to index
|/
* 1e796aa add index

$ cat index.html
Hello and Welcome
```

View the Log Ref

✓ You can always use git reflog to see all of the work done

```
$ git reflog
218eb38 HEAD@{0}: merge hello: Fast-forward
5e8197b HEAD@{1}: checkout: moving from hello to master
218eb38 HEAD@{2}: commit (merge): fix merge conflict
92b526a HEAD@{3}: checkout: moving from master to hello
5e8197b HEAD@{4}: merge welcome: Fast-forward
1e796aa HEAD@{5}: checkout: moving from welcome to master
5e8197b HEAD@{6}: commit: added welcome
1e796aa HEAD@{7}: checkout: moving from master to welcome
1e796aa HEAD@{8}: checkout: moving from hello to master
92b526a HEAD@{9}: commit: added hello to index
1e796aa HEAD@{10}: checkout: moving from master to hello
1e796aa HEAD@{11}: commit (initial): add index

$
```

How To Abort

- Never leave a merge or rebase in an unfinished state!
- You can abort a **merge** with:
  - `git merge --abort`
- You can abort a **rebase** with:
  - `git rebase —abort`

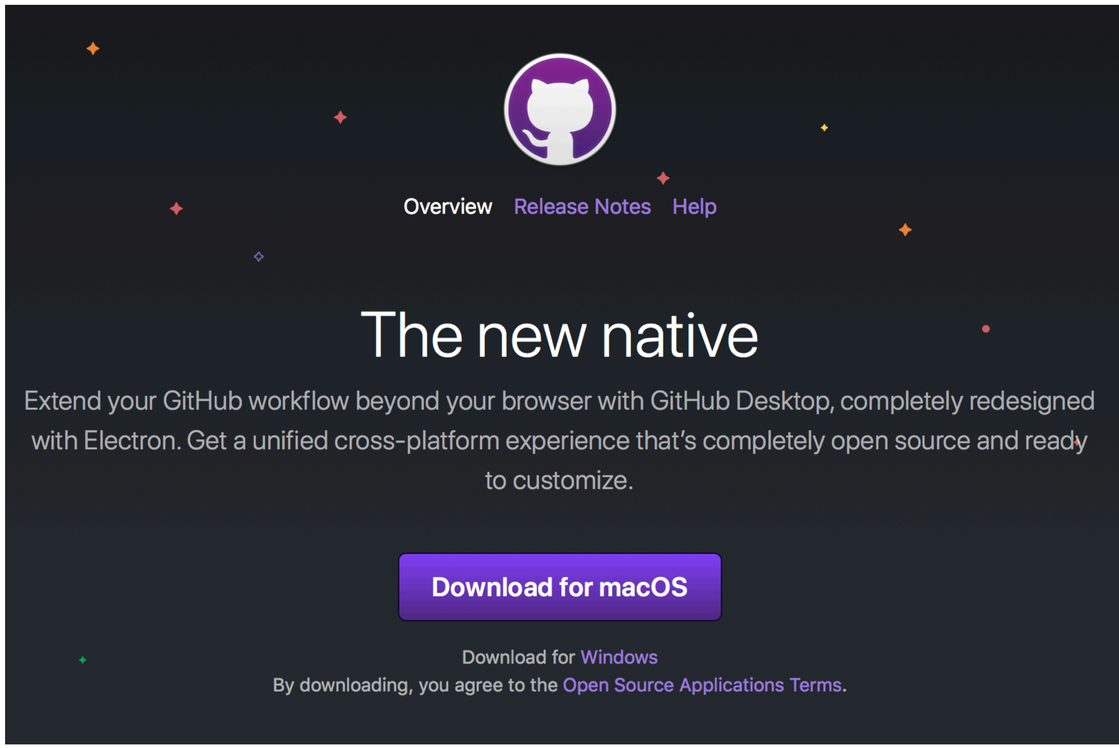
Now that you know Git

Let's look at GitHub Desktop

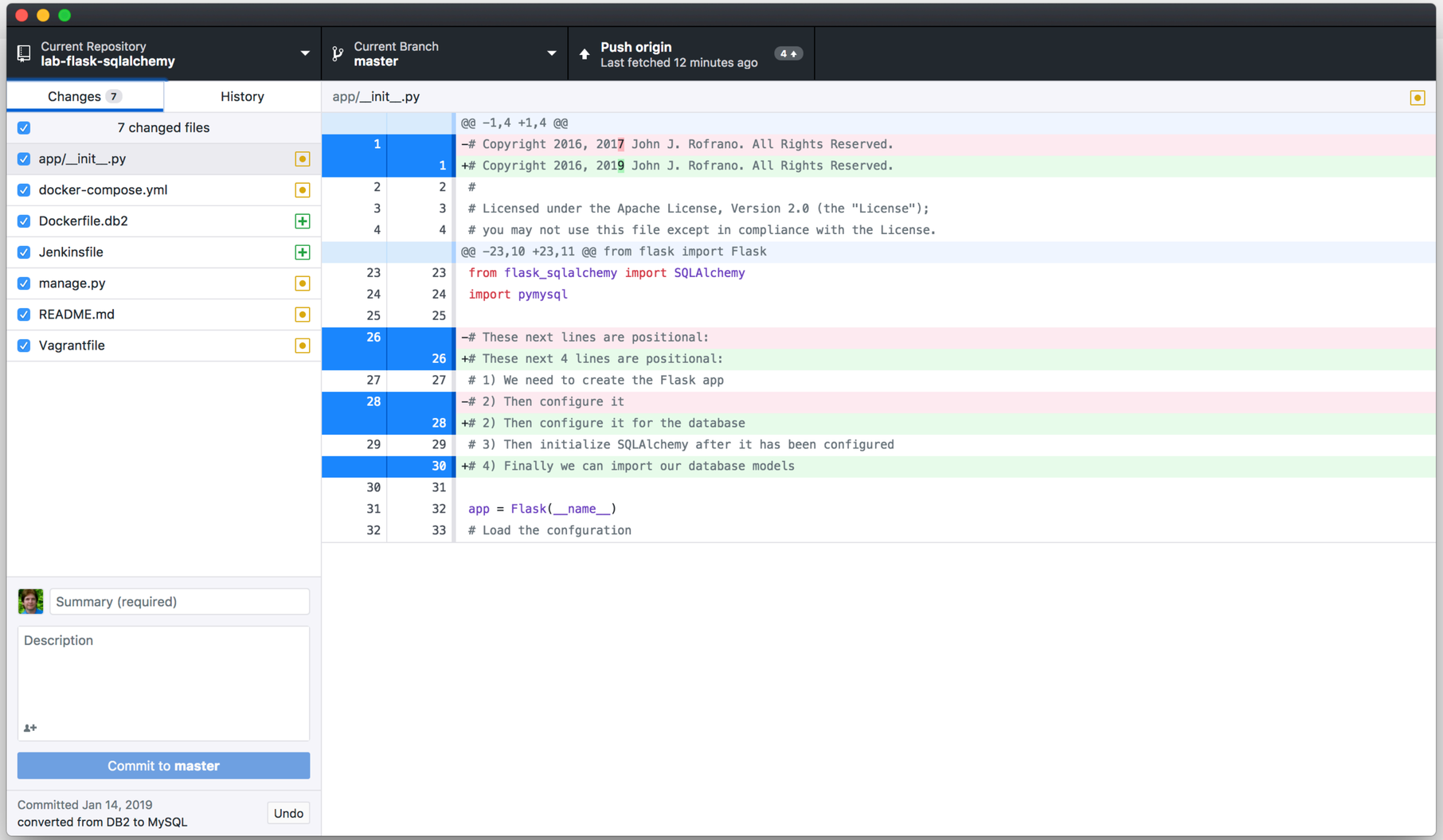
GitHub Desktop

http://desktop.github.com

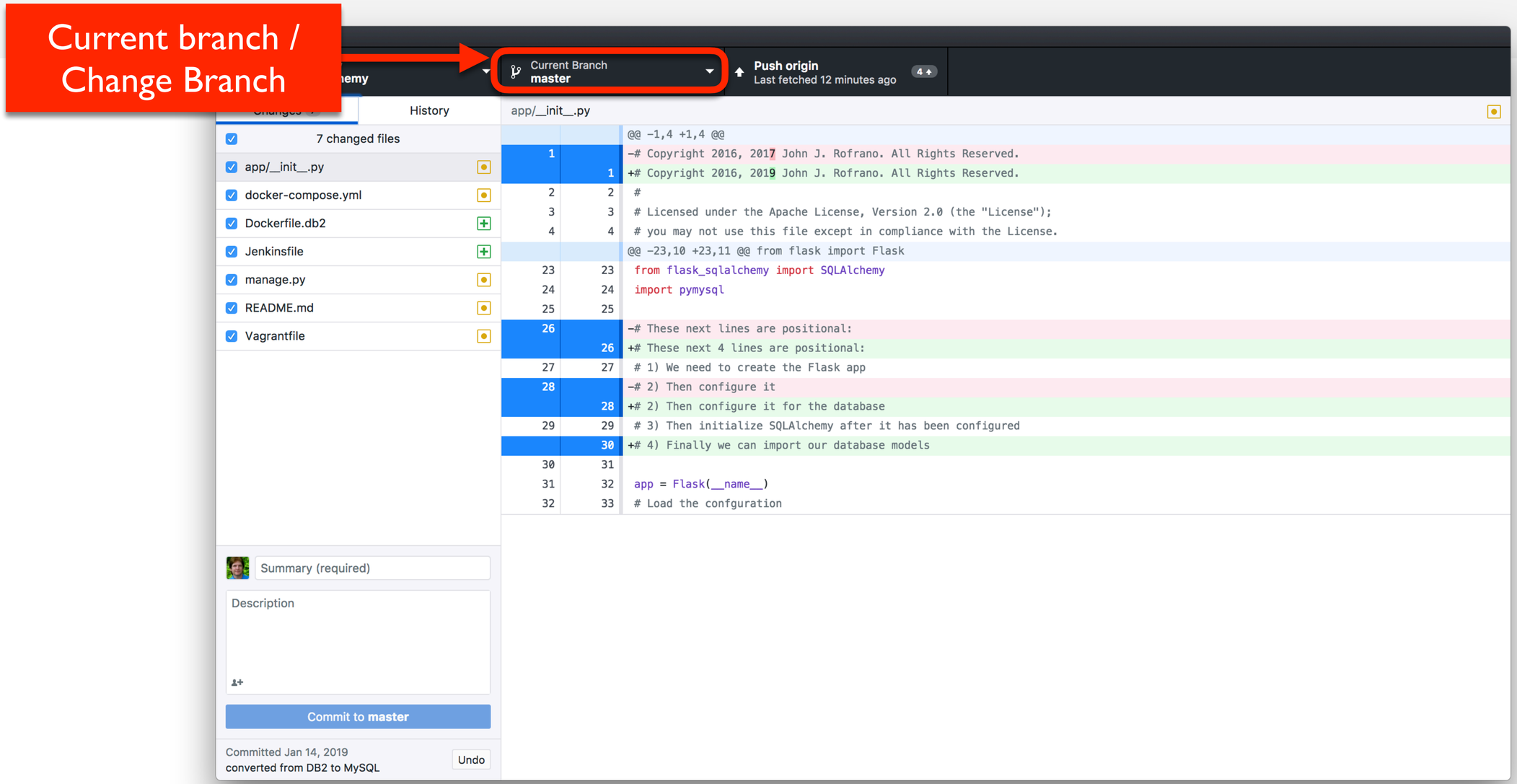
- GitHub Desktop is a Graphical User Interface for GitHub
- Makes it easy to contribute to projects on GitHub and GitHub Enterprise
- Integrates with the Atom Editor and Terminal



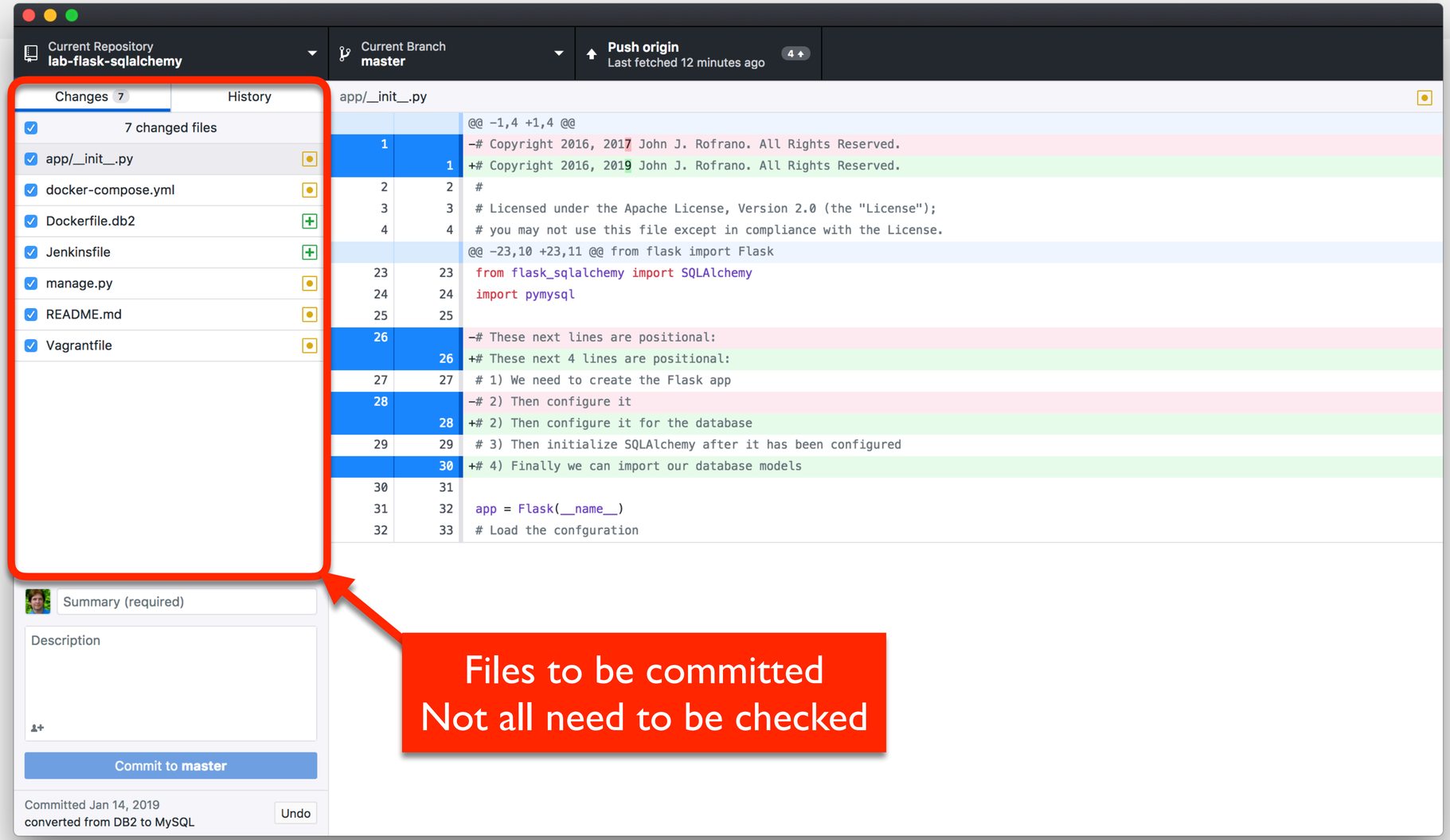
GitHub Desktop UI



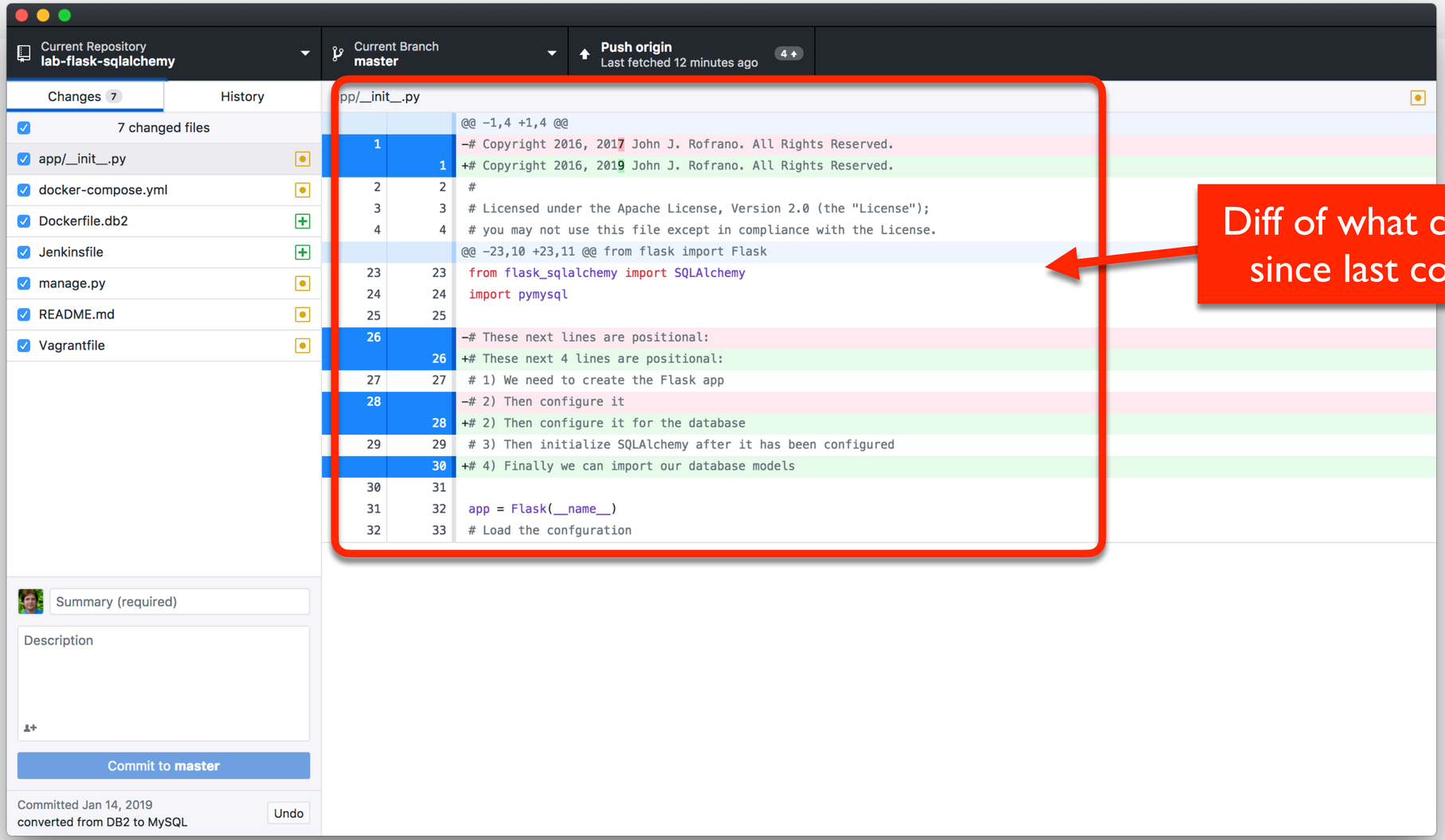
GitHub Desktop UI



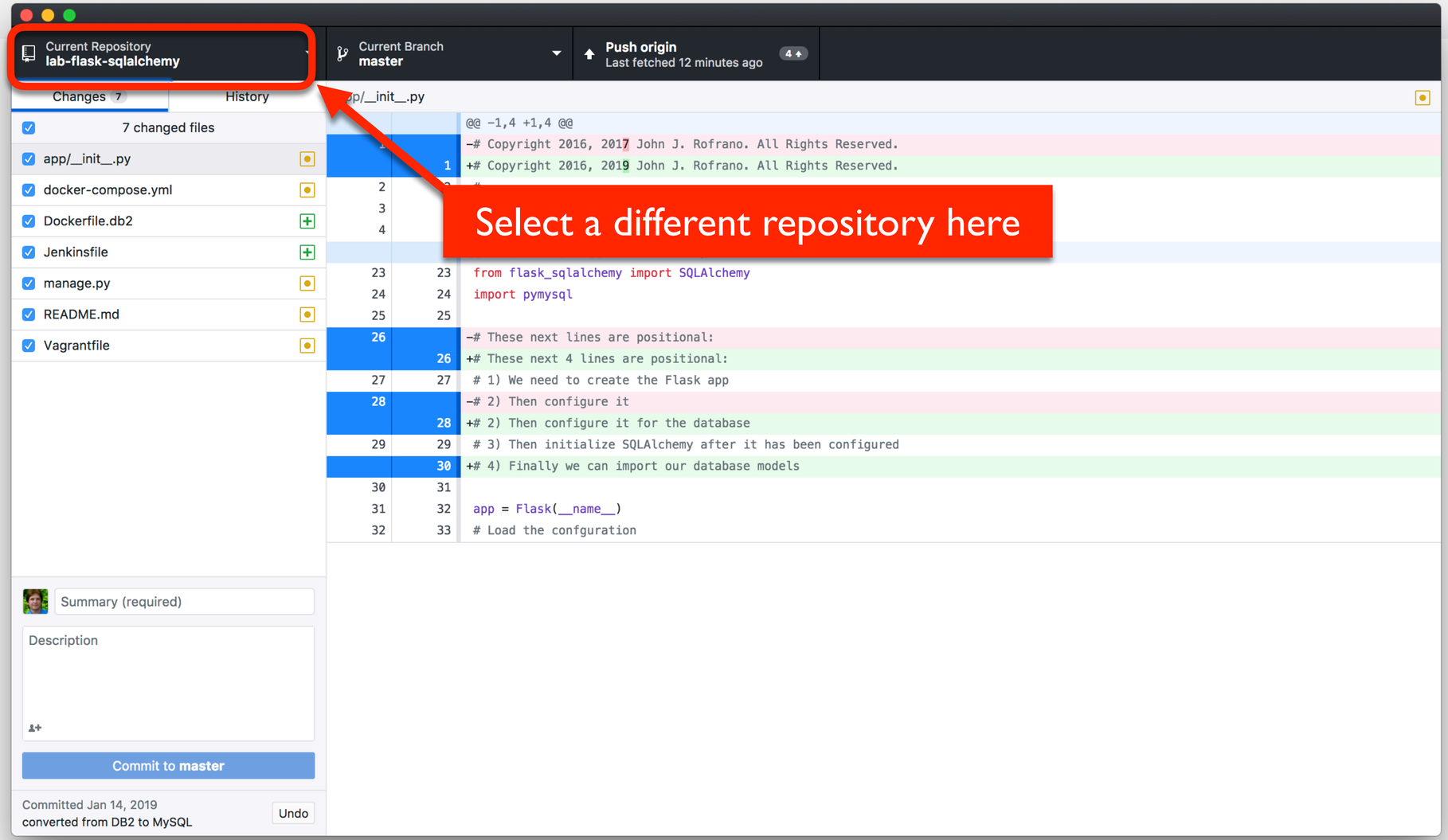
GitHub Desktop UI



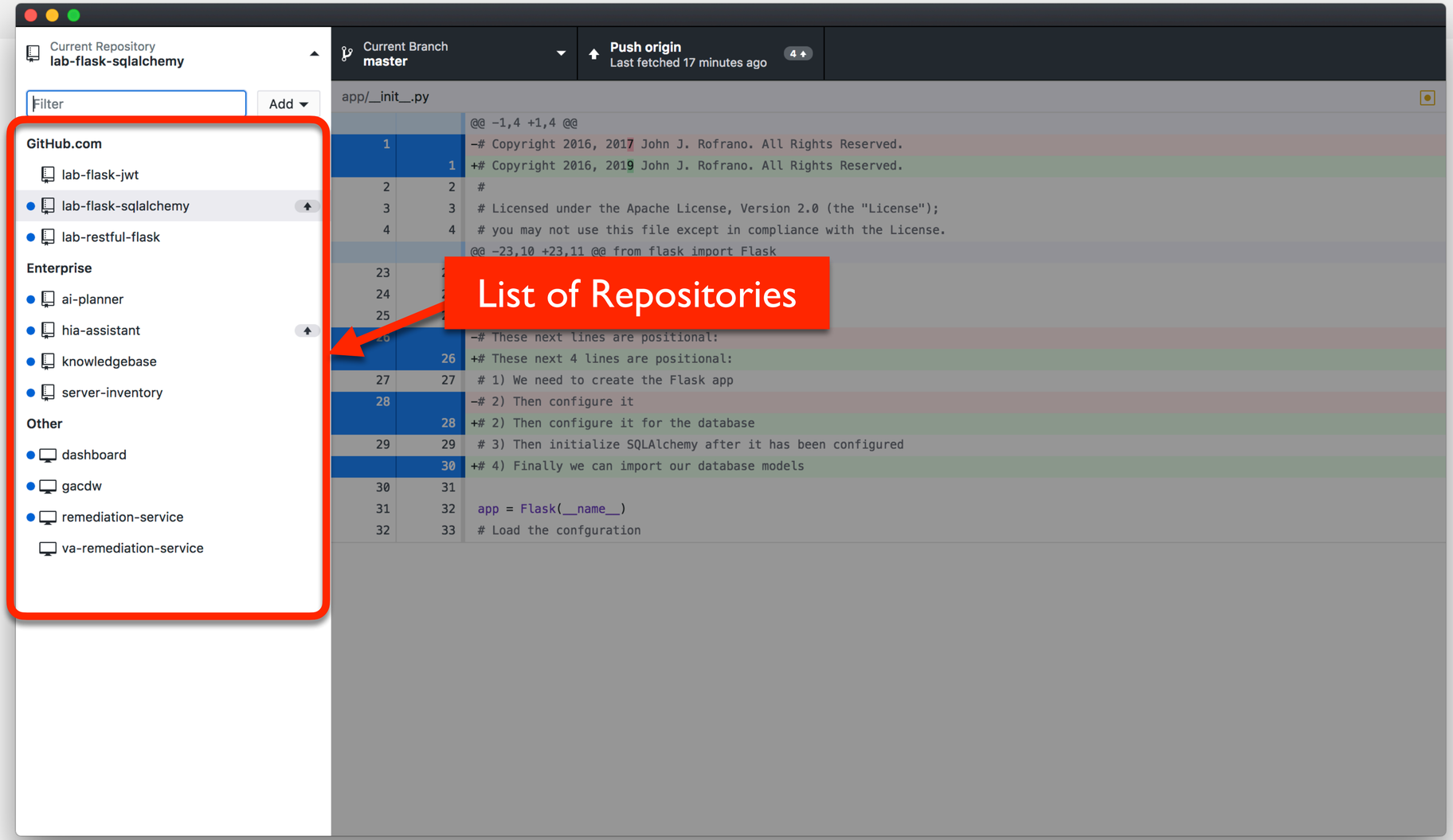
GitHub Desktop UI



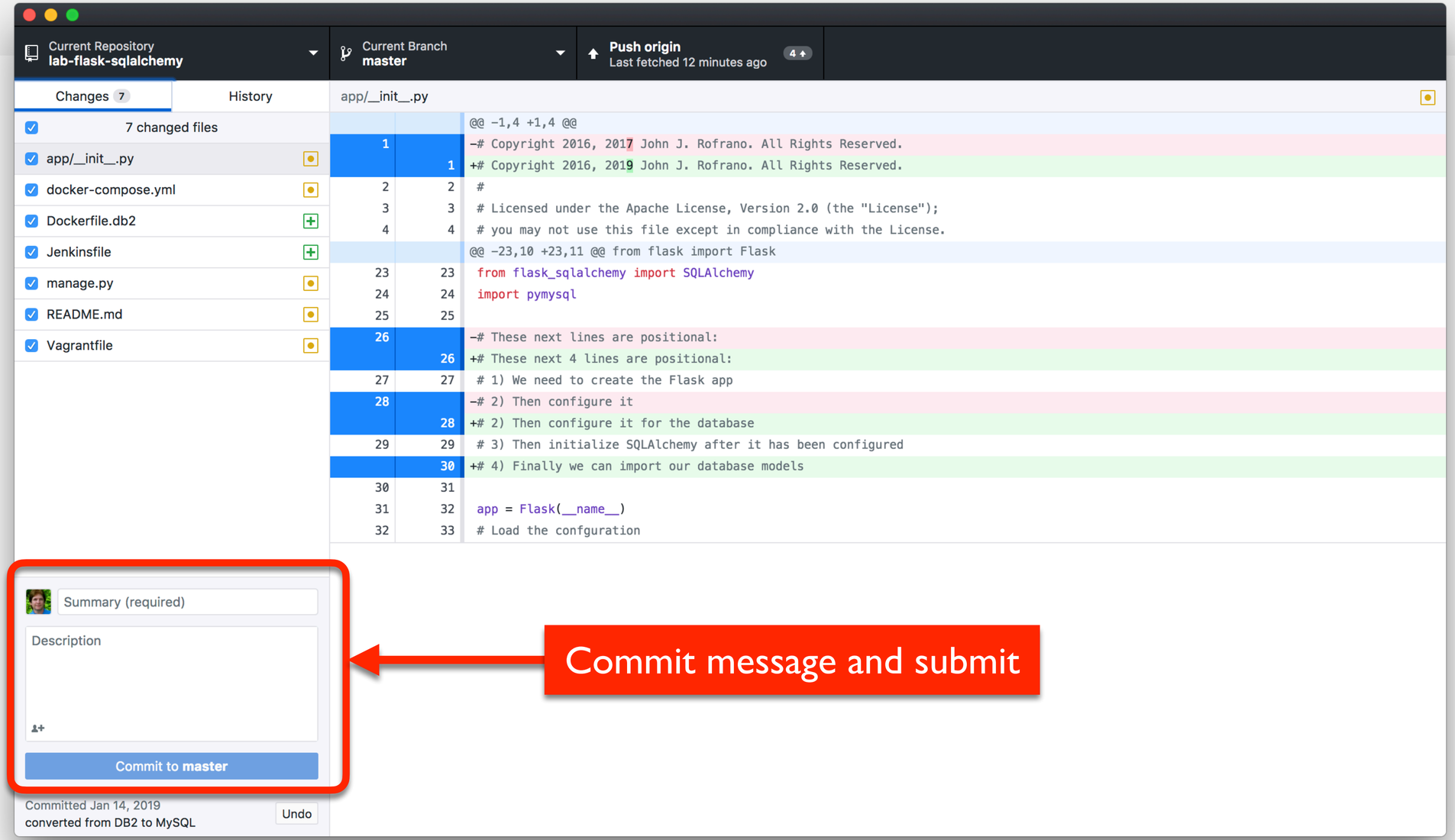
GitHub Desktop UI



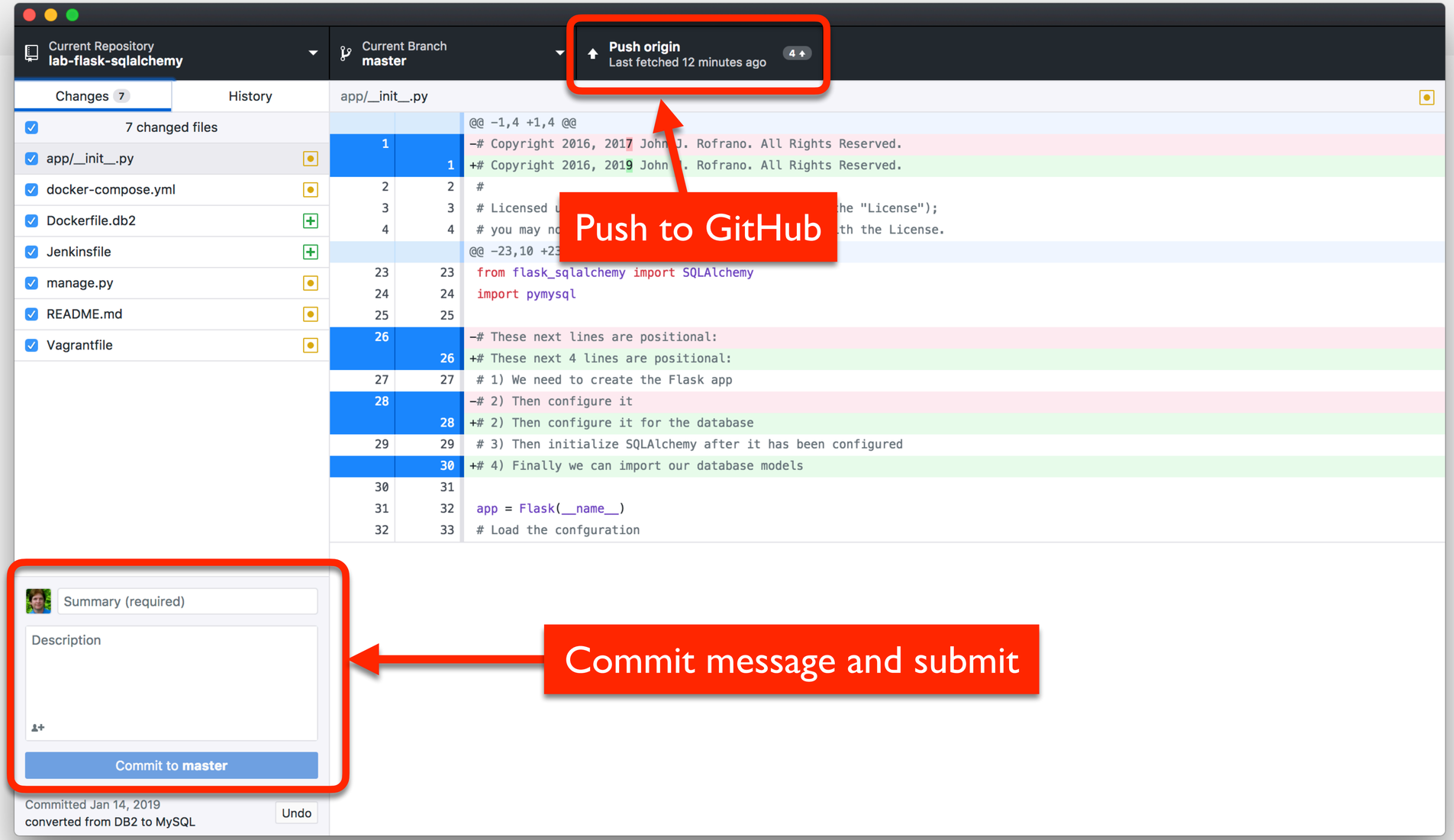
Change Repository



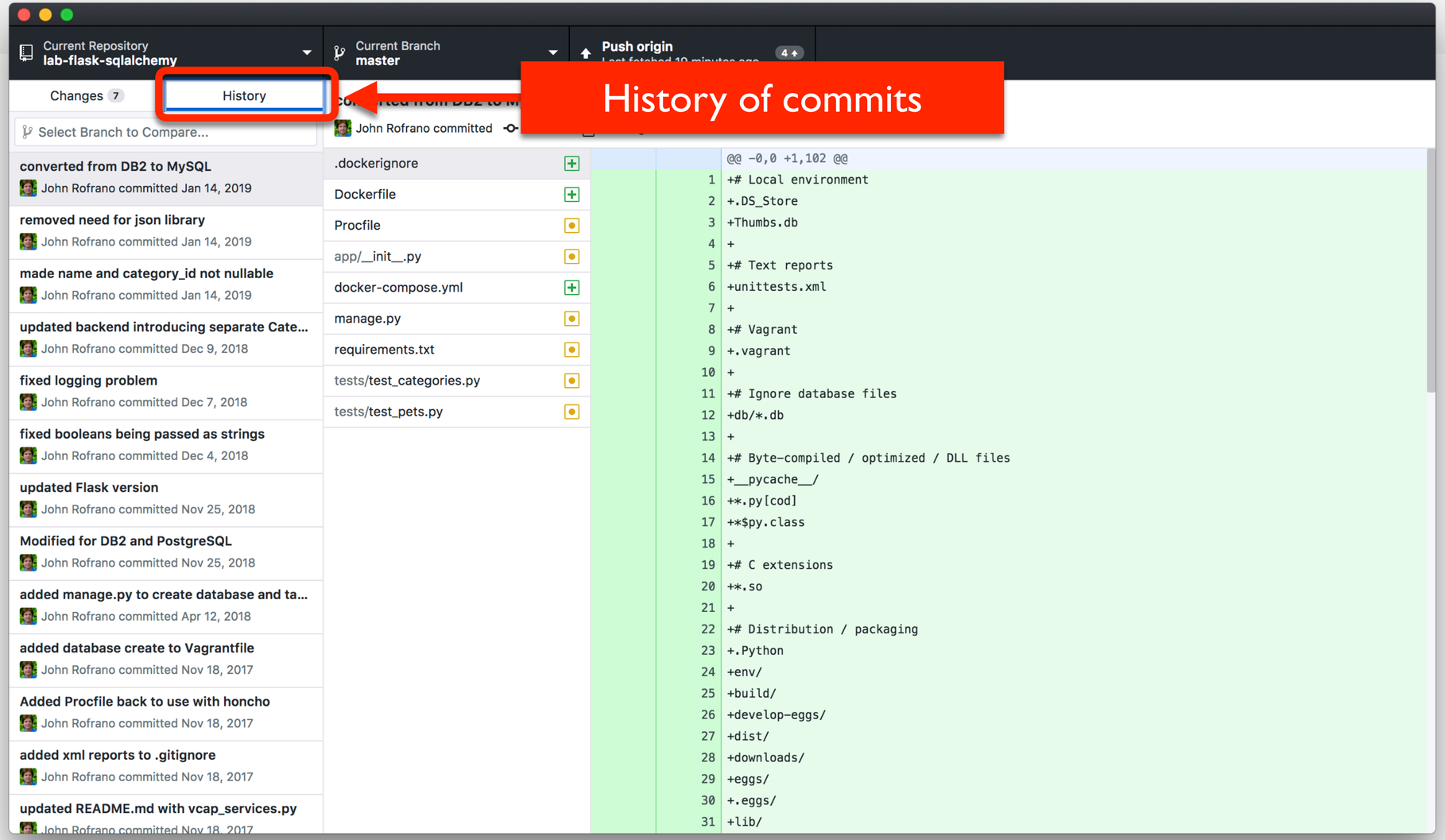
GitHub Desktop UI



GitHub Desktop UI



History of Commits



What Have We Covered?

The basic workflow for cloning, branching, pushing, and pull requests

How to merge, handle merge conflicts, and rebase

How to use status, logs, and reflog to see where you are at

Additional Reading

- GitHub Learning Resources
  - https://github.github.io/on-demand/
- Git documentation
  - https://git-scm.com/docs/
- Git Feature Branch Workflow:
  - https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow