22.1. What is Git?

22.1.1. Version Control Systems

A version control system (VCS) is a system for tracking changes to a code base and storing each version. Version control systems assist programmers with keeping backups and a history of the revisions made to the code base over time. With that history, programmers can roll back to a version without a particular bug. A VCS also enables collaboration between programmers as they can work on different versions of a code base and share their work.

Git is one VCS and is prevalent amongst programmers and corporations.

A VCS has a **repository** or storage container for the code base. Repositories include the files within the code base, the versions over time and a log of the changes made. When a programmer updates the repository, it means they are making a **commit**.

22.1.2. Getting Started with Git

In order to get started with Git, you need to install Git on your machine and install Visual Studio Code.

22.1.3. Check Your Understanding

Question

What is a benefit of using a VCS?

22.2. Respositories and Commits

22.2.1. Create a Repository

To get started with a git repository, the programmer must first create one. To create a git repository, the programmer navigates to their project directory and uses the command `git init`, like so:

```
Students-Computer:~ student$ mkdir homework
Students-Computer:~ student$ cd homework
Students-Computer:homework student$ git init
   Initialized empty Git repository in /Users/student/homework/.git/
```

Now the programmer is ready to code away!

22.2.2. Making Commits

After a while, the programmer has made a lot of changes and saved their code files many times over. So when do they make a commit to their repository?

The general rule of thumb is that any time a significant change in functionality is made, a commit should be made.

If the programmer has created the Git repository and is ready to commit, they can do so by following the commit process.

Note

Git does have a simple commit command, however, making a proper commit requires that the programmers follow a longer procedure than just one command.
The procedure for making a commit to a Git repository includes 4 stages.

1. `git status` gives the programmer information about files that have been changed.
2. `git add` allows the programmers to add specific or all changed files to a commit.
3. `git commit` creates the new commit with the files that the programmer added.
4. `git log` displays a log of every commit in the repository.

If the steps above are followed correctly, the programmer will find their latest commit at the top of the log.

Here is how the process will look in the terminal:

```
Students-Computer:homework student$ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        learning-git.js

nothing added to commit but untracked files present (use "git add" to track)
Students-Computer:homework student$ git add .
Students-Computer:homework student$ git commit -m "My first commit"
```

```
 [master (root-commit) 2c1e0af] My first commit
  1 file changed, 1 insertion(+)
  create mode 100644 learning-git.js
Students-Computer:homework student$ git log
commit 2c1e0af9467217d76c7e3c48bcf9389ceaa4714b
Author: Student <lc101.student@email.com>
Date:  Wed Apr 24 14:44:59 2019 -0500

    My first commit
```

To break down what happens in a commit even further:

When using `git status`, the output shows two categories: modified tracked files and modified untracked files. Modified tracked means that Git the file exists in the repository already, but is different than the version in the repository. Modified untracked means that it is a new file that is not currently in the repository.

`git add` adds files to the commit, but it does not commit those files. By using `git add .`, all the modified files were added to the commit. If a programmer only wants add one modified file, they can do so.

`git commit` actually commits the files that were added to the repository. By adding `-m "My first commit"`, a comment was added to the commit. This is helpful for looking through the log and seeing detailed comments of the changes made in each commit.

`git log` shows the author of the commit, the date made, the comment, and a 40-character hash. This hash or value that is a key for Git to refer to the version. Programmers rarely use these hashes unless they want to roll back to that version.

22.2.3. Check Your Understanding

Question

What git command is NOT a part of the commit process?

a. `git add`
b. `git log`
c. `git status`
d. `git push`

22.3. Remote Repositories

22.3.1. Local, Remote, Github, Oh My!

So far, the book has covered how to setup a Git repository on the local machine. But, one of the benefits of using a VCS was storage of backups. So, what happens to the code base if something happens to the machine? That is where remote repositories come in. Instead of keeping a Git repository only on a local machine, the code base is in a **remote repository** and the programmers working on it keep copies on their local machine.

To get started with remote repositories, create an account on Github. From there, programmers can create a remote repository, view commit history, and report issues with the code.

22.3.2. Collaborating with Colleagues

What if a programmer wants to start collaborating with their colleagues on a new project? They might need to start with the work that one of their colleagues has already done. In this particular case, the programmer has to import the work that is being stored in an online repository onto their local machine.

They can clone a remote repository by using the `git clone <url>` command. Github and other online Git systems give users the option to clone the repository through HTTPS or SSH depending on how their Github profile is set up. The `<url>` of the command is where the programmer adds the url to the repository that they are cloning.

Note

Throughout this book, HTTPS will be used for cloning repositories.

22.3.3. Contributing to a Remote Repository

Now that the programmer has a profile on Github and a local copy of a remote repository, they start coding!

Once they create a new feature, it is time to make a commit. When working with a remote, the commit process has five steps:

1. `git status`
2. `git add`
3. `git commit`
4. `git push origin master`
5. `git log`

The fourth step uses the new command `git push` where the commit is pushed to the remote from the local. `origin` indicates that the commit does indeed go to the remote and `master` is the name of the branch that the commit goes to.

22.3.4. Check Your Understanding

Question

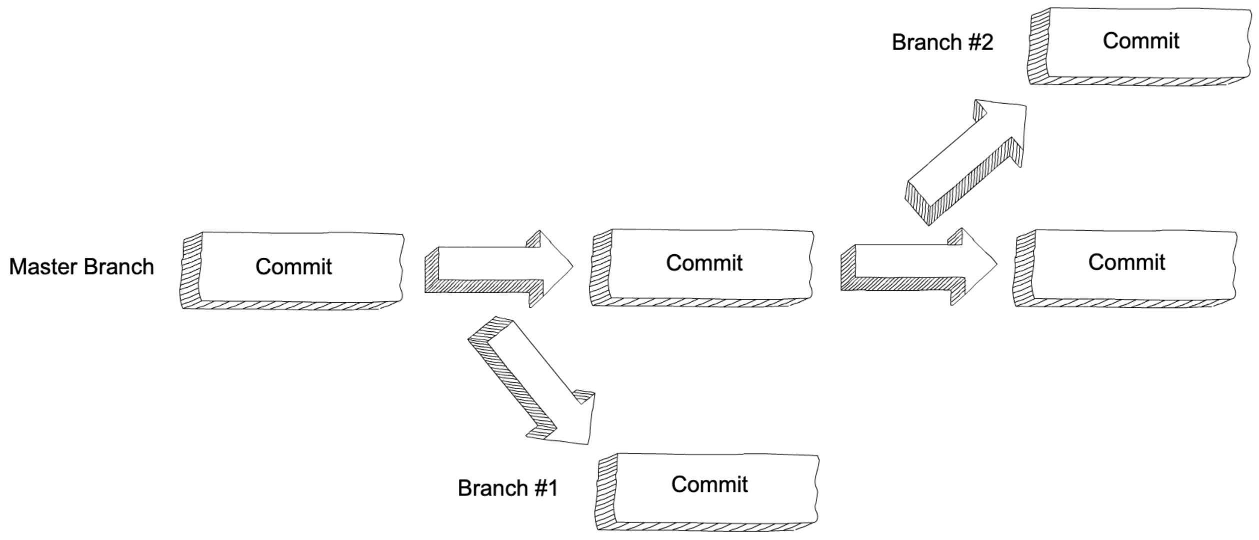
What is the new command for making a commit to a remote repository?

22.4. Branches

22.4.1. Branching in Git

So far this book has talked about Git's ability to store different versions of a code base. What if two programmers want to work on different features of the code base at the same time? They may want to start with the same version and then one programmer wants to change the HTML and the other the CSS. It would not be effective for the two programmers to commit their changes to the repository at the same time. Instead, Git has branches. A **branch** is a separate version of the same code base. Like a branch on a tree, a branch in Git shares the same trunk as other branches, but is an individual. With branches, the two programmers could work on separate versions of the same website without interfering with each other's work. Besides collaboration, programmers use branches for storing and testing new features of software called feature branches.

In the previous section, when checking the status, the top line was `On branch master`. The master branch is the default branch of the repository. Many programmers keep the live version of their code in the master branch. For that reason, major work should be done in a new branch, so it doesn't impact the live software.



22.4.2. Creating a New Branch

A programmer is on master and they want to start building a new feature in a new branch. Their first step would be to create a new branch for their work.

To create a branch, the command is `git checkout -b <branch name>`. By using this command, not only is a new branch created, but also the programmer switches to their new branch.

22.4.3. Switching to an Existing Branch

If the branch already exists, the programmer may want to switch to that branch. To do so, the command is `git checkout <branch name>`.

22.4.4. Check Your Understanding

Question

What is a reason for creating a branch in Git?

22.5. Merging in Git

22.5.1. How to Merge

A **merge** in Git is when the code in two branches are combined in the repository. The command to merge a branch called `test` into `master` is `git merge test`. Before running the merge command, the programmer should make sure they are on the branch they want to merge into!

22.5.2. Merge Conflicts

This process is often seamless. In the example in the previous section, a programmer created a branch to change the HTML and the other programmer did the same to change the CSS. Because the two programmers changed different files, the merge of the updated HTML and updated CSS won't create a conflict. A **merge conflict** is when a change was made to the same line of code on both branches. Git doesn't know which change to accept, so it is up to the programmers to resolve it. Merge conflicts are minor on small applications, but can cause issues with large enterprise applications. Even though the thought of ruining software can be scary, every programmer deals with a merge conflict during their career. The best way to deal with a merge conflict is to face it head on and rely on teammates for support!

22.5.2.1. Ways to Avoid Merge Conflicts

Even though merge conflicts are normal in Git, it is also normal for programmers to want to do everything they can to avoid one. Here are some tips on how to avoid a merge conflict:

1. Git has a dry-run option for many commands. When a programmer uses that option, Git outputs what WILL happen, but doesn't DO it. With merging in Git, the command to run a dry-run and make sure there aren't any conflicts is `git merge --no-commit --no-ff <branch>`. `--no-commit` and `--no-ff` tell Git to run the merge without committing the result to the repository.
2. Before merging in a branch, any uncommitted work that would cause a conflict needs to be dealt with. A programmer can opt to not commit that work and instead **stash** it. By using the `git stash` command, the uncommitted work is saved in the stash and the repository is returned to the state at the last commit.

22.5.3. Check Your Understanding

Question

If a programmer is on the branch `test` and wants to merge a branch called `feature` into `master`, what steps should they take?

22.6. Exercises: Git

22.6.1. Working in a Local Repository

We will use our new terminal powers to move through the Git exercises.

1. In whichever directory you are keeping your coursework, make a new directory called `Git-Exercises` using the `mkdir` command.
2. Inside the `Git-Exercises` directory, initialize a new Git respository using `git init`.
3. Add a file called `exercises.txt` using the `touch` command in the terminal.
4. Commit your local changes using the `git commit` procedures.
5. Add “Hello World!” to the file called `exercises.txt`.
6. Commit your local changes following the same steps to you used for step 4.
7. View and screenshot the result when you use `git log`. Make note of what you see!

22.6.2. Setting up a Github Account

For our remote repositories, we will be using Github.

To create your account, follow these steps:

1. Navigate to Github’s site using the link above.
2. Sign up for an account on the homepage either by filling out the form or clicking the “Sign Up” button.
3. Once you have an account, you are ready to store your remote work.

22.7. Studio: Communication Log

22.7.1. Getting Ready: Code Together

Coding together allows you to work as a team so you can build bigger projects faster.

In this studio, we will practice the common Git commands used when multiple people work on the same code base.

You and a partner will begin by coding in tag-team shifts. By the end of the task you should have a good idea about how to have two people work on the same code at the same time. You will learn how to:

1. Quickly add code in pull + push cycles *(Important! This is the fundamental process!)*
2. Add a collaborator to a GitHub Project
3. Share *repositories* on GitHub
4. Create a *branch* in Git
5. Create a *pull request* in GitHub
6. Resolve merge conflicts (which are not as scary as they sound)

This lesson reinforces:

1. Creating repositories
2. Cloning repositories
3. Working with Git concepts: Staging, Commits, and Status

22.7.2. Overview

The instructor will discuss why GitHub is worth learning. You already know how to use a local Git repository with one branch, giving you the ability to move your code forward and backward in time. Working with branches on GitHub extends this ability by allowing multiple people to build different features at the same time, then combine their work. Pull requests act as checkpoints when code flows from branch to branch.

Students *must* pair off for this exercise. If you have trouble finding a partner, ask your TA for help.

22.7.3. Studio

We are going to simulate a radio conversation between the shuttle pilot and mission control.

First, find a new friend to share the activity.

You and your partner will alternate tasks, so designate one of you as **Pilot** and the other as **Control**. Even when it is not your turn to complete a task, read and observe what your partner is doing to complete theirs. The steps here mimic how a real-world collaborative Git workflow can be used within a project.

Warning

As you go through these steps, you'll be working with branches. It's very likely you will make changes to the code only to realize that you did so in the wrong branch. When this happens (and it happens to all of us) you can use `Git stash` to cleanly move your changes to another branch. Read about how to do so in our Git Stash tutorial.

22.7.3.1. Step 1: Create a New Repository

Control: Navigate to your development folder. Follow these instructions to create a new project.

```
$ mkdir communication-log
$ cd communication-log
$ git init
```

In that directory, open a new file *index.html* in the editor of your choice. Paste in this code:

```
<html>
<body>
  <p>Radio check. Pilot, please confirm.</p>
</body>
</html>
```

Let's check that our html looks okay by opening it in a browser. Do this by selecting *File > Open File* in your web browser, and navigating to the location of your new HTML file. The URL will look something like this: `file:///Users/cheryl/Development/communication-log/index.html`.

Once you've seen this file in the browser, let's stage and commit it.

```
$ git status
On branch master

Initial commit

Untracked files:
(use "Git add <file>..." to include in what will be committed)

   index.html

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

The file is not staged. Let's add everything in this directory.

```
$ git add .
$ git status
On branch master

Initial commit

Changes to be committed:
(use "git rm --cached <file>..." to unstage)

   new file:   index.html
```

We see that the file is staged. Let's commmit.

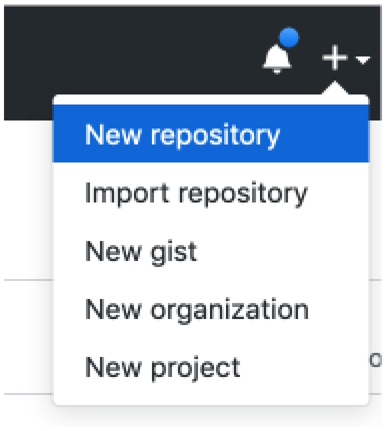
```
$ git commit -m 'Started communication log.'
[master (root-commit) e1c1719] Started communication log.
1 file changed, 5 insertions(+)
create mode 100644 index.html
$ git log
commit 679de772612099c77891d2a3fab12af8db08b651
Author: Cheryl <chrisbay@gmail.com>
Date:   Wed Apr 5 10:55:56 2017 -0500

   Started communication log.
```

Great! We’ve got our project going locally, but we’re going to need to make it accessible for **Pilot** also. Let’s push this project up to GitHub.

22.7.3.2. Step 2: Share Your Repository On GitHub

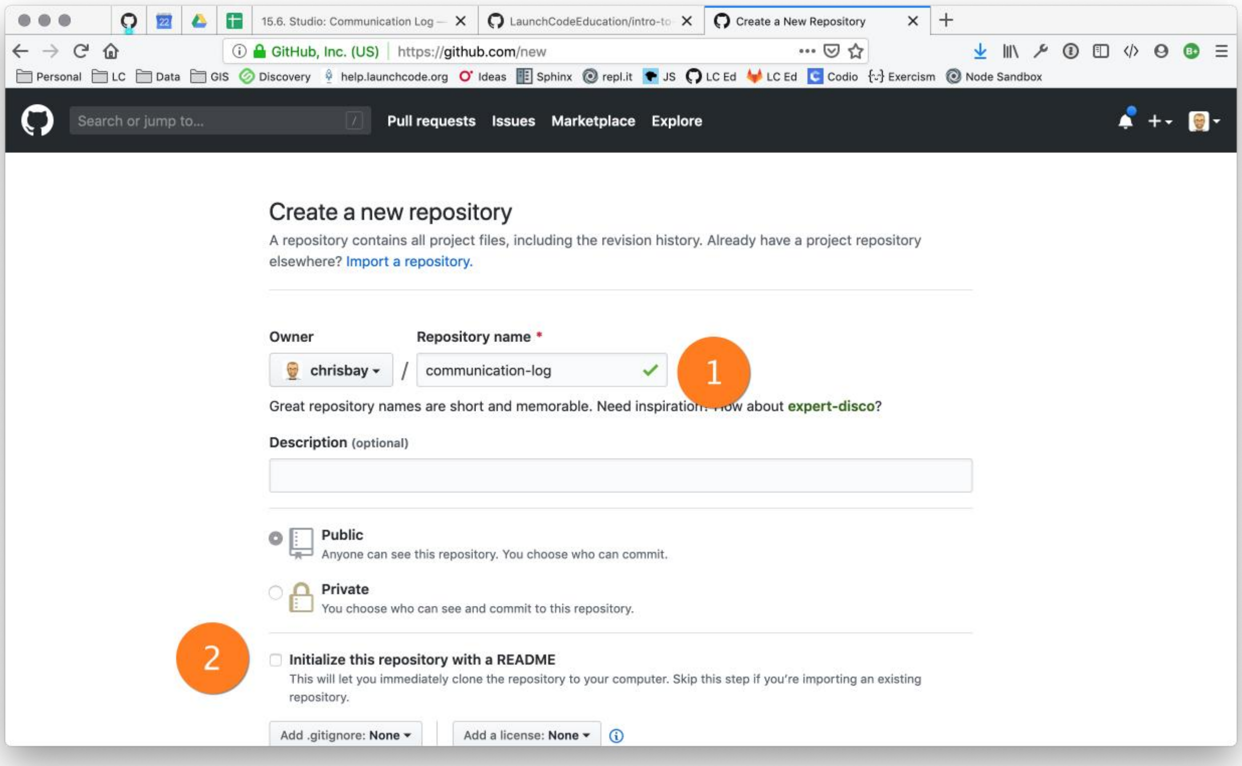
Control: Go to your GitHub profile in a web browser. Click on the “+” button to add a new repository (‘repo’).



The New Repository link is in the dropdown menu at top right on GitHub.

To create a new repository:

1. Fill in the name and description.
2. Uncheck *Initialize this repository with a README* and click *Create Repository*.



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner: chrisbay / Repository name *: communication-log

Great repository names are short and memorable. Need inspiration? How about expert-disco?

Description (optional)

Public
Anyone can see this repository. You choose who can commit.

Private
You choose who can see and commit to this repository.

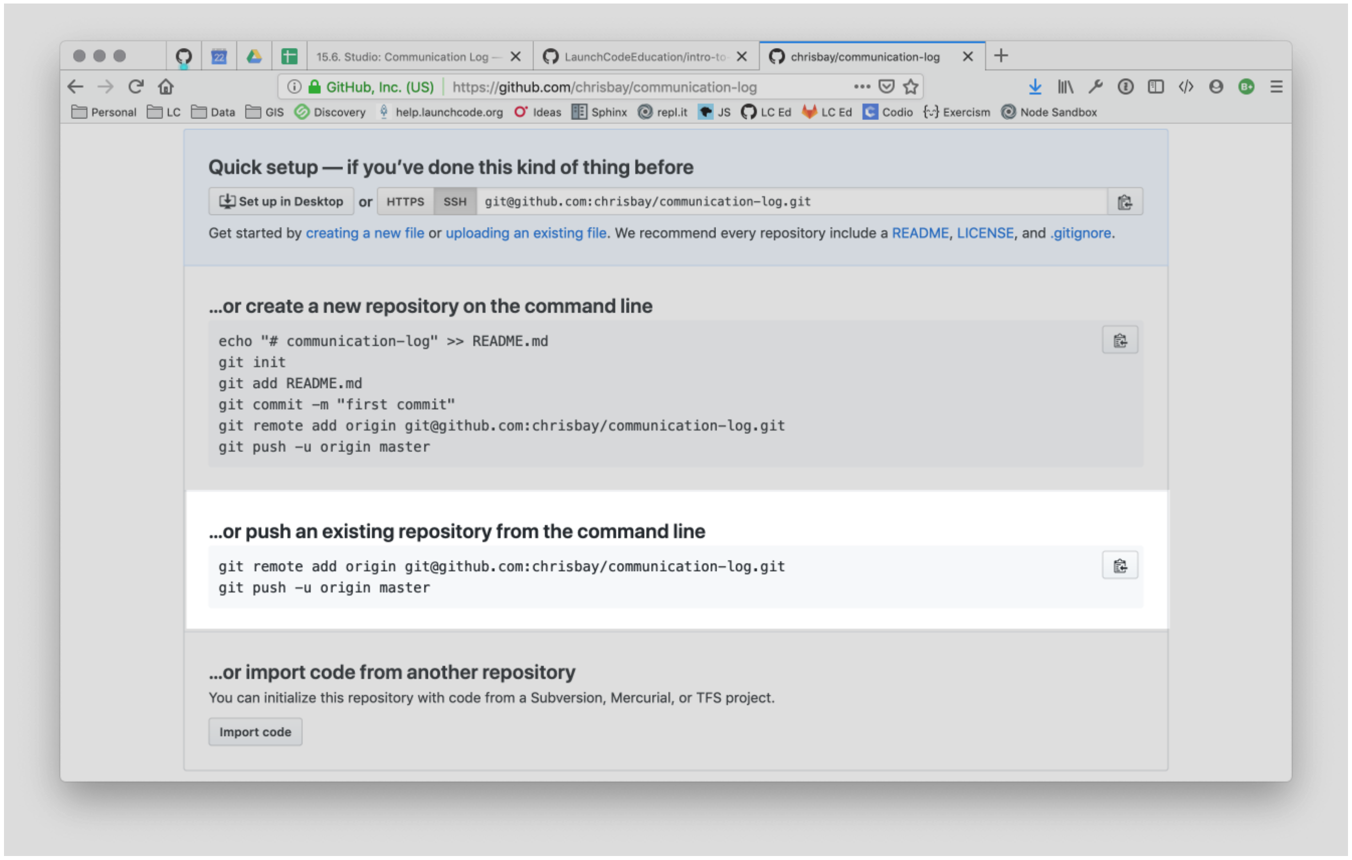
Initialize this repository with a README
This will let you immediately clone the repository to your computer. Skip this step if you’re importing an existing repository.

Add .gitignore: None | Add a license: None

Create a new repository in GitHub

Note

If you initialize with a README, in the next step Git will refuse to merge this repo with the local repo. There are ways around that, but it's faster and easier to just create an empty repo here.
After clicking, you should see something similar to:


15.6. Studio: Communication Log
LaunchCodeEducation/intro-to
chrisbay/communication-log
GitHub, Inc. (US)
https://github.com/chrisbay/communication-log
Personal
LC
Data
GIS
Discovery
help.launchcode.org
Ideas
Sphinx
repl.it
JS
LC Ed
LC Ed
Codio
Exercism
Node Sandbox

Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH git@github.com:chrisbay/communication-log.git

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

…or create a new repository on the command line

```
echo "# communication-log" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin git@github.com:chrisbay/communication-log.git
git push -u origin master
```

…or push an existing repository from the command line

```
git remote add origin git@github.com:chrisbay/communication-log.git
git push -u origin master
```

…or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Import code

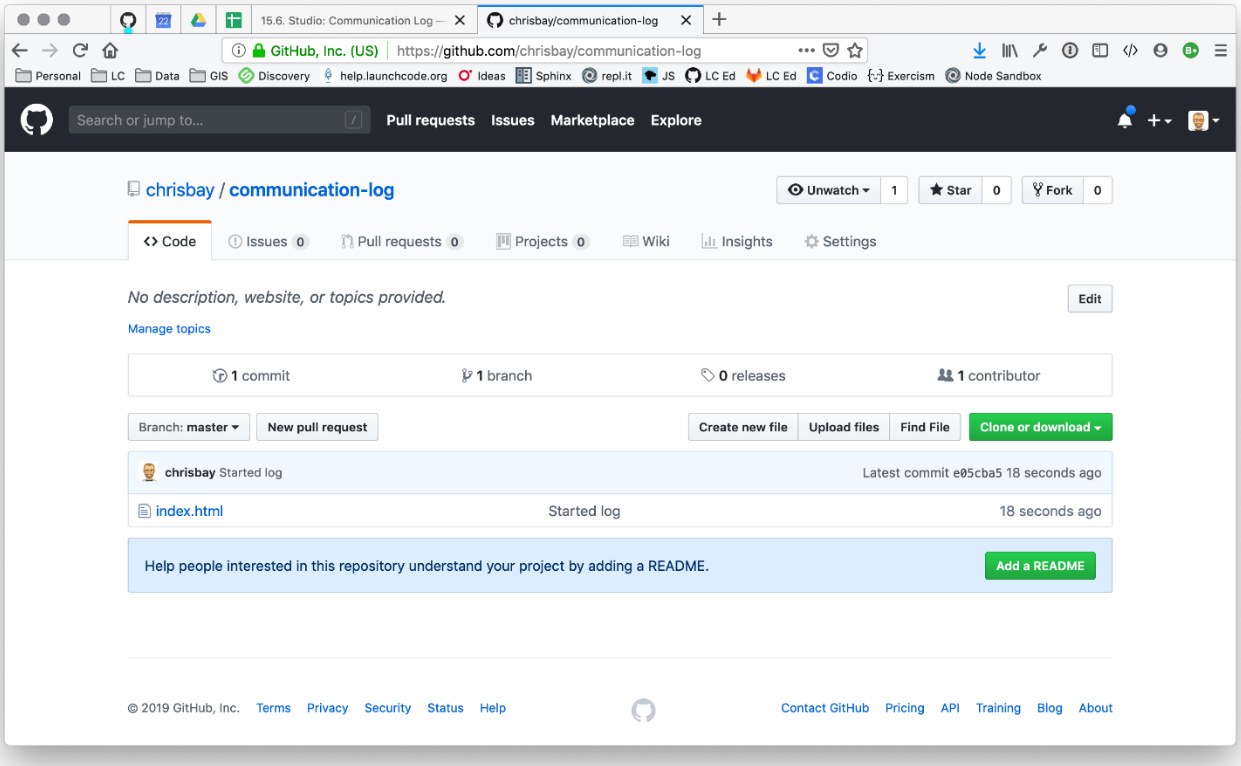
Connecting to a repository in GitHub

Now go back to your terminal and cut and paste the commands shown in the instructions on GitHub. These should be very similar to:

```
$ git remote add origin https://github.com:chrisbay/communication-log.git
$ git push origin master
```

Warning

Unless you've set up an SSH key with GitHub, make sure you've selected the HTTPS clone URL. If you're not sure whether you have an SSH key, you probably don't.
Now you should be able to confirm that GitHub has the same version as your local project. (File contents in browser match those in terminal). Click around and see what is there. You can read all your code through GitHub's web interface.



https://github.com/chrisbay/communication-log

No description, website, or topics provided.

chrisbay Started log — Latest commit e05cba5 18 seconds ago

index.html — Started log — 18 seconds ago

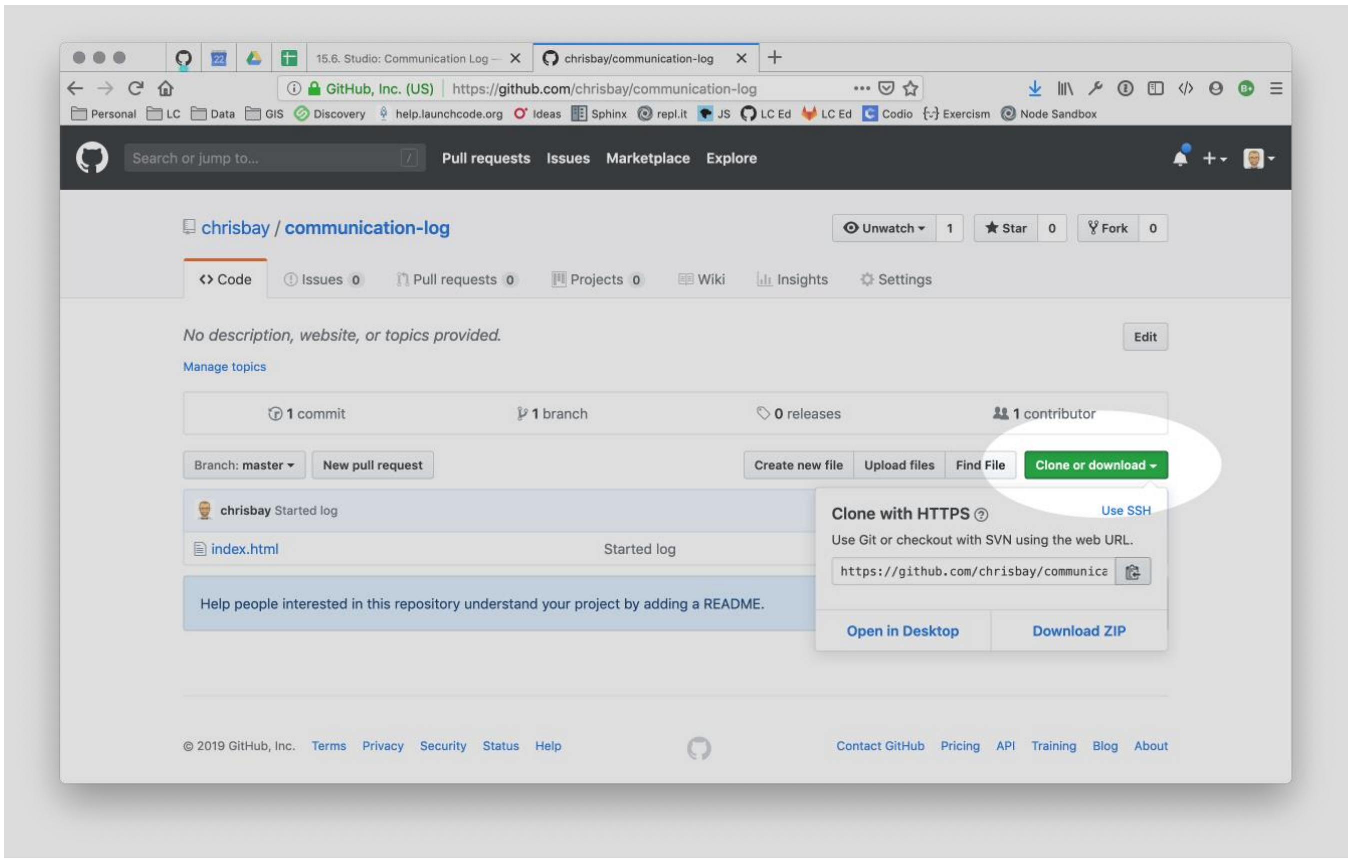
Help people interested in this repository understand your project by adding a README.

© 2019 GitHub, Inc.

A repository with one commit in GitHub

22.7.3.3. Step 3: Clone a Project from GitHub

Pilot: Go to Control’s GitHub profile and find the communication-log repo. Click on the green *Clone or download* button. Use HTTPS (not SSH). Copy the url to your clipboard.



Cloning a repository in GitHub

In your terminal, navigate to your development folder and clone down the repo. The command should look something like this.

```
$ git clone https://github.com/chrisbay/communication-log.git
```

Now you can respond to Control! Open the *index.html* file in your editor and add your response to mission control. Be creative—the communication can go anywhere! Just don't ask your partner what you should write. After you finish, commit your change.

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working directory clean
$ git add index.html
$ git commit -m 'Added second line to log.'
```

Now we need to push up your changes so Control can use them as well.

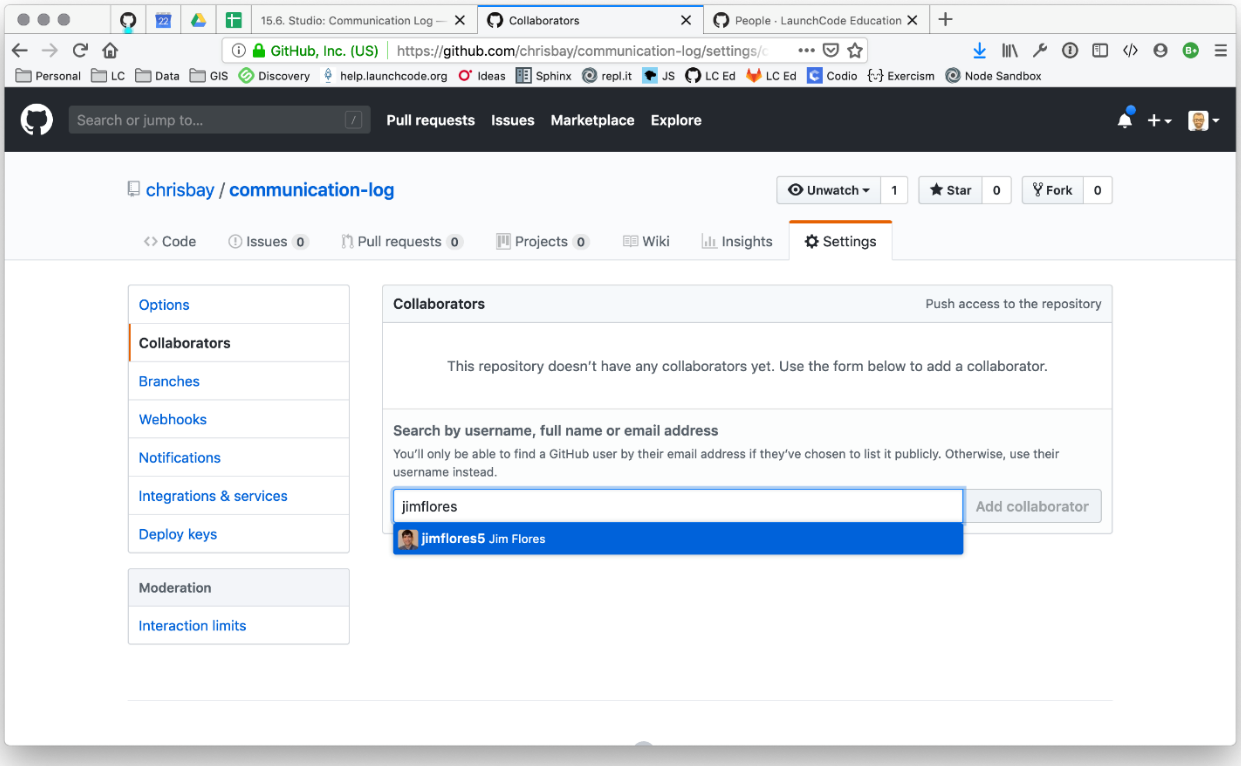
```
$ git push origin master
ERROR: Permission to chrisbay/communication-log.git denied to pilot.
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.
```

Great error message! It let us know exactly what went wrong: Pilot does not have security permissions to write to Control's repo. Let's fix that.

22.7.3.4. Step 4: Add A Collaborator To A GitHub Project

Control: In your web browser, go to your *communication-log* repo. Click the *Settings* button then click on *Collaborators*. Enter in Pilot's GitHub username and click *Add Collaborator*.



Collaborators

Push access to the repository

This repository doesn't have any collaborators yet. Use the form below to add a collaborator.

Search by username, full name or email address

You'll only be able to find a GitHub user by their email address if they've chosen to list it publicly. Otherwise, use their username instead.

jimflores

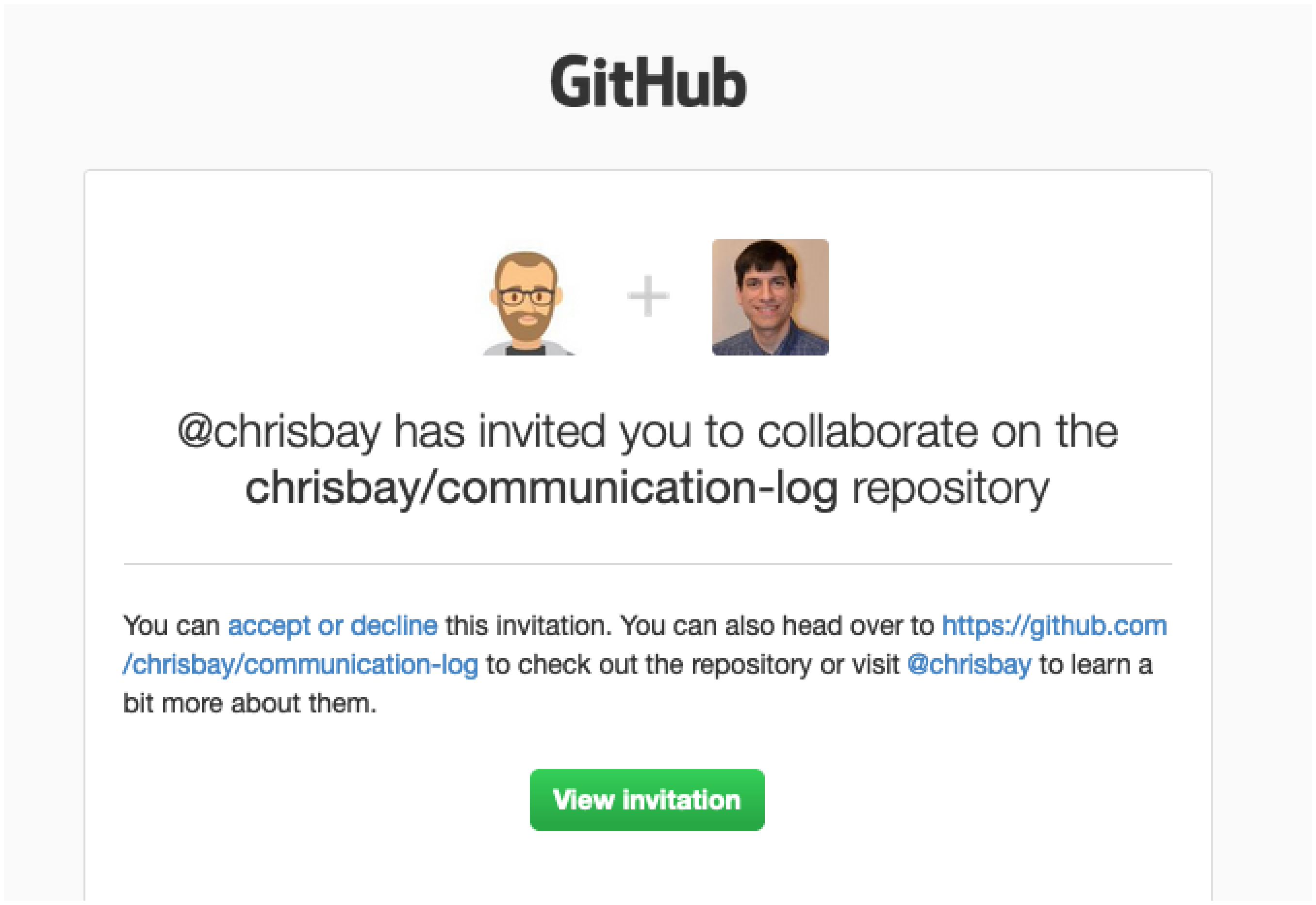
Add a collaborator to your repo in GitHub

22.7.3.5. Step 5: Join the Project and Push

Piolt: You should receive an email invitation to join this repository. View and accept the invitation.

Note

If you don't see an email (it may take a few minutes to arrive in your inbox), check your Spam folder. If you still don't have an email, visit the repository page for the repo that Control created (ask them for the link), and you'll see a notification at the top of the page.


GitHub

@chrisbay has invited you to collaborate on the chrisbay/communication-log repository

You can accept or decline this invitation. You can also head over to https://github.com/chrisbay/communication-log to check out the repository or visit @chrisbay to learn a bit more about them.

Invited to collaborate email in GitHub

Now let’s go enter that command again to push up our code.

```
$ git push origin master
Counting objects: 9, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (9/9), done.
Writing objects: 100% (9/9), 1.01 KiB | 0 bytes/s, done.
Total 9 (delta 8), reused 0 (delta 0)
remote: Resolving deltas: 100% (8/8), completed with 8 local objects.
To git@github.com:chrisbay/communication-log.git
   511239a..679de77  master -> master
```

Anyone reading the HTML through GitHub’s browser interface should now see the new second line.

22.7.3.6. Step 6: Pull Pilot’s Line and Add Another Line

Control: You might notice you don’t have the second line of code in your copy of the project on your computer. Let’s fix that. Go to the terminal and enter this command to pull down the updated code into your local git repository.

```
$ git pull origin master
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 3 (delta 1), pack-reused 0
Unpacking objects: 100% (3/3), done.
From github.com:chrisbay/communication-log
   e0de62d..e851b7e  master     -> origin/master
Updating e0de62d..e851b7e
```

```
Fast-forward
index.html | 1 +
1 file changed, 1 insertion(+)
```

Now, in your editor, add a third line to the communication. Then add, commit, and push it up.

You can have your story go anywhere! Try to tie it in with what the pilot wrote, without discussing with them any plans on where the story will go.

22.7.3.7. Step 7: Do It Again: Pull, Change, and Push!

Pilot: You might notice now *you* don't have the third line on your computer. Go to the terminal and enter this command to pull in the changes that Control just made.

```
$ git pull origin master
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 3 (delta 1), pack-reused 0
Unpacking objects: 100% (3/3), done.
From github.com:chrisbay/communication-log
   e851b7e..167684c  master     -> origin/master
Updating e851b7e..167684c
Fast-forward
index.html | 1 +
1 file changed, 1 insertion(+)
```

Now add a fourth line to the log. Again, be creative, but no planning!

Then add, commit, and push your change.

You can both play like this for a while! Feel free to repeat this cycle a few times to add to the story.

22.7.3.8. Step 8: Create a Branch In Git

This workflow is a common one in team development situations. You might wonder, however, if professional developers sit around waiting for their teammates to commit and push a change before embarking on additional work on their own. That would be a drag, and thankfully, there is a nice addition to this workflow that will allow for simultaneous work to be carried out in a reasonable way.

Pilot: While Control is working on an addition to the story, let's make another change simultaneously. In order to do that, we'll create a new branch. Recall that a branch is a separate "copy" of the codebase that you can commit to without affecting code in the `master` branch.

```
$ git checkout -b open-mic
Switched to a new branch 'open-mic'
```

This command creates a new branch named `open-mic`, and switches your local repository to use that branch.

Create a new file named `style.css` and add the following rules:

```css
body {
   color: white;
   background-color: black;
}
```

Then link it in `index.html`. It should look something like this:

```html
<html>
   <head>
      <link rel="stylesheet" type="text/css" href="style.css">
```

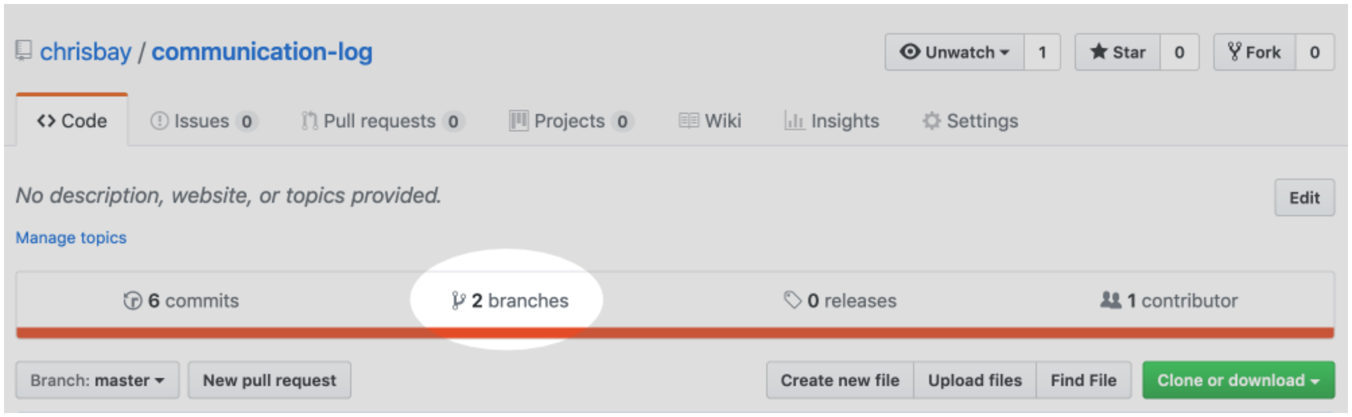
```
    </head>
    <body>
        <p>Radio check. Pilot, please confirm.</p>
        ... your content here
    </body>
</html>
```

Now stage and commit these changes.

```
$ git add .
$ git commit -m 'Added style.css'
$ git push origin open-mic
```

Note that the last command is a bit different than what we’ve used before (*git push origin master*). The final piece of this command is the name of the branch that we want to push to GitHub.

You and your partner should both now see a second branch present on the GitHub project page. To view branches on GitHub, select *Branches* from the navigation section just below the repository title.



No description, website, or topics provided.

Branches Button in GitHub

In your terminal, you can type this command to see a list of the available branches:

```
$ git branch
* open-mic
master
```

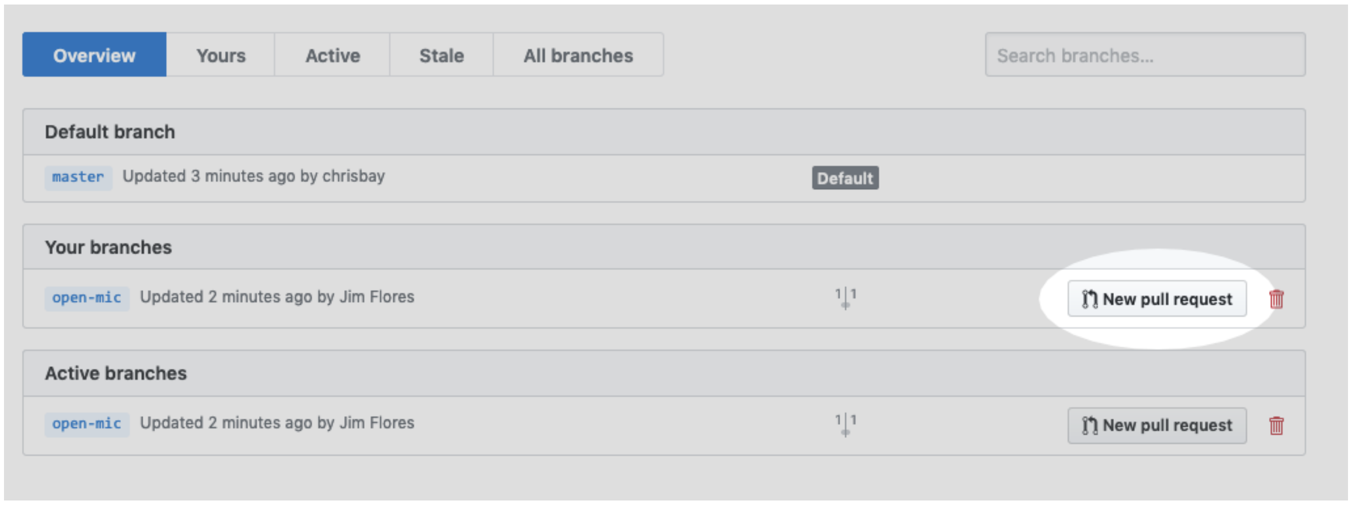
Note that creating and being able to see a branch in your local repository via this command does NOT mean that the branch is on GitHub. You’ll need to push the branch for it to appear on GitHub.

Note

The * to the left of `open-mic` indicates that this is the active branch.
Great! Now let’s show the other player your work in GitHub and ask them to merge it in to the main branch.

22.7.3.9. Create a Pull Request In GitHub

Pilot: If you haven’t already, in your browser, go to the GitHub project and click on *Branches* and make sure you see the new branch name, *open-mic*.



Default branch

master Updated 3 minutes ago by chrisbay Default

Your branches

open-mic Updated 2 minutes ago by Jim Flores 1 | 1 New pull request

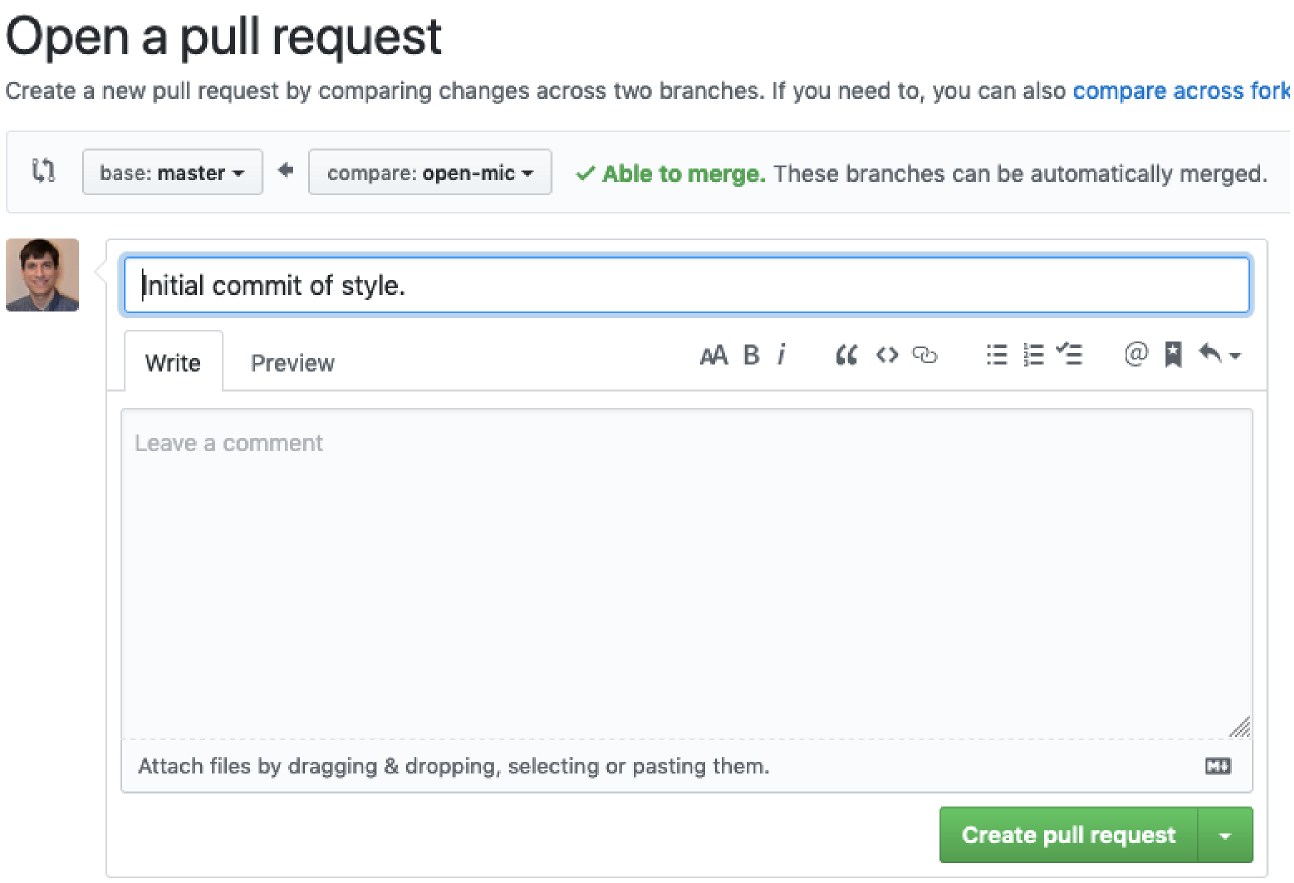
Active branches

open-mic Updated 2 minutes ago by Jim Flores 1 | 1 New pull request

Branches Page in GitHub

Click *New Pull Request* to begin the process of requesting that your changes in the `open-mic` branch be incorporated into the `master` branch. Add some text in the description box to let Control know what you did and why.

Note that the branch selected in the *base* dropdown is the one you want to merge *into*, while the selected branch in the *compare* dropdown is the one you want to merge *from*.



Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also compare across fork

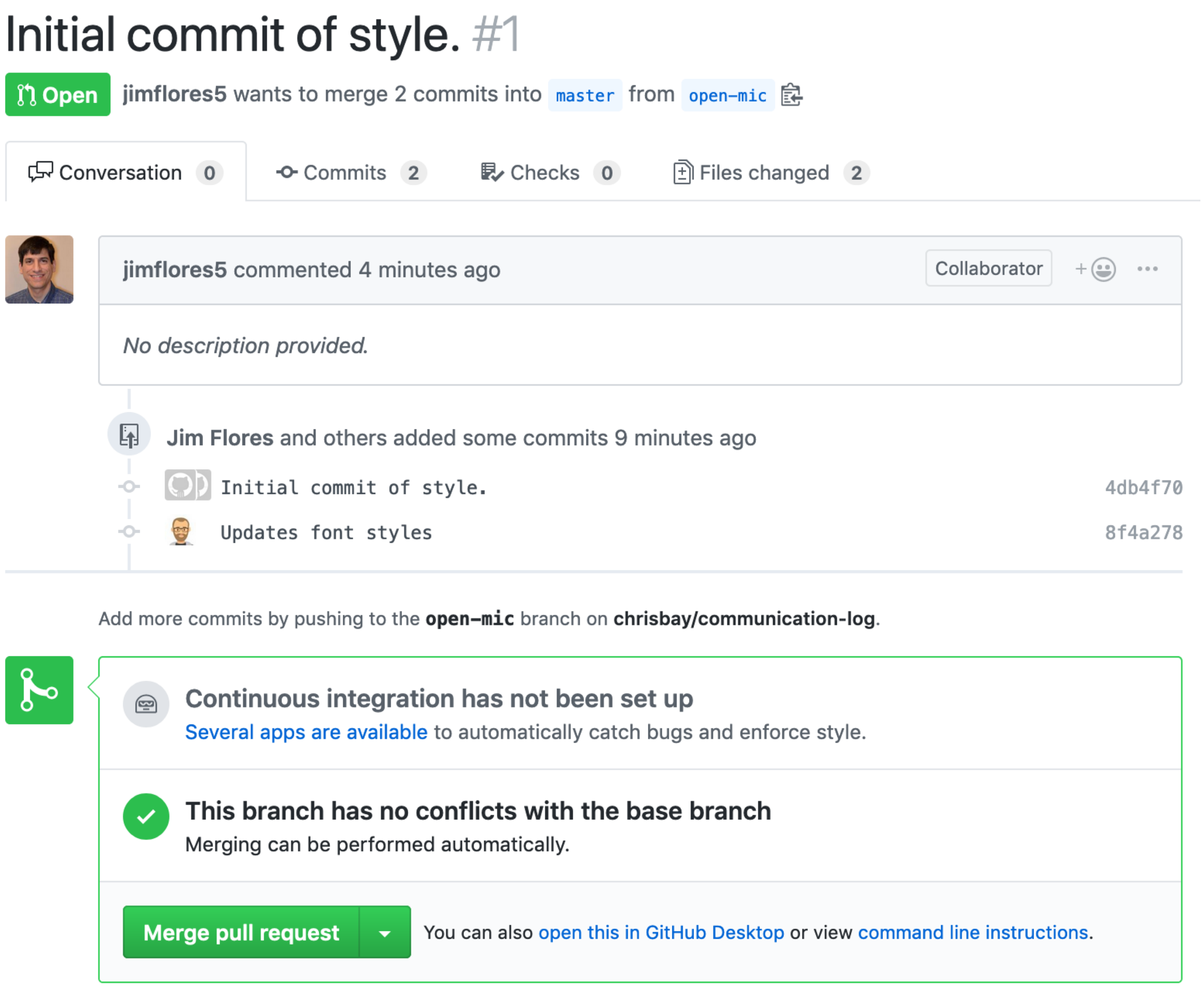
base: master ← compare: open-mic ✓ Able to merge. These branches can be automatically merged.

Initial commit of style.

Write
Preview
Leave a comment
Attach files by dragging & dropping, selecting or pasting them.
Create pull request

Open a PR in GitHub

This is what an opened pull request looks like:


Initial commit of style. #1

Open jimflores5 wants to merge 2 commits into `master` from `open-mic`

Conversation 0 · Commits 2 · Checks 0 · Files changed 2

jimflores5 commented 4 minutes ago · Collaborator

No description provided.

Jim Flores and others added some commits 9 minutes ago

```
Initial commit of style.    4db4f70
Updates font styles         8f4a278
```

Add more commits by pushing to the **open-mic** branch on **chrisbay/communication-log**.

Continuous integration has not been set up
Several apps are available to automatically catch bugs and enforce style.

This branch has no conflicts with the base branch
Merging can be performed automatically.

Merge pull request · You can also open this in GitHub Desktop or view command line instructions.

An open PR in GitHub

22.7.3.10. Step 10: Make a Change in the New Branch

Control: You will notice that you do not see the new `style.css` file locally. Type this command to see what branches are on your local computer:

```
$ git branch
* master
```

If you want to work with the branch before merging it in, you can do so by typing these commands:

```
$ git fetch origin open-mic
...
$ git branch
open-mic
* master
$ git checkout open-mic
Switched to branch 'open-mic'
Your branch is up-to-date with 'origin/open-mic'.
```

Make a change, commit, and push this branch–you will see that the pull request in GitHub is updated to reflect the changes you added. The context in the description box is NOT updated, however, so be sure to add comments to the pull request to explain what you did and why.

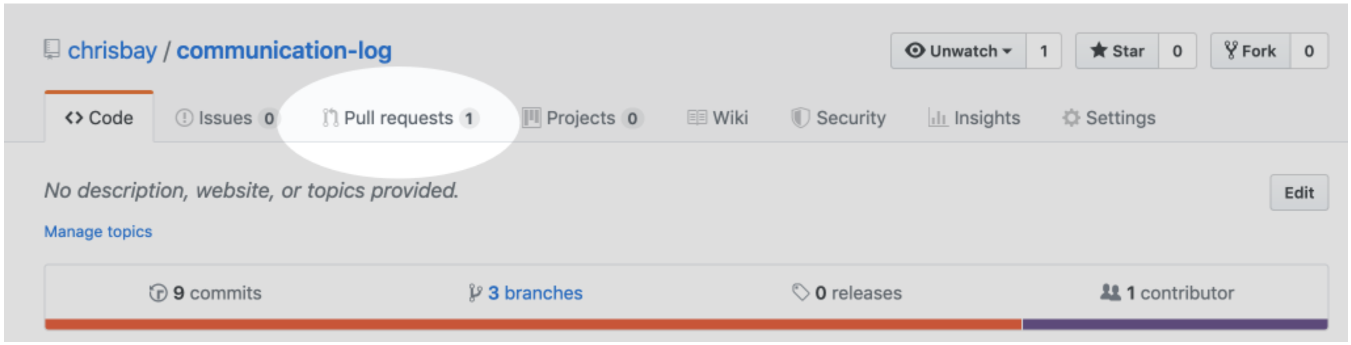
Now switch back to the `master` branch:

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

You will see your files no longer have the changes made in the `open-mic` branch. Let’s go merge those changes in, so that the `` `master` `` branch adopts all the changes in the `open-mic` branch.

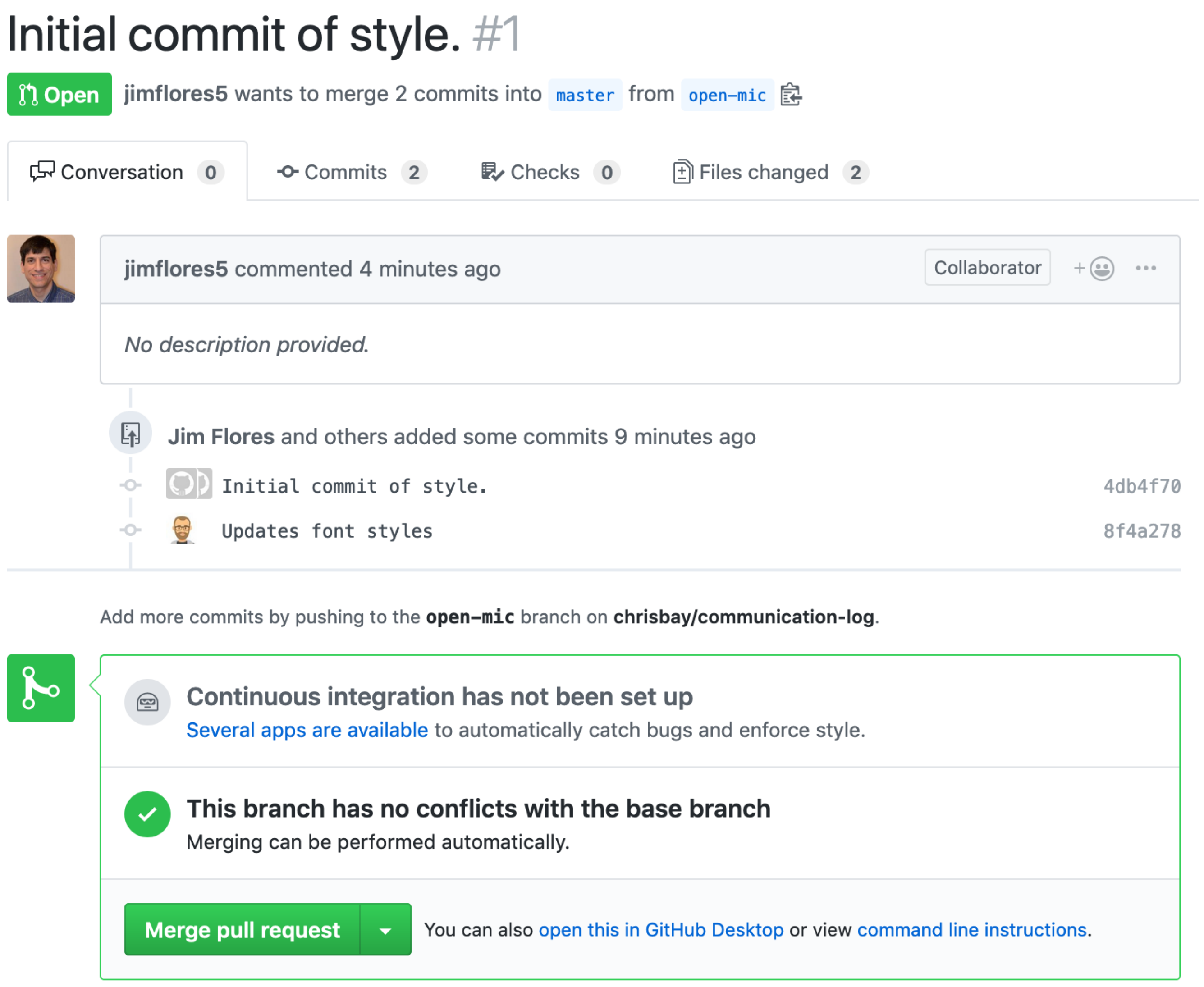
22.7.3.11. Step 11: Merge the Pull Request

Control: Go to the repo in GitHub. Click on *Pull Requests*.



PR Open in GitHub

Explore this page to see all the information GitHub shows you about the pull request.


Initial commit of style. #1

Open jimflores5 wants to merge 2 commits into master from open-mic

Conversation 0 · Commits 2 · Checks 0 · Files changed 2

jimflores5 commented 4 minutes ago — Collaborator

No description provided.

Jim Flores and others added some commits 9 minutes ago

```
Initial commit of style.    4db4f70
Updates font styles         8f4a278
```

Add more commits by pushing to the **open-mic** branch on **chrisbay/communication-log**.

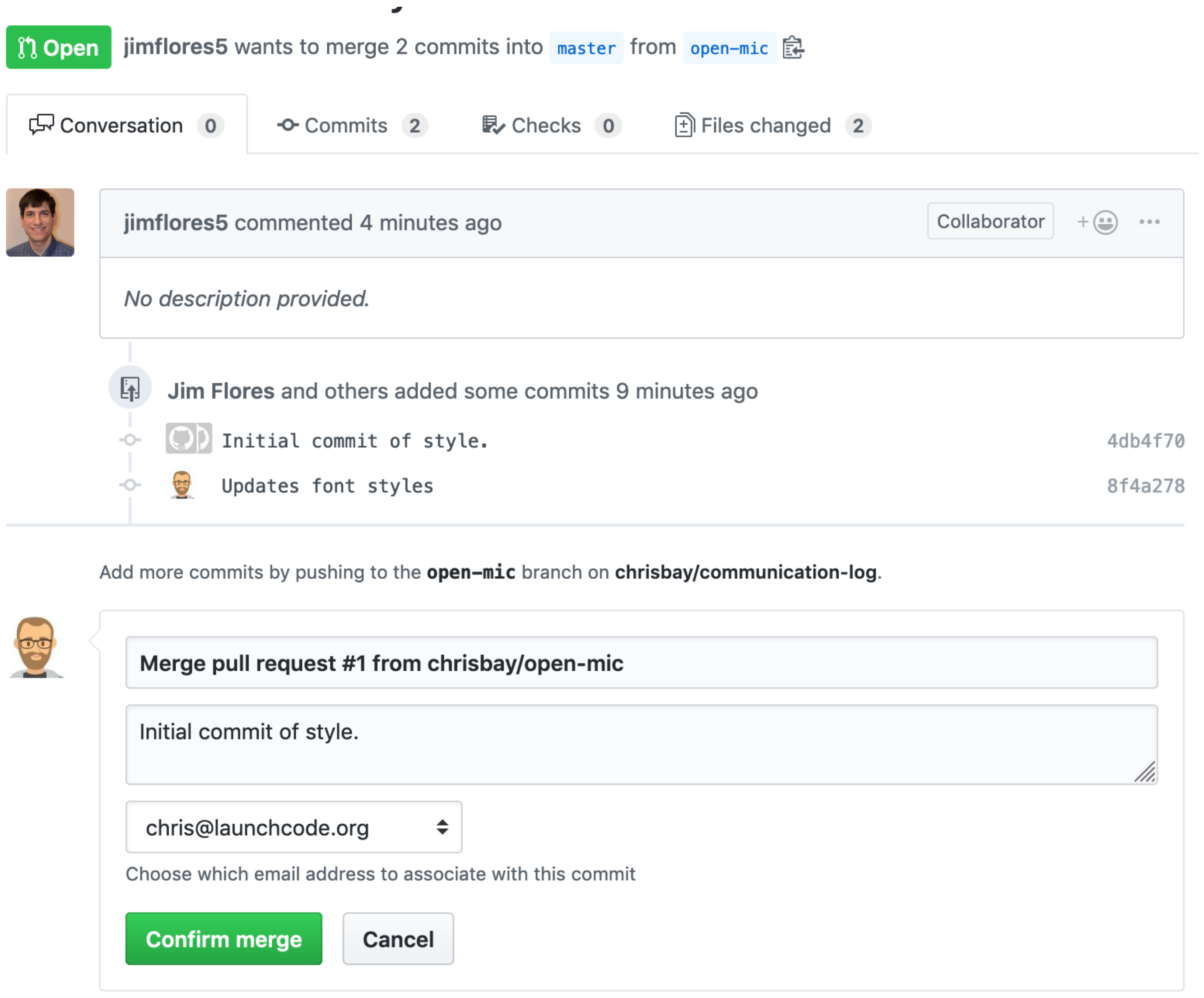
Continuous integration has not been set up
Several apps are available to automatically catch bugs and enforce style.

This branch has no conflicts with the base branch
Merging can be performed automatically.

Merge pull request — You can also open this in GitHub Desktop or view command line instructions.

Merge a Pull Request in GitHub

When you’re happy with the changes, merge them in. Click *Merge Pull Request* then *Confirm Merge*.


Open jimflores5 wants to merge 2 commits into `master` from `open-mic`

Conversation 0 | Commits 2 | Checks 0 | Files changed 2

jimflores5 commented 4 minutes ago — Collaborator

No description provided.

Jim Flores and others added some commits 9 minutes ago

```
Initial commit of style.                4db4f70
Updates font styles                     8f4a278
```

Add more commits by pushing to the **`open-mic`** branch on **chrisbay/communication-log**.

Merge pull request #1 from chrisbay/open-mic

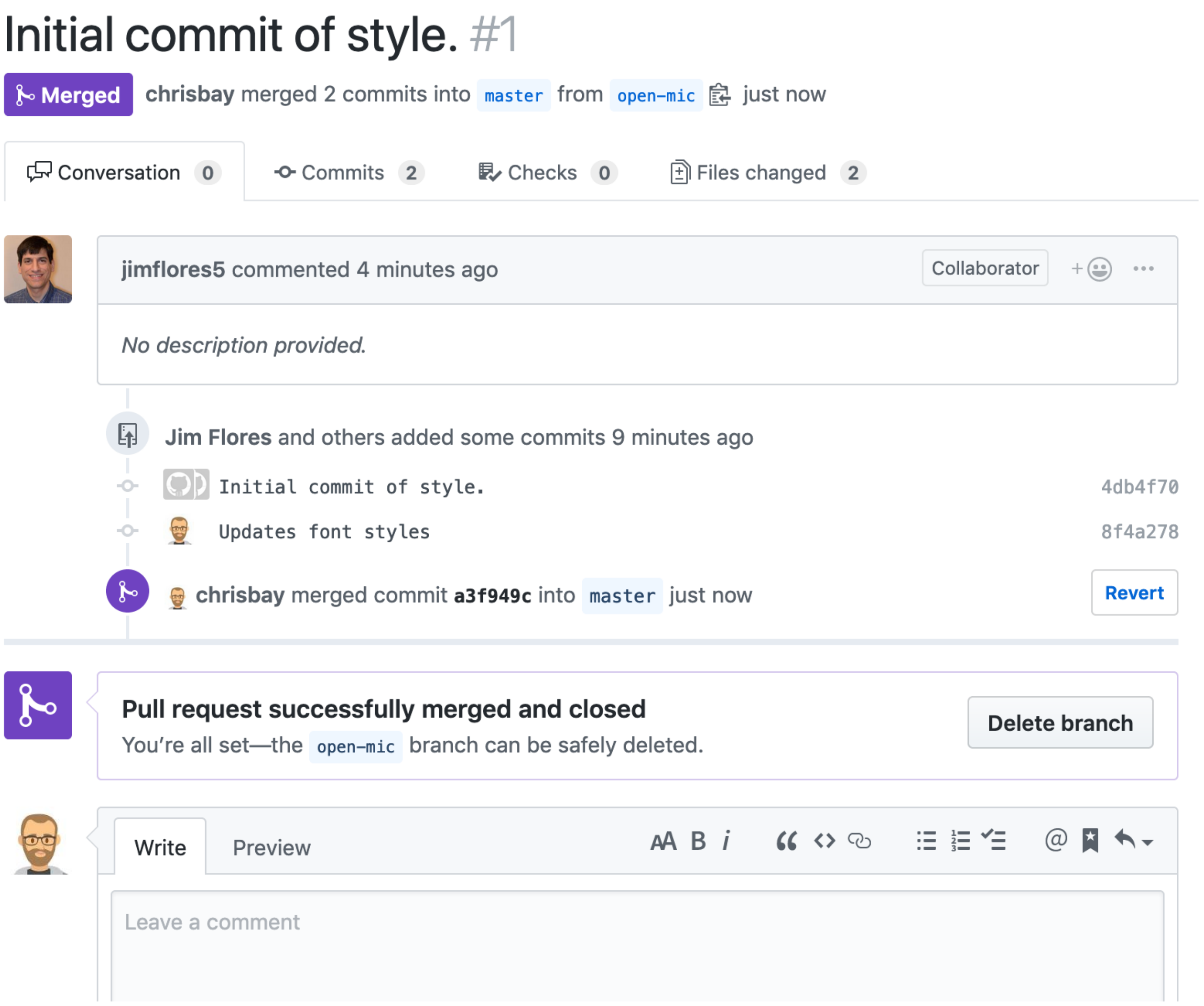
Initial commit of style.

chris@launchcode.org

Choose which email address to associate with this commit

Confirm PR Merge in GitHub

Upon a successful merge, you should see a screen similar to the following:


Initial commit of style. #1

Merged chrisbay merged 2 commits into `master` from `open-mic` just now

jimflores5 commented 4 minutes ago

No description provided.

Jim Flores and others added some commits 9 minutes ago

```
Initial commit of style.    4db4f70
Updates font styles         8f4a278
```

chrisbay merged commit **a3f949c** into `master` just now

Pull request successfully merged and closed

You're all set—the `open-mic` branch can be safely deleted.

PR Merged in GitHub

The changes from `open-mic` are now in the `master` branch, but only in the remote repository on GitHub. You will need to pull the updates to your `master` for them to be present locally.

```
$ git checkout master
$ git pull origin master
```

Git is able to merge these files on its own.

22.7.3.12. Step 12: Merge Conflicts!

When collaborating on a project, things won't always go smoothly. It's common for two people to make changes to the same line(s) of code, at roughly the same time, which will prevent Git from being able to merge the changes together.

Git Merge Conflicts

This isn't such a big deal. In fact, it's very common. To see how we can handle such a situation, we'll intentionally create a merge conflict and then resolve it.

Pilot: Let's change something about the style file. Our HTML is looking pretty plain, so let's pick a nice font and add some margins.

First, switch back to the `master` branch.

```
$ git checkout master
```

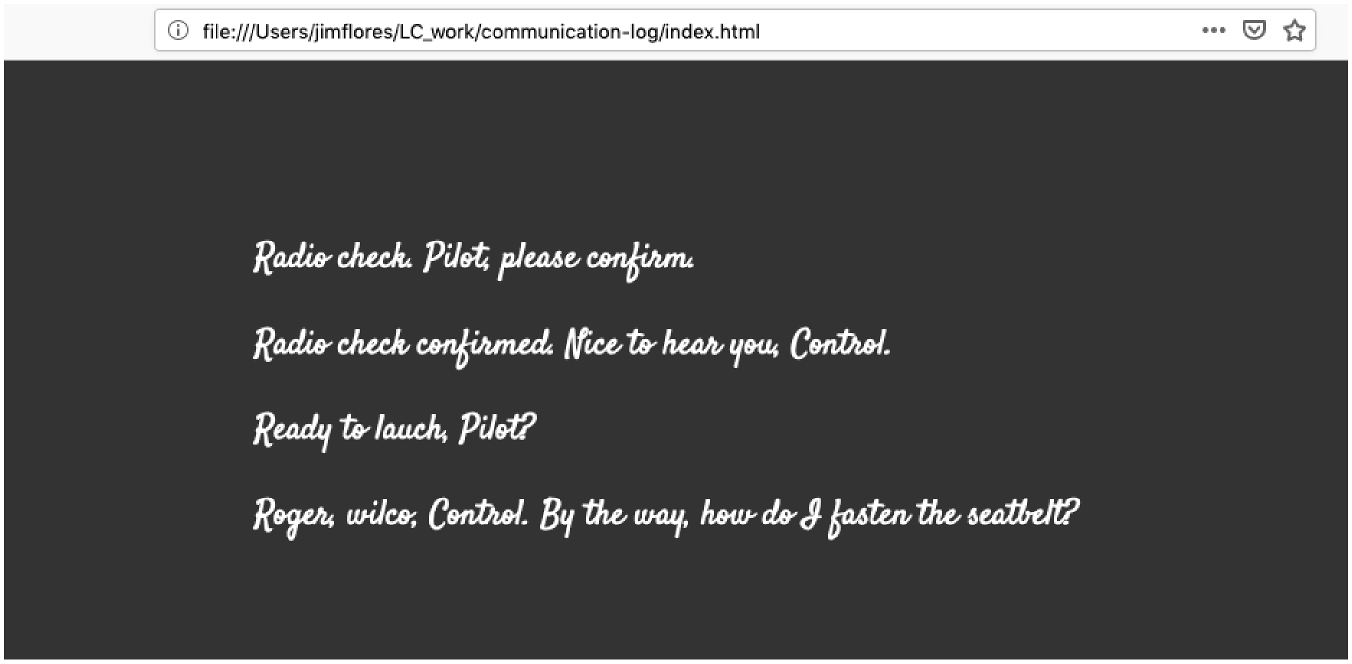
Let's change our font. To do so, add this link to your `index.html` file, right after the first stylesheet link:

```
<link href="https://fonts.googleapis.com/css?family=Satisfy" rel="stylesheet">
```

And spice up your `style.css` file to look like this:

```
body {
  color: white;
  background-color: #333;
  font-size: 150%;
  font-family: 'Satisfy', cursive;
  margin: 5em 25%;
}
```

The result:


file:///Users/jimflores/LC_work/communication-log/index.html

Radio check. Pilot, please confirm.

Radio check confirmed. Nice to hear you, Control.

Ready to lauch, Pilot?

Roger, wilco, Control. By the way, how do I fasten the seatbelt?

Satisfying!

Stage and commit your changes and push them up to GitHub. If you don't remember how to do this, follow the instructions above. Make sure you're back in the `master` branch! If you're still in `open-mic`, then your changes will be isolated, and you won't get the merge conflict you need to learn about.

Meanwhile…

Control: Let's change something about the style file that Pilot just edited. Change it to look like this:

```
body {
  color: white;
  background-color: black;
  font-family: 'Sacramento', cursive;
  font-size: 32px;
  margin-top: 5%;
  margin-left: 20%;
  margin-right: 20%;
}
```

Don't forget to link the new font in your `index.html` file, after the other link:

```
<link href="https://fonts.googleapis.com/css?family=Sacramento" rel="stylesheet">
```

Commit your changes to branch `master`.

22.7.3.13. Step 13: Resolving Merge Conflicts

Control: Try to push your changes up to GitHub. You should get an error message. How exciting!

```
$ git push origin master

To git@github.com:chrisbay/communication-log.git
! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'git@github.com:chrisbay/communication-log.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
```

```
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

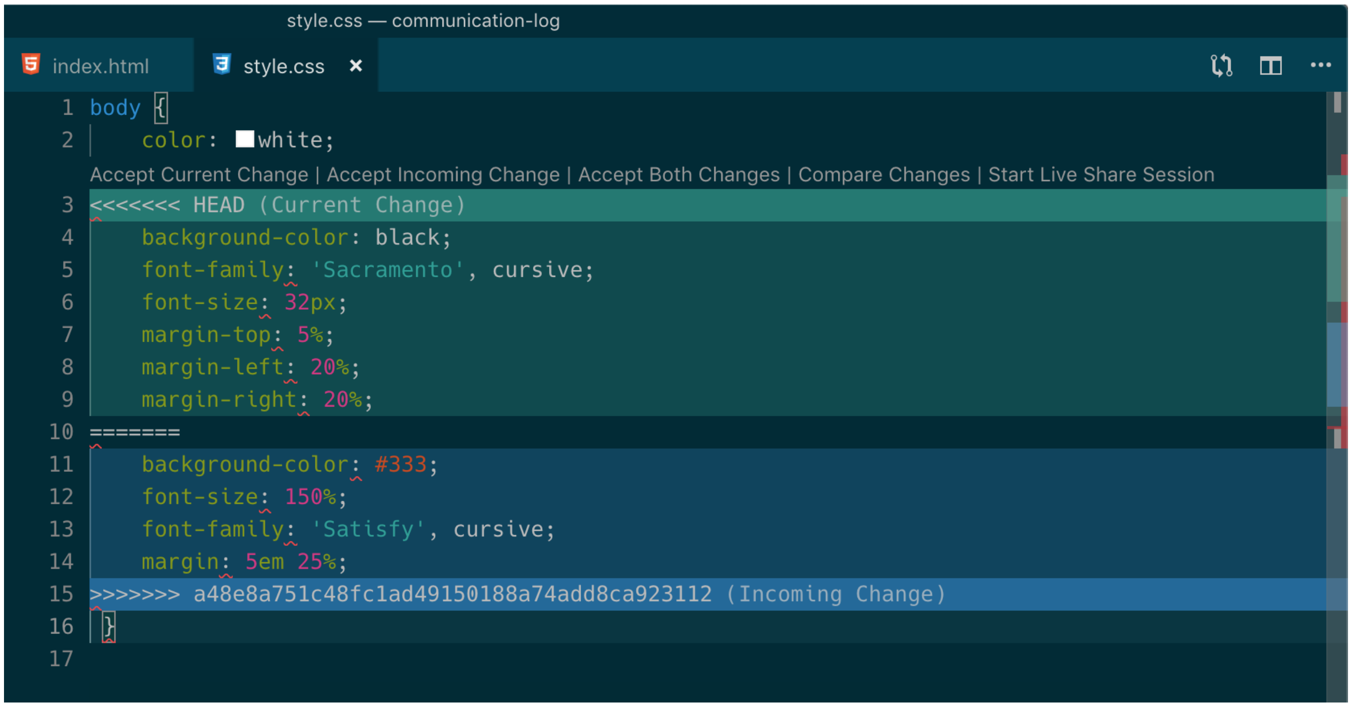
There's a lot of jargon in that message, including some terminology we haven't encountered. However, the core of the message is indeed understandable to us: "Updates were rejected because the remote contains work that you do not have locally." In other words, somebody (Pilot, in this case), pushed changes to the same branch, and you don't have those changes on your computer. Git will not let you push to a branch in another repository unless you have incorporated all of the work present in that branch.

Let's pull these outstanding changes into our branch and resolve the errors.

```
$ git pull
remote: Counting objects: 4, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 1), reused 4 (delta 1), pack-reused 0
Unpacking objects: 100% (4/4), done.
From github.com:chrisbay/communication-log
   7d7e42e..0c21659  master     -> origin/master
Auto-merging style.css
CONFLICT (content): Merge conflict in style.css
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
```

Since Pilot made changes to some of the same lines you did, Git was unable to automatically merge the changes.

The specific locations where Git could not automatically merge files are indicated by the lines that begin with `CONFLICT`. You will have to edit these files yourself to incorporate Pilot's changes. Let's start with `style.css`.


style.css — communication-log
index.html
style.css
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes | Start Live Share Session

```
body {
    color: white;
<<<<<<< HEAD (Current Change)
    background-color: black;
    font-family: 'Sacramento', cursive;
    font-size: 32px;
    margin-top: 5%;
    margin-left: 20%;
    margin-right: 20%;
=======
    background-color: #333;
    font-size: 150%;
    font-family: 'Satisfy', cursive;
    margin: 5em 25%;
>>>>>>> a48e8a751c48fc1ad49150188a74add8ca923112 (Incoming Change)
}
```

Merge conflicts in `style.css`, viewed in VS Code

At the top and bottom, there is some code that could be merged without issue.

Between the `<<<<<<< HEAD` and `=======` symbols is the version of the code that exists locally. These are *your* changes.

Between `=======` and `>>>>>>> a48e8a75...` are the changes that Pilot made (the hash `a48e8a75...` will be unique to the commit, so you'll see something slightly different on your screen).

Let's unify our code. Change the CSS to look like this, making sure to remove the Git markers so that only valid CSS remains in the file.

```css
body {
  color: white;
  background-color: black;
  font-family: 'Sacramento', cursive;
  font-size: 150%;
  margin: 5em 25%;
}
```

Tip

Like many other editors, VS Code provides fancy buttons to allow you to resolve individual merge conflicts with a single click. There's nothing magic about these buttons; they do the same thing that you can do by directly editing the file.
You will need to do the same thing for the `index.html` file. You only need the link for the Sacramento font, not the Satisfy font. Then stage, commit, and push your changes; you should not see an error message this time.

22.7.3.14. Step 14: Pulling the Merged Code

Pilot: Meanwhile, Pilot is sitting at home, minding their own business. A random `git status` seems reassuring:

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

Your local Git thinks the status is quo. Little does it know that up at GitHub, the status is not quo. We'd find this out by doing either a `git fetch`, or if we just want the latest version of this branch, `git pull`:

```
$ git pull
remote: Counting objects: 13, done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 13 (delta 4), reused 13 (delta 4), pack-reused 0
Unpacking objects: 100% (13/13), done.
From Github.com:chrisbay/communication-log
   0c21659..e0de62d  master     -> origin/master
Updating 0c21659..e0de62d
Fast-forward
index.html | 3 ++-
style.css  | 4 ++--
2 files changed, 4 insertions(+), 3 deletions(-)
```

Great Scott! Looks like Control changed both `index.html` and `style.css`. Note that *Pilot* didn't have to deal with the hassle of resolving merge conflicts. Since Control intervened, Git assumes that the team is okay with the way they resolved it, and *fast forwards* our local repo to be in sync with the remote one. Let's look at `style.css` to make sure:

```css
body {
  color: white;
  background-color: black;
  font-family: 'Sacramento', cursive;
  font-size: 150%;
  margin: 5em 25%;
}
```

22.7.3.15. Step 15: More Merge Conflicts!

Let’s turn the tables on the steps we just carried out, so Pilot can practice resolving merge conflicts.

1. **Control and Pilot**: Confer to determine the particular lines in the code that you will both change. Make different changes in those places.
2. **Control**: Stage, commit, and push your changes.
3. **Pilot**: Try to pull in Control’s changes, and notice that there are merge conflicts. Resolve these conflicts as we did above (ask Control for help, if you’re uncertain about the process). Then stage, commit, and push your changes.
4. **Control**: Pull in the changes that Pilot pushed, including the resolved merge conflicts.

Merge conflicts are a part of the process of team development. Resolve them carefully in order to avoid bugs in your code.

22.7.3.16. Resources

- Git Branching - Basic Branching and Merging
- Adding Another Person To Your Repository
- Resolving Conflicts In the Command Line