
CS 320: Lecture: Version Control

git theory

Version Control

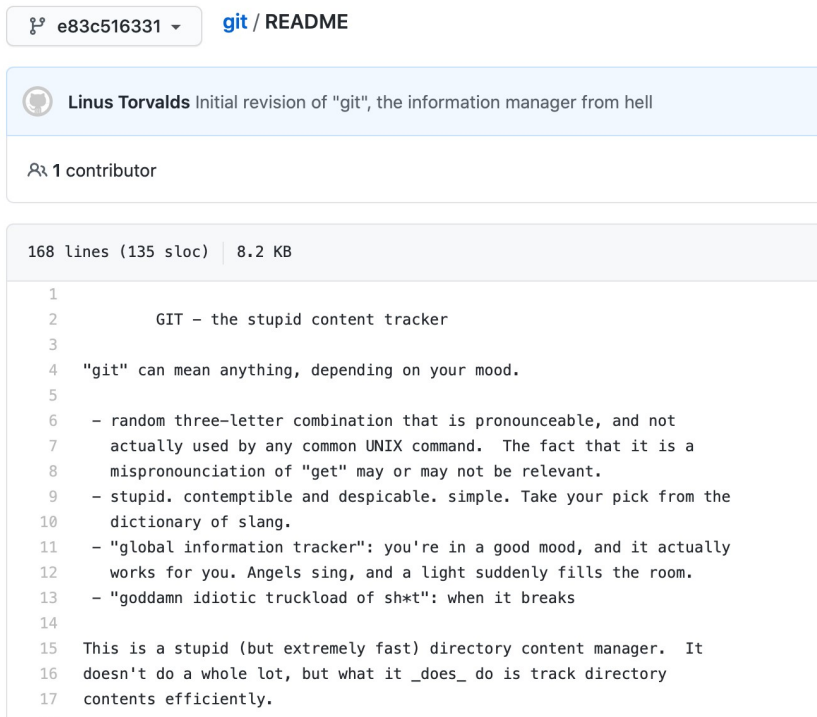
Version control is a way to keep track of changes made over time and enables the ability to revert to certain changes if necessary. Some version control systems record only changes, but git records something more akin to snapshots. However, at each snapshot, it does not re-store files if they have not been changed. Instead, it stores a link to the previous identical file. This makes git much more efficient than other version control systems.

At this stage (and most stages) of your CS career, it is not very important to understand *how* git works. It works, and it works well. Knowing how to use it is critical and will make you much more employable because everyone uses git. This is how you will work on projects for the rest of your technical life, so it is time to **git gud**.

A Brief History of git

To talk about the history of git you must first talk about the history of Linux. The Linux kernel is a very large and very old open source project. Open Source means that it has many people working on it, not connected to any organization or business. This creates a problem, however; how do contributors share and combine their work, with everyone working at the same time? The answer is version control. In the beginning, also known as the 90s, changes were passed around as patches and archived files. This was not much better than emailing files back and forth. In 2002, the Linux Community moved to BitKeeper, a proprietary, but at the time free, version control software. Then in 2005, it became not free. This made a lot of people unhappy and was widely considered a bad move. Linux people did not want to go back to emailing files back and forth, so Linus Torvalds did what Linus Torvalds does, and made his own version control system. Thus, git was born. It had a lot of improvements over other version control systems, like branching, and quickly became the most widely used version control system, much to the dismay of everyone trying to make money off of version control. It remains free and open source, managed by a non-profit. If you want to know anything (or everything) there is to know about git, you can go to <https://git-scm.com> where there are comprehensive reference manuals and guides, all for free.

What is git?



```

1
2     GIT - the stupid content tracker
3
4     "git" can mean anything, depending on your mood.
5
6     - random three-letter combination that is pronounceable, and not
7       actually used by any common UNIX command. The fact that it is a
8       mispronunciation of "get" may or may not be relevant.
9     - stupid. contemptible and despicable. simple. Take your pick from the
10      dictionary of slang.
11     - "global information tracker": you're in a good mood, and it actually
12       works for you. Angels sing, and a light suddenly fills the room.
13     - "goddamn idiotic truckload of sh*t": when it breaks
14
15     This is a stupid (but extremely fast) directory content manager. It
16     doesn't do a whole lot, but what it _does_ do is track directory
17     contents efficiently.

```

From the first git README

git set up

Part 1: Create a GitHub account

- Go to <https://github.com> and sign up
- Verify your e-mail
 - If you did not use a .edu email, go to your account settings and add your .edu email as an alternate email: Profile (top right corner) > Settings > Email
- You can then go to <https://education.github.com/pack> and get the Student Developer pack. This will give you access to thousands of dollars of free credits and materials on tons of platforms

Part 2: Pick a Git Client

Git Clients are a visual alternative to doing git through command line. There are many different options for Git Clients. This page walks you through using two free choices: GitHub Desktop and GitKraken. GitHub Desktop is bare bones and very easy to use, but doesn't have many extra features. GitKraken has more features (and a pro version), and also has a visual display of your repository's branches. (more on branches later)

Downloading GitHub Desktop

- Go to <https://desktop.github.com>
- Make sure you move it from downloads, as it will run in your downloads folder if you let it.
- Sign in to GitHub.com
- You will then be directed to "Configure Git"
- Enter the Name/Email you want to be associated with your commits
- Click continue
- It will ask for usage stats.
- Click Finish

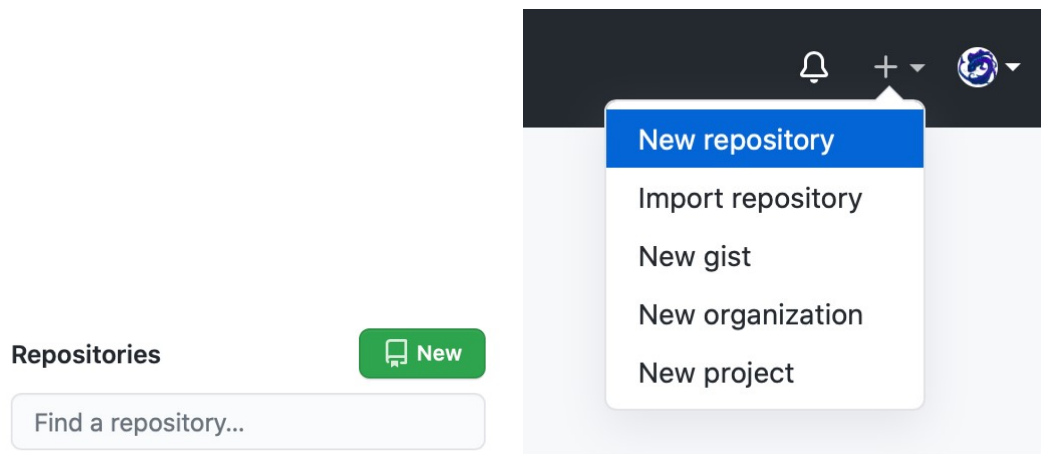
Downloading GitKraken

- Go to <https://www.gitkraken.com>
- Enjoy the adorable loading animation
- Sign in with GitHub
 - If you are already logged in to GitHub this is as easy as clicking authorize App
- Fill out your profile name, and then your name and email that will be associated with your commits
- Click okay

Part 3: Create a Repository

A repository (or repo) is where your code is stored. You have a remote repository, that is located on a service like GitHub, and a local repository, on your machine. You work on the code on your local repository, and then commit your changes and push them to your remote repository. When you are starting a new repository it is always easier to create your remote repository first, and then bring it to your machine.

On GitHub



To create a new repository on GitHub, you can either hit the green "New" button next to "Repositories" on the left of the home page, or click the "+" menu by your profile in the upper right and select "New Repository." This will take you to a new page:

Create a new repository

Repository template

Start your repository with a template repository's contents.

No template ▾

Owner *

 KaiFish ▾

Repository name *

/ unladen-swallow ✓

Great repository names are short and memorable. Need inspiration? How about [congenial-fortnight?](#)

Description (optional)

Intro to Git for CS320

☒  **Public**

Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**

You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

☒ **Add a README file**

This is where you can write a long description for your project. [Learn more.](#)

☒ **Add .gitignore**


Choose which files not to track from a list of templates. [Learn more.](#)

.gitignore template: **Java** ▾

☒ **Choose a license**

A license tells others what they can and can't do with your code. [Learn more.](#)

License: **MIT License** ▾

This will set  **main** as the default branch. Change the default name in your [settings](#).

Create repository

In this form:

- Enter a repository name
- Select public or private.
 - Public repositories can be seen by everyone, private repositories can only be seen by added collaborators

Optional:

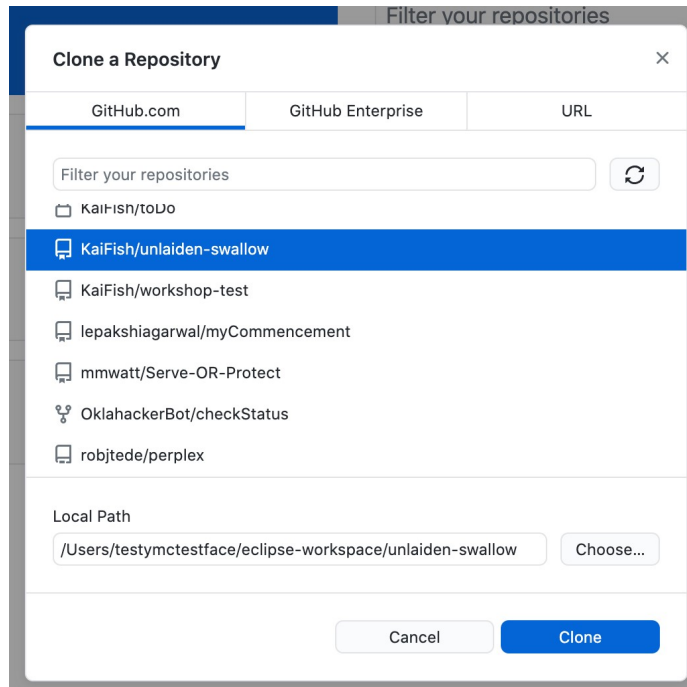
- Add a README - This is a long description of your repository and what it's about (this is highly recommended)
- Add .gitignore - This file tells git which files in your repo not to track. It is very useful for Java because you do not want to commit .class files (a repository is only for source code)
 - GitHub is nice and has template .gitignore files based on the language you are using.
- Add a License - GitHub is dedicated to supporting open source, but an important part of open source is letting people know what can and cannot be done with your code. A license is a document that states those rules. GitHub has <https://choosealicense.com> to help you pick one.

Click the green "Create Repository" Button when you are done.

Part 4: Clone the Repository

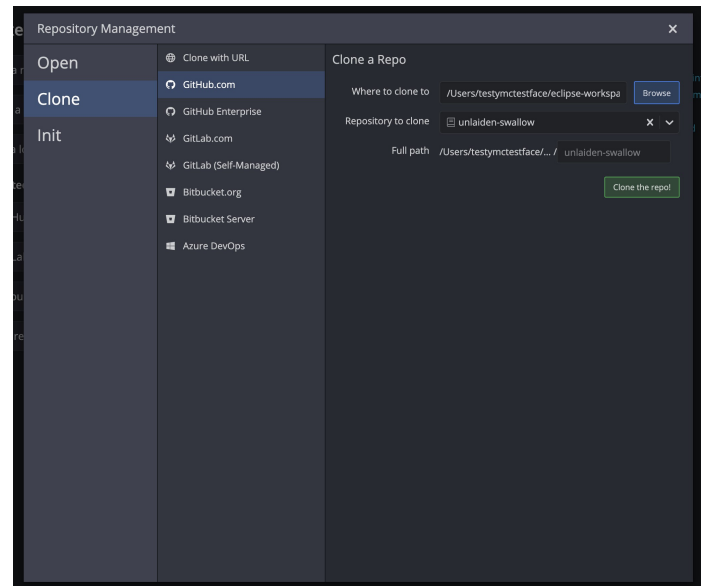
The process of bringing a remote (online) repository to your computer is called cloning. On your preferred git client select the option for cloning your repository

Github Desktop



- Click "Clone a Repository from the Internet..."
 - You will be able to search the repositories connected to your GitHub account and select the one you want to clone. It will show you the "Local Path" where it will put the repository. Select "Choose" to change this.
 - **IF YOU ARE USING ECLIPSE SELECT YOUR ECLIPSE WORKSPACE FOLDER AS THE LOCATION**
- Click "Clone"

GitKraken

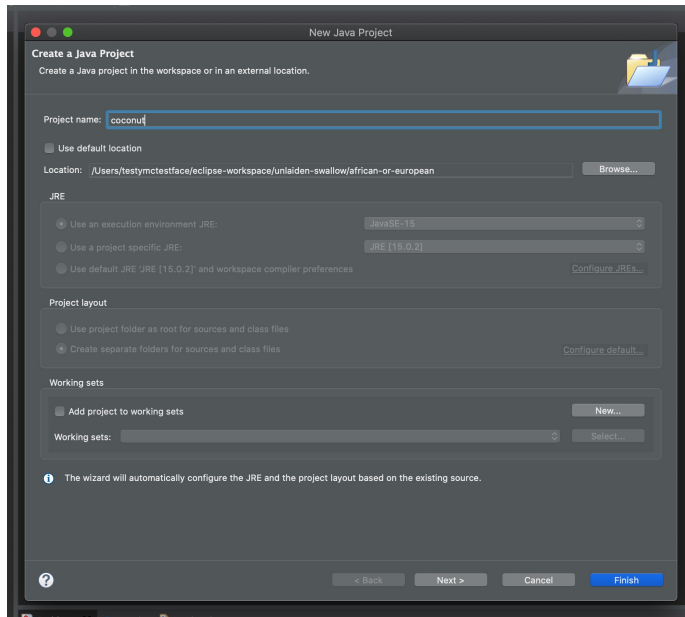


- Click "Clone a repo"
- Select GitHub.com as the source
- Select the location where GitKraken will put the repository beside "Where to clone to"
 - **IF YOU ARE USING ECLIPSE SELECT YOUR ECLIPSE WORKSPACE FOLDER AS THE LOCATION**
- Select the Repository you want to clone from the "Repository to clone" drop down
- Click "Clone the repo!"

Part 5: Repositories and Eclipse

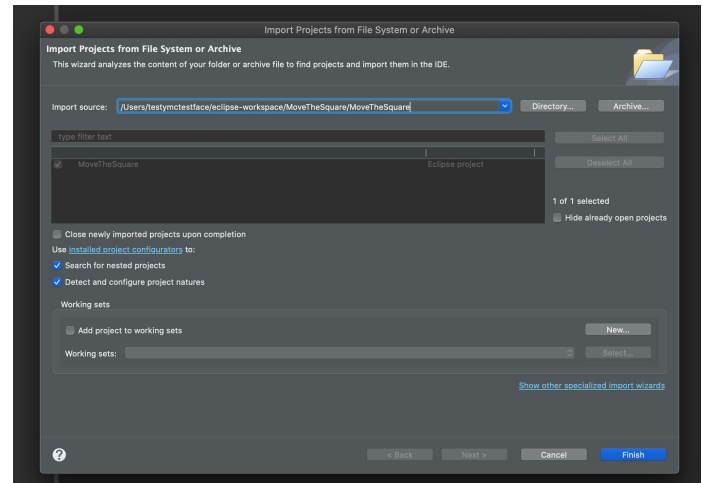
Eclipse is fussy when it comes to repositories. Below are the instructions for using a repository with no existing Eclipse Project, and for getting an Eclipse Project from a repository you have cloned to your machine. **THIS WILL ONLY WORK IF YOU CLONED THE REPOSITORY INTO YOUR ECLIPSE WORKSPACE.**

No Existing Eclipse Project



- Click "Create a Java Project"
- UNCHECK "Use Default Location"
- Name your Project
- For Location select "Browse"
- Navigate to your repository folder in your Eclipse Workspace. In your repository, make a new folder. Select this as the location for your Project
- Click "Finish"

Existing Eclipse Project



- File > Open Projects from File System
- Click Directory
- In your repository folder, in the Eclipse workspace, select the Eclipse Project folder, click "Open"
- Click "Finish"

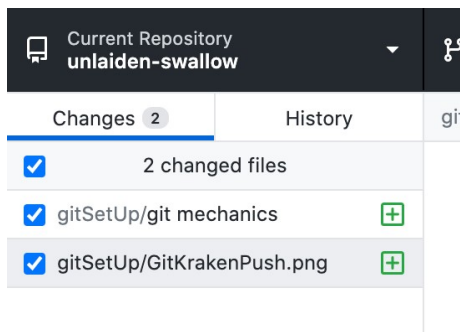
git mechanics

How to git it

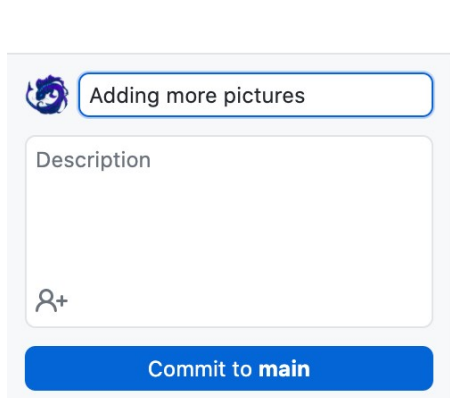
Git saves your work, but only when you tell it to. The process for telling it what to save to your repository is called committing. The repository on your machine is called your local repository. The repository on GitHub is called your remote or upstream repository. To save changes to your local repository you do "git add" and then "git commit". To move changes from your local repository to your remote repository you must do a "git push" git tracks any and all changes made to the repository. These changes, until you tell git otherwise, are "unstaged," which means that git will not record them. To stage these changes you must do a "git add". To add these to your local repository you must do a "git commit" The direction below explain how to do this process on GitHub Desktop and GitKraken.

GitHub Desktop

The Panel on the left shows you all the files that have been changed. GitHub Desktop automatically stages (adds) these for you, which is indicated by a check mark. If you DO NOT want to add a file, you can uncheck it to unstage it.



You can click on any of the file names and GitHub Desktop will show you the changes made in the file between versions (This is called a "git diff")



To commit these files to your repository you MUST add a commit message in the "Summary" field. The message should explain what you did in this commit.

If you want to be more detailed you can add more information in the "Description" field.

When you are ready to commit, click the blue "Commit" button at the bottom of the right panel.

Once you have committed your changes, the main section will offer "some friendly suggestions on what to do next" (If it doesn't, select the "changes" tab on the left panel)



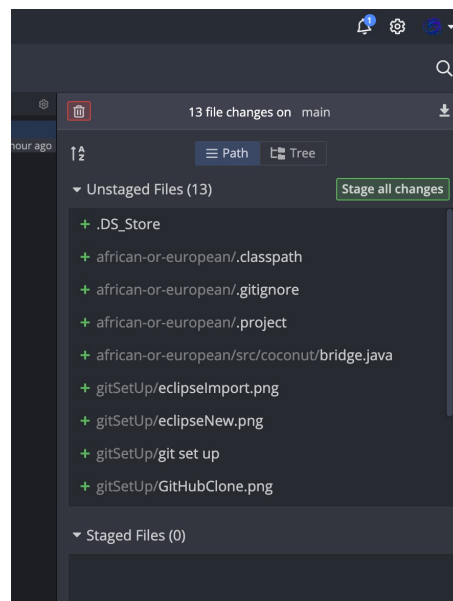
GitKraken

The panel on the right shows you all the files that have been changed under "Unstaged Files".

You can click on any of the file names and GitKraken will show you any changes made between versions. (This is called a "git diff")

To stage these files click "Stage all changes"

You can also stage files individually by hovering over the file name and selecting "Stage File"

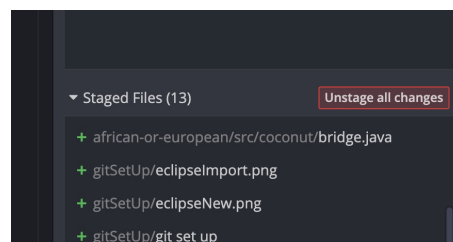


Once files are staged they will appear under "Staged Files"

To commit these files to your repository you MUST add a commit message in the "Summary" field. The message should explain what you did in this commit.

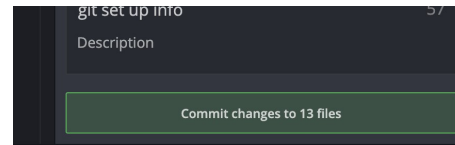
If you want to be more detailed you can add more information in the "Description" field.

When you are ready to commit, click the green "Commit changes" Button

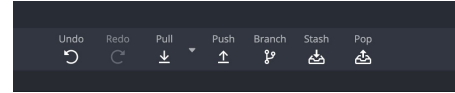


Here, you can click "Push origin" and push your changes to GitHub. (Origin is another name for remote)

There is also a "Push origin" button on the top bar of the window, with a number indicating how many commits you are pushing.



To push these changes to your remote repository, click the "Push" button at the top of the window.

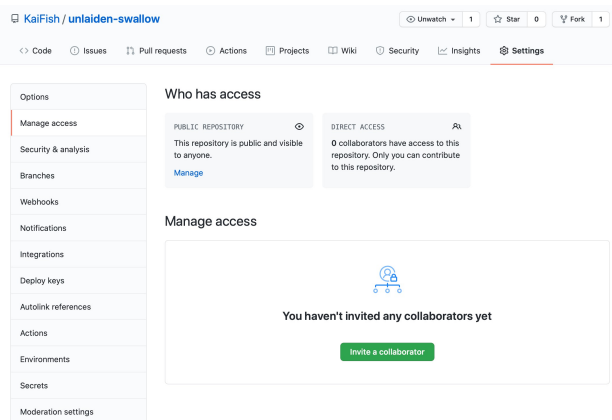


git collaboration

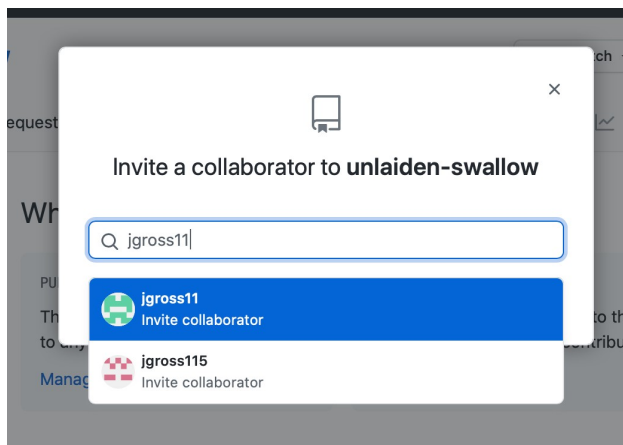
The process of add, commit, push is all good and fine when you are working by yourself. **IT DOES NOT WORK** when you are working in a team. To effectively work in a team you must use branches. Before you branch, you must pull. Before you do any of that, you must add collaborators to your repository.

Step 0: Add Collaborators

Collaborators are people who are authorized to work on your repository. To add them go to the settings tab and click "Manage access" and then "Invite a collaborator"

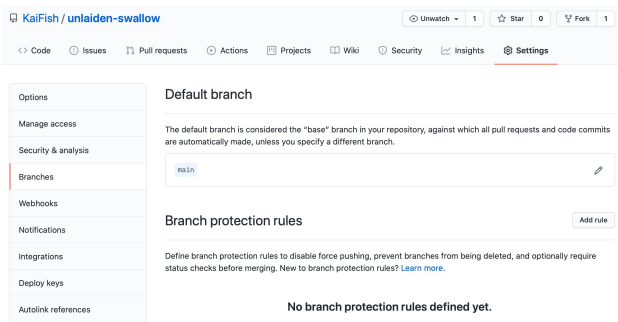


A window will come up where you can look up GitHub users by username or email. Select who you want to invite, and GitHub will send an invite to their email. They must accept this email before they can do anything with the repo.



A Word of Advice

Collaborators can do anything to your repository that you can do, with few limitations. When people do not know what they are doing, or abuse their privileges, things can get very messy very quickly. While you are in the Settings menu, it is highly recommended that you click on "Branches" and then "Add rule"



After that apply the following settings and click the green "Create" button:

Branch protection rule

Branch name pattern

Protect matching branches

☒ **Require pull request reviews before merging**
 When enabled, all commits must be made to a non-protected branch and submitted via a pull request with the required number of approving reviews and no changes requested before it can be merged into a branch that matches this rule.
 Required approving reviews: 1 ▼

☐ **Dismiss stale pull request approvals when new commits are pushed**
 New reviewable commits pushed to a matching branch will dismiss pull request review approvals.

☐ **Require review from Code Owners**
 Require an approved review in pull requests including files with a designated code owner.

☐ **Require status checks to pass before merging**
 Choose which [status checks](#) must pass before branches can be merged into a branch that matches this rule. When enabled, commits must first be pushed to another branch, then merged or pushed directly to a branch that matches this rule after status checks have passed.

☐ **Require signed commits**
 Commits pushed to matching branches must have verified signatures.

☐ **Require linear history**
 Prevent merge commits from being pushed to matching branches.

☒ **Include administrators**
 Enforce all configured restrictions above for administrators.

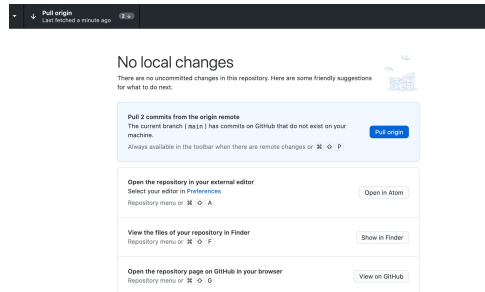
These settings means no one can make and apply changes without someone else reviewing and approving them. This will make your life easier.

Step 1: pull

Before you make a branch you must check to make sure there are no new changes on the remote. To get these changes you must "fetch" the remote and "pull" the changes into your local repository

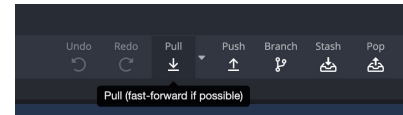
GitHub Desktop

GitHub Desktop will usually regularly fetch for you, but it is good practice to click the "Fetch" button in the top bar to make sure it is up to date. If there are changes, the "Fetch" button will change to a "Pull" button. Clicking "Pull" will merge these changes into your local repository. If this does not work, go to the section labeled "git conflicts"



GitKraken

GitKraken will fetch and pull in one click. All you need to do is click the "Pull" button at the top of the window. If this does not work, go to the section labeled "git conflicts"

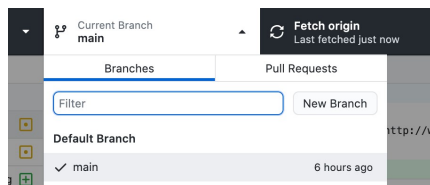


Step 2: branch

Your main branch should always contain working code. You should **never** program on your main branch. Any new features, patches, bug fixes, etc should have their **own** branch. This helps prevent merge conflicts and ensures that the main branch is always working. As previously mentioned, you should always pull from the main remote branch, and then create a new branch.

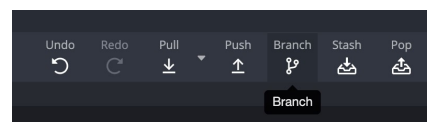
GitHub Desktop

GitHub Desktop has a tab on the top bar that tells you which branch you are on. Clicking this will show all your branches and give you the opportunity to create a new branch by clicking "New branch." Your branch name should describe what is being done on it.



GitKraken

GitKraken shows you your current branch on the left of the top bar. You can create a new branch by clicking the "Branch" button in the middle of the top bar. Your branch name should describe what is being done on it.



Step 3: code

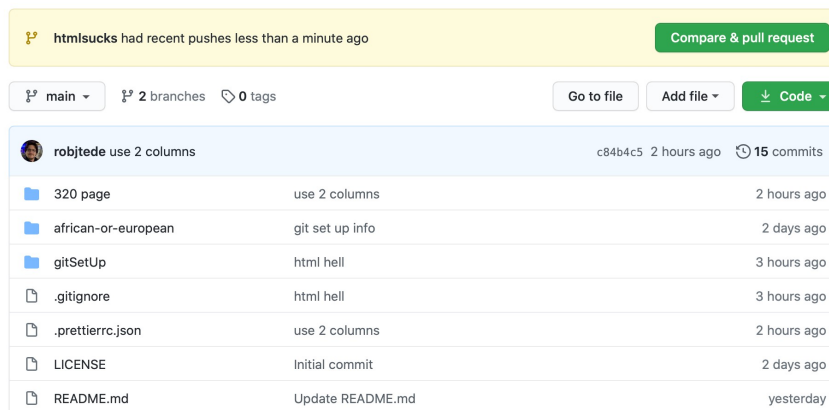
This step should be obvious, but you never know. Once you are on your branch, you may proceed with work as normal.

Step 4: commit

You should commit at logical checkpoints in your progress, like fixing a bug, or finishing a section of logic. When you commit, you should follow the steps as outlined in git mechanics. They will work the same way, except instead of pushing to the main branch, you will be pushing your branch to the remote repository. (GitHub Desktop calls this "Publishing" the branch.) Once your branch is in the remote repository **and you are ready to merge it to the main branch**, you may proceed to step 5.

Step 5: pull request

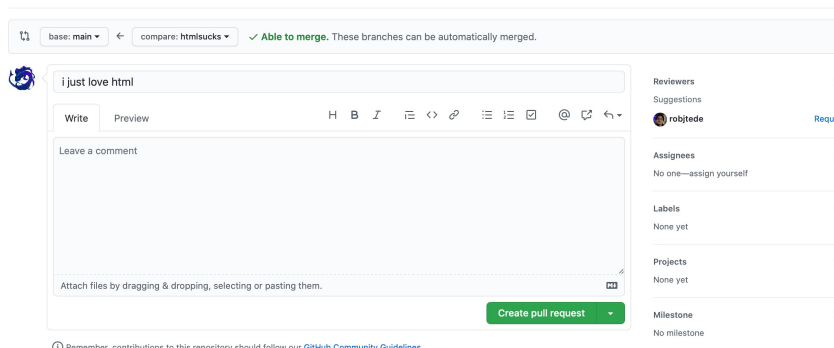
It is time to return to your repository on GitHub, and refresh the page if you have left it open. At the top of the repository you should see and alert that "[branch-name] had recent pushes" with a green button that says "Compare & pull request." Click the button.



This will lead you to a form for opening a pull request, comparing your branch to the main branch. It will auto fill with your commit message as the title, but you can add more description of what you did in the branch in the comment field. When you are done, click the "Create pull request" button.

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).



If you set up the previously recommended repository rule, one of your collaborators must review and approve your pull request. The format of the pull request allows for comments and discussion, so teammates can discuss changes. If you did not, you should be able to merge it automatically, by clicking the "Merge pull request" button. If you cannot merge it, GitHub will not let you, because you have a merge conflict. Please go to the section labeled "merge conflicts"

to learn more about this. Otherwise, congratulations, you have merged your branch! Return to step 1.

git conflicts

this will be a different lecture. please come back later. if you have a problem contact a tutor

[Copyright \(c\) 2006–2021](#) | Unless indicated otherwise, content is freely redistributable |

