So, You Want to Use Git?

A Starter's Guide to Git Including:

GitHub
GitLab
GitKraken
The Terminal

Mark C. Anderson
Brigham Young University
Version 0.1.2

Why was this tutorial written?

Git is a wonderful tool designed to help you write even better code, work with files, and collaborate with others. It does this by keeping track of changes that you make to your files and allowing you to easily access old versions. Git also enables you to work with others on a common project by letting everyone access the files, make improvements, and then merge all the improvements back into the original project. Git is used by individuals, large corporations, and everyone in between.

However, Git has a notoriously steep learning curve. If you've previously tried learning Git you may have been faced with a mess of command line text, never quite understanding what was happening. It's hard to keep trudging through that, and that's where this tutorial comes in handy. This tutorial aims to teach the reader how to use Git by first explaining what Git is and how it will benefit them, and then demonstrating click by click how to interface with the common online Git providers GitHub and GitLab, all without opening the terminal.

After the reader is comfortable using an online Git provider, subsequent sections demonstrate how to use Git on a Desktop through GitKraken and the finally the terminal. These are elaborately described with screenshots showing every step. The section that uses the terminal shows every line that must be used and explains what each line does in plain terms.

You may have noticed how long this tutorial appears to be. However, this tutorial is designed to be cherry-picked to pieces to meet your personal Git goals. If you just want a place to store files and track how they change over time, feel free to read the few pages that describe either GitHub or GitLab and then put the tutorial down. If you want an in-depth experience on how to work in teams and do it all from the terminal, feel free to read just the sections about the terminal, and then go to the Further Learning section listed at the end to expand your knowledge. If you want something in between, or are just exploring, feel free to read any sections you want.

For questions regarding this tutorial or if you have any recommendations for how to make it even better, feel free to email Mark Anderson at anderson.mark.az@gmail.com.

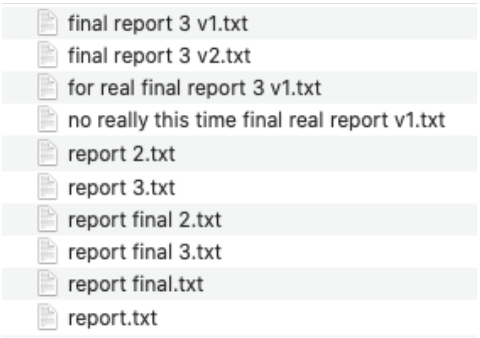
About this version:
The "What is Git" and "Quick Start Guide for Personal Use" chapters are complete. The content in this version is enough for a reader to install, use, and become competent in several different ways of using Git. The next chapter under development will be the "Working in Groups" chapter, followed by a potential "Advanced Topics" chapter as desired. Parts of the appendix are already written.

Anderson

1 What is Git?

1.1 Purpose

You know that feeling you get when you open a folder and see something like this?



Or when you try to change something in a file, decide it was a bad idea, but then can't get it to the way it was in before you made those changes? Git is the solution to handle all of these file versions. In fact, Git is a "Version Control System", or VCS for short. There are many other similar tools, but Git is the most widely adopted VCS all around the world for both small and large groups to handle their files.

Git enables you to collaborate with others on a single file. Imagine that one of your teammates creates a file and gives everyone access to it. You notice an improvement that you can make and so you go ahead and make it. Rather than mass-emailing out a new version of the file, Git lets you merge your new version with the original version, and everyone can automatically access your improved version. These practices can be scaled up to companies with thousands of employees or used as a simple VCS for personal use.

1.2 Git Workflow

1.2.1 Repositories

A group of files stored on Git is called a **repository**. A repository can be stored on your computer, online, or both. Git stores changes that are made to each file over time and lets you see that history. A single repository can be shared between many people, with each person adding to it.

1.2.2 Staging

Sometimes you are ready to save new versions of some of your files to Git, but not all of them. The **staging area** is a place that you can store files that you are ready to save (or commit) to Git.

1.2.3 Committing

Think of the phrase "commit to memory", which means that you memorize something and keep it in your brain for future use. That is exactly what a Git **commit** does. A Git commit takes all of the files that are in the staging area and "commits" them to Git, where they will forever be stored and can be accessed at any time. Each commit is a snapshot of the current state of your files, and Git lets you see the differences between your files over time from commit to commit.

1.2.4 Branching

You always want to have a current functioning version of your files. If your files contain computer code, then this means that you have a version of your files that you can use at any time and have

it work properly. This is difficult when you're trying to make improvements to your files because your changes may have unintended consequences. This difficulty is amplified when working in teams. To overcome this challenge, you create a **branch**, which makes a copy of the repository that you can work in, improve, mess up, and perfect until you're ready to bring those improvements to the functioning version of your files.

1.2.5 Merging

When you create a branch, improve your files, and then decide to bring those improvements back to the main branch (typically called the **master branch**), you do a **merge** of your branch with the master branch. You may experience some conflicting changes between your improvements and someone else's improvements, and these can be settled by using a **merge tool** to help you see any conflicts and resolve them. Once you merge your changes to the master branch this becomes the new current functioning version of your files.

1.3 Ways to Access Git

In this tutorial we will cover four ways to access Git. There are many other ways to access Git, so if you find another product or method feel free to explore it. Below are descriptions of the four methods that we will cover.

1.3.1 GitHub

GitHub is an online location to store Git repositories. The web interface lets you use Git without needing to use the terminal and provides a user-friendly and simple Git experience. GitHub is the most popular online Git provider in the world.[1]

1.3.2 GitLab

GitLab is an online location for Git repositories, very similar to GitHub. The interface is different, but all of the same functionality is provided. The decision of whether to you GitHub or GitLab depends on your needs, and both provide more than enough for the purposes of this tutorial.[2,3]

1.3.3 GitKraken

GitKraken is a tool that you can install on your computer to manage both your online and local Git repositories. This provides a nice way to visualize the Git workflow and interact with your files, again without needing to use the terminal.

1.3.4 The Terminal (or Command Line)

The terminal[4] is arguably the most powerful and efficient way to manage your Git repositories. The terminal provides a very flexible working environment and can be a fast and effective way to navigate files and commit your files to Git. This is where most online tutorials seem to start, and it understandably can look very intimidating. This is a normal feeling and as you learn more about the terminal and how to use Git you will become more comfortable using it.

---

[1] As of June 2020

[2] https://www.youtube.com/watch?v=s8DCpG1PeaU&feature=youtu.be

[3] The BYU physics department uses GitLab. To access, go to http://git.physics.byu.edu and sign in with your netID and password

[4] The terminal and the command line are synonyms in this tutorial, and the go-to word that we will use is "terminal"

1.4 How This Document Works

This tutorial is designed to be cherry-picked based on your individual Git needs and goals. The next section is a quick start guide for users that want to use Git for personal projects. The process is described using all four approaches (GitHub, GitLab, GitKraken, and the terminal). Choose the sections that are most relevant to your goals and read those. After that, the next section is a guide for users that want to work in teams and discusses how to use branches and merging for each of the four methods. The final section goes more in-depth on some advanced topics. If you are an absolute beginner (as we all are to begin with) then I personally recommend that you choose an online version (either GitHub or GitLab) to learn about and play with and then a local version (either GitKraken or the terminal). As you gain experience and confidence you will find that your productivity increases as you use Git, even if you feel like your productivity takes a hit for a little while as you are getting up to speed.

2 Quick Start Guide for Personal Use

The purpose of this chapter is to guide you to the point where you can use any of the four methods (GitHub, GitLab, GitKraken, and the terminal) to manage your personal files in Git repositories. As we do this, you will see screenshots and text telling you exactly what to do to get everything working. When you complete this chapter, feel free to explore all of the extra buttons and features in the different tools and you will learn even more. To complete the sections on GitKraken and the terminal you will need either a GitHub or GitLab account because we will be discussing how to use GitKraken and the terminal to manage **remote repositories**.

For this part of the tutorial we are going to be using two files:

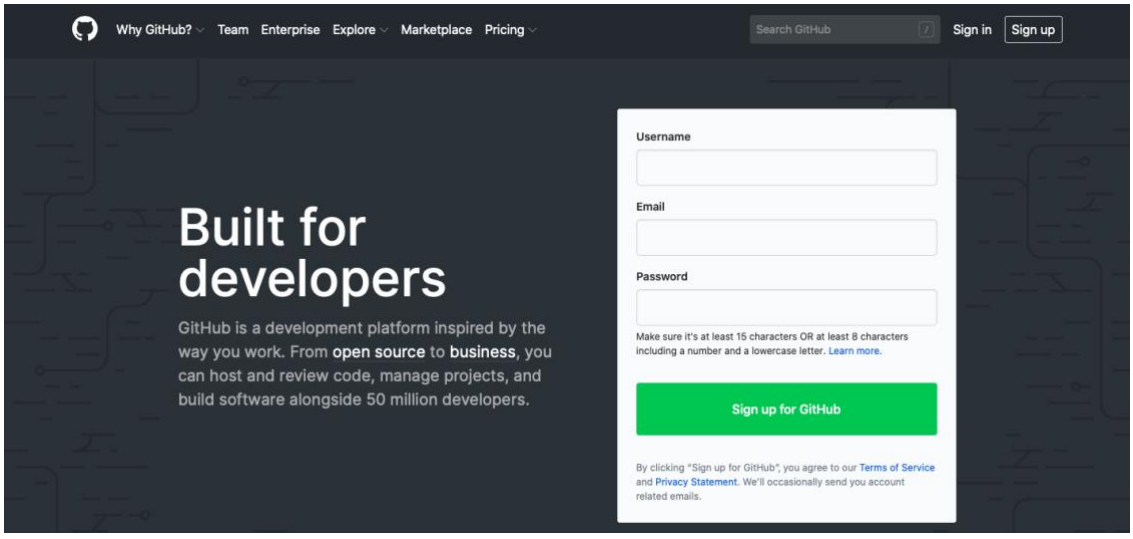
some_math.m
important-notes.txt

The some_math.m file will have some math written in the MATLAB programming language and important-notes.txt will have some plain text written in it. It will be easier to follow along if you create your own "practice" files. Your files don't have to match the names of the files in this tutorial, and the actual content of the files doesn't matter too much, as long as you can go in and make changes to the files during the tutorial so we can practice using Git to track those changes.

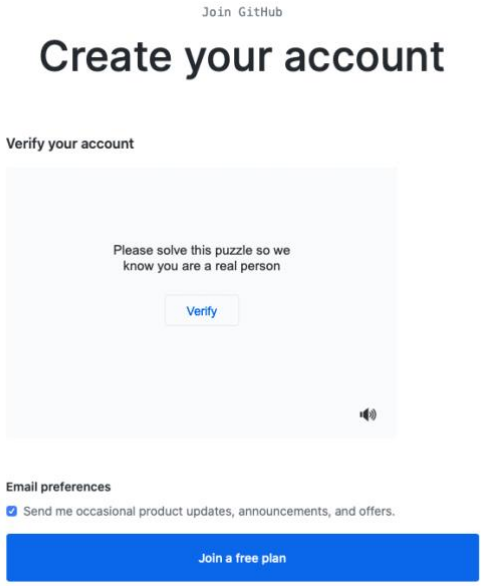
2.1 Online Providers

2.1.1 GitHub

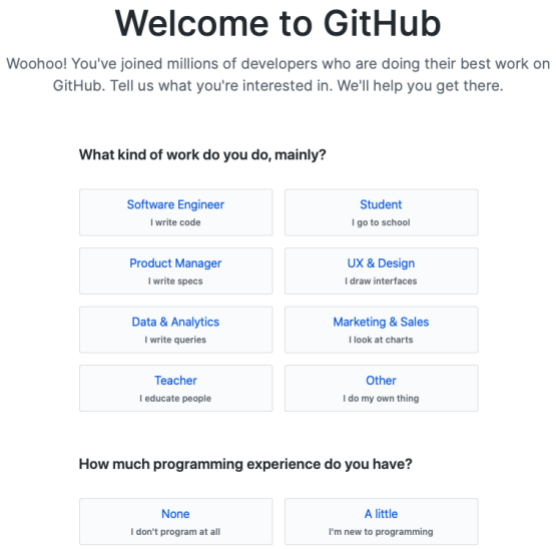
GitHub is currently the most commonly used online Git provider. They offer a simple interface and you can do many of the useful Git tasks completely online, without ever having to type code into your terminal. To get started, go to https://github.com and create an account. Type in your username and email, create a password, and then press the green "Sign up for GitHub" button.



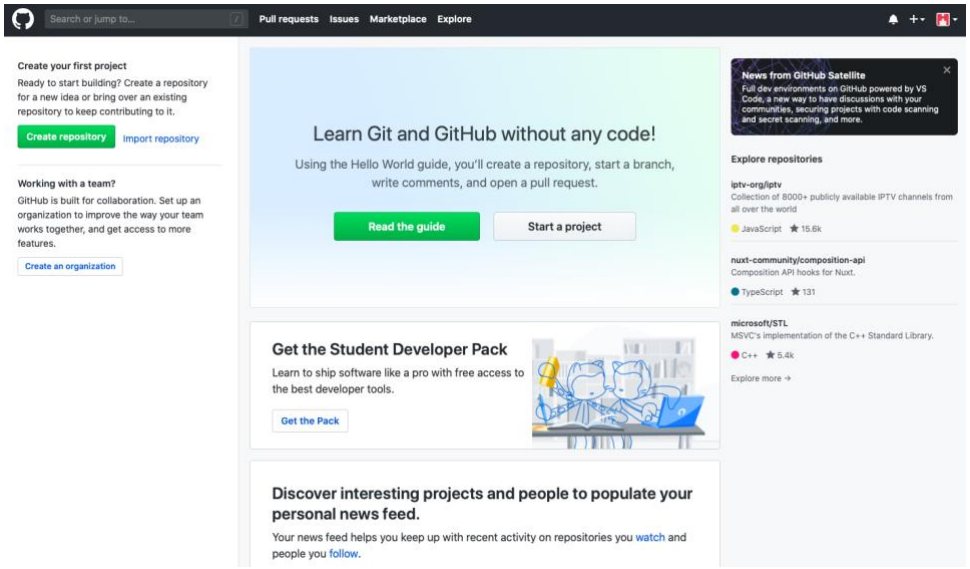
You may be asked to perform some task that verifies you are a human and not a robot, and then click the blue "Join a free plan" button.



You'll then be brought to a page with several questions about who you are and why you want to use GitHub. Go ahead and answer them appropriately and then click on the blue "Complete setup" button at the bottom of the screen.

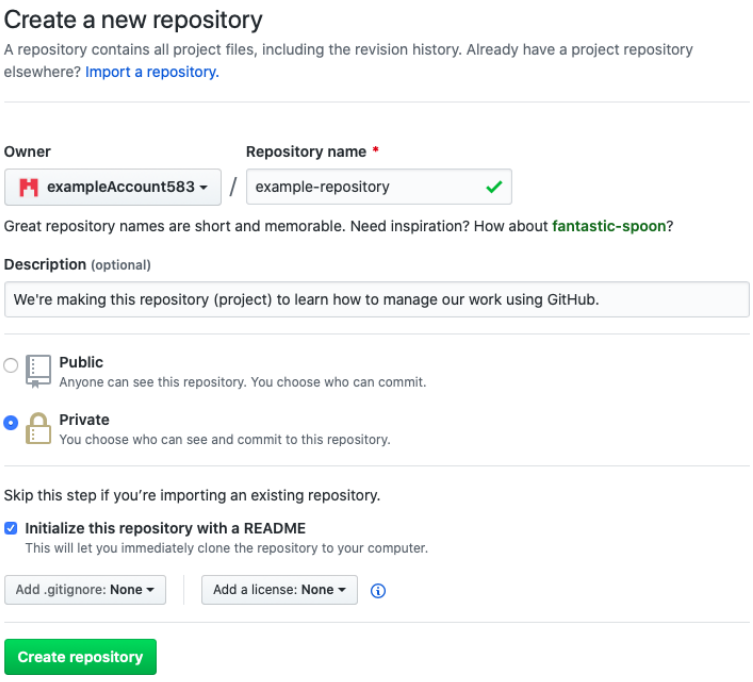


You'll then be greeted with the following window, which will likely be customized to your responses on the previous page, and your account has been successfully created!



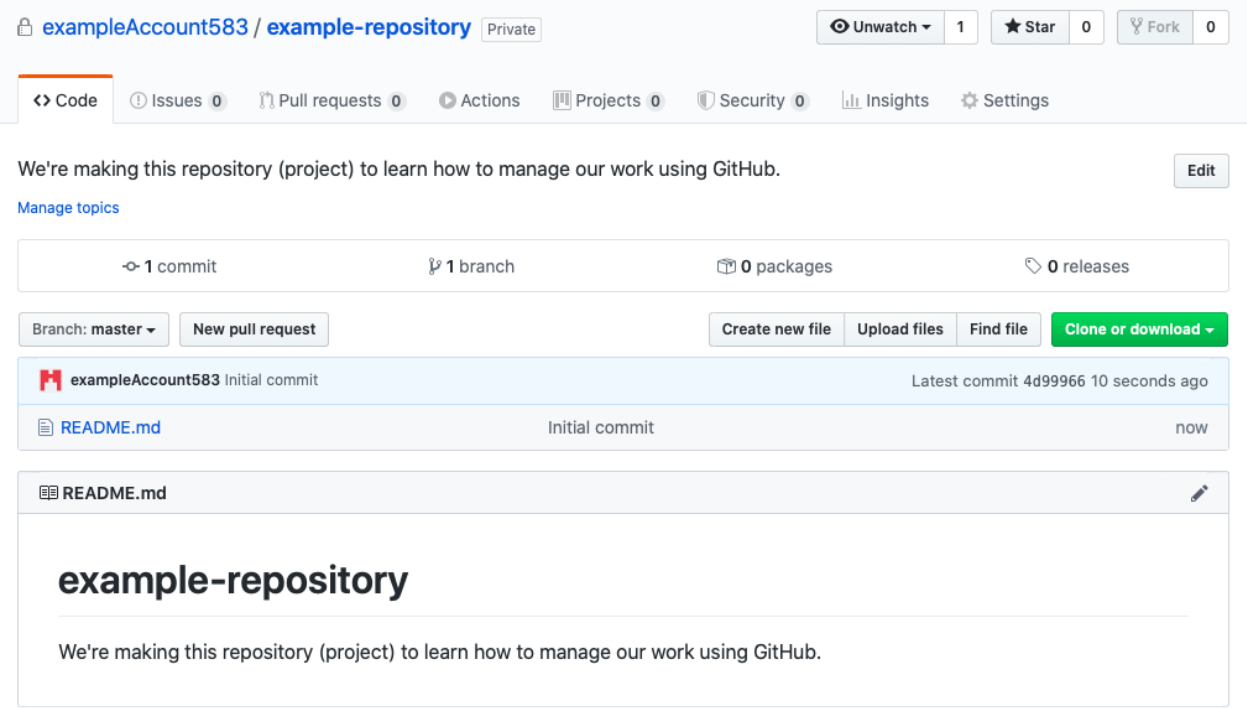
2.1.1.1 Creating a Repository

To create a repository, click on the green “Create repository” button on the left-hand side of the screen. If you’re using an account that already has repositories, the button to make a new repository will look different but will still be located in the same general area of the web page next to a list of your repositories.



You’ll be brought to the above page, where you enter some details about the repository you would like to make. In the “Repository name” field, enter a simple name that reflects the project you are making. Although this can be changed later down the road, try to choose a good name

now because changing the name can cause issues. In the “Description” field put a short description of what the project is about. Depending on your account permissions, you may not be able to choose between “Public” and “Private”, but if you can choose then you’ll most likely want to choose “Private” so that only you and people you allow can access the project. You will want to check the box that says, “Initialize this repository with a README”, which will take the description you wrote and display it with your repository so that anyone accessing your repository will immediately see a description of your project. For now, leave the “Add .gitignore” and “Add a license” boxes as they are by default. When finished, click the green “Create repository” button at the bottom of the page. You’ll be brought to the following new page:



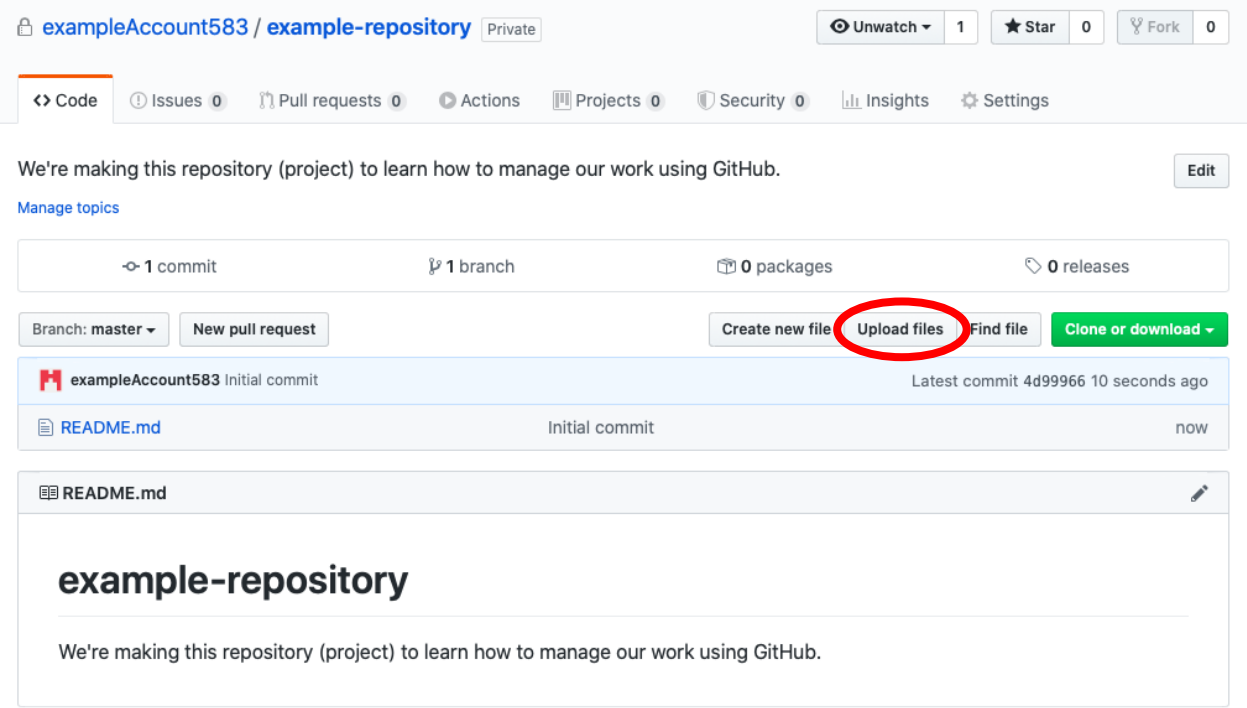
You have successfully created a Git repository using GitHub. You can upload files, see how your files have changed over time, and keep all of your work in a safe spot online. You may have some additional popups offering to help you learn about all the different options available to you on this page, explore them if you’d like or go ahead and dismiss them.

2.1.1.2 Uploading Files

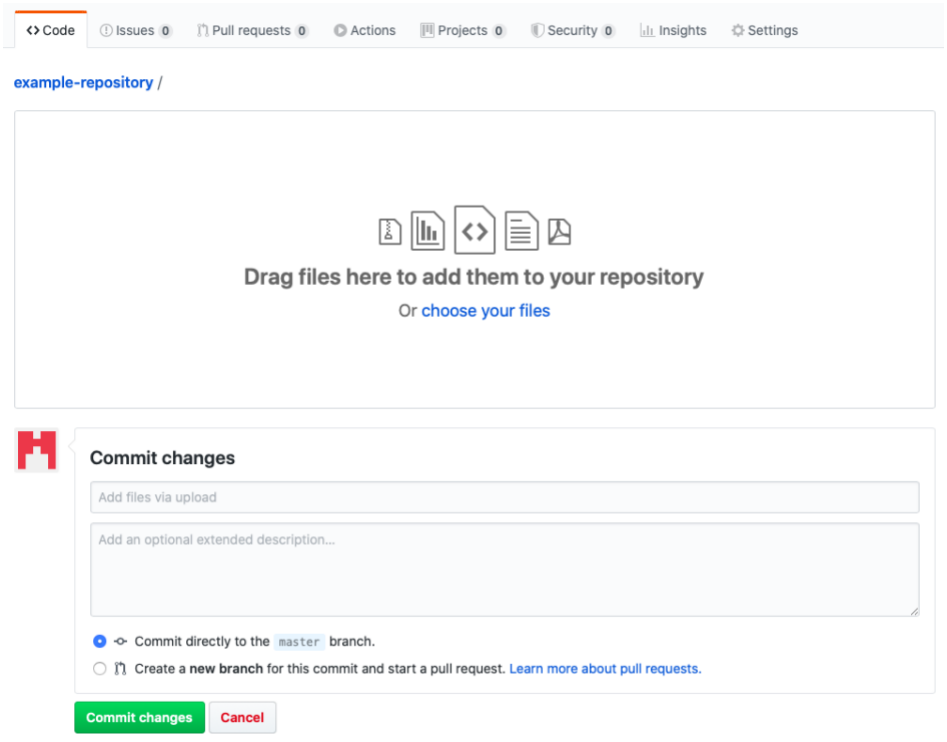
Let’s say that you have two files in your project that are saved on your computer:

some_math.m
important-notes.txt

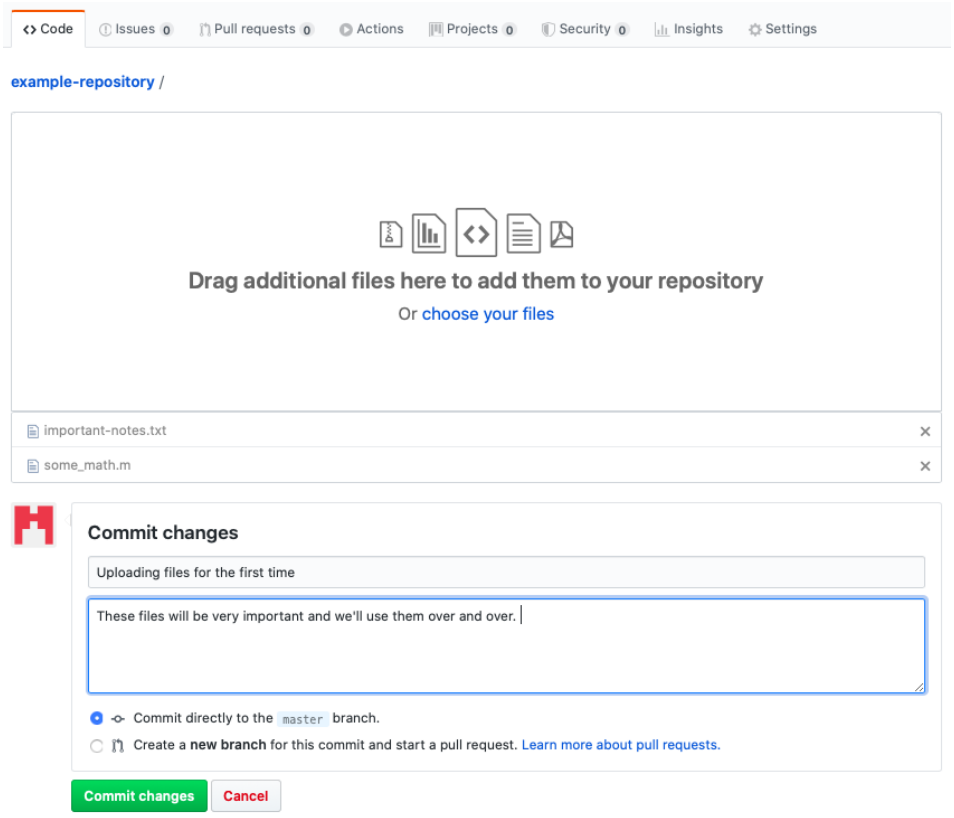
You would like to upload them to GitHub so that you can track all of the changes that you make to them over time and maybe someday collaborate on the files with another user. To upload these files, click on the “Upload files” button.



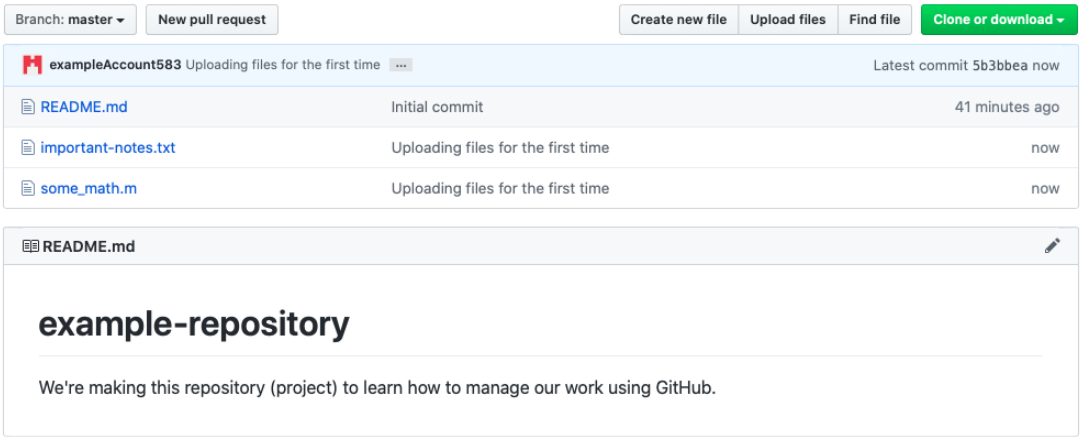
You will be brought to this page, which understandably may look rather frightening:



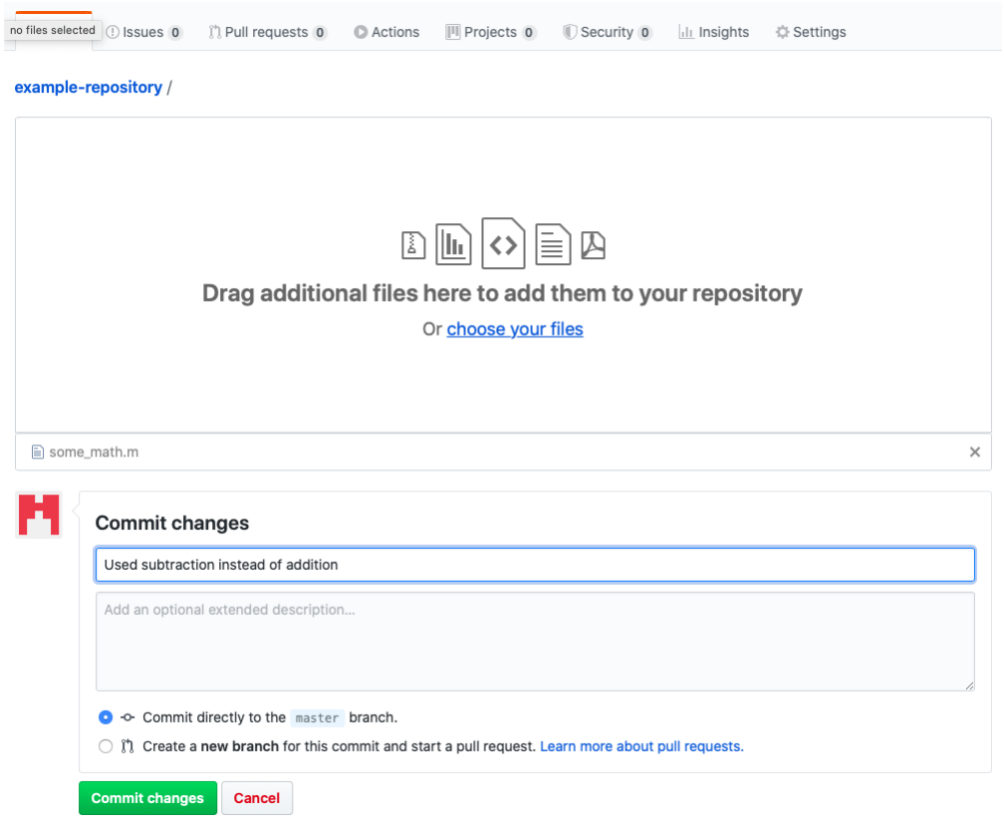
Select the blue “choose your files” text to add files. Then you will need to add a commit message describing the changes that you are making to the project. If your message is too long you will be prompted to put additional details in the larger text box so that as needed other people can see your message in full detail. Go ahead and leave other options as they are.



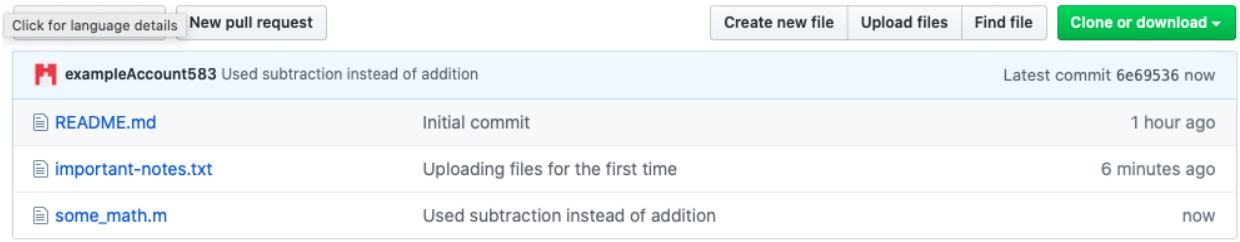
When you’re ready, click the green “Commit changes” button to save these to your repository. After this, your repository will look like this:



Now, let's say that you have made changes to some_math.m on your computer, and you would like to save those changes on GitHub too. Again, click on "Upload files" and upload the new version of the file from your computer, like so:



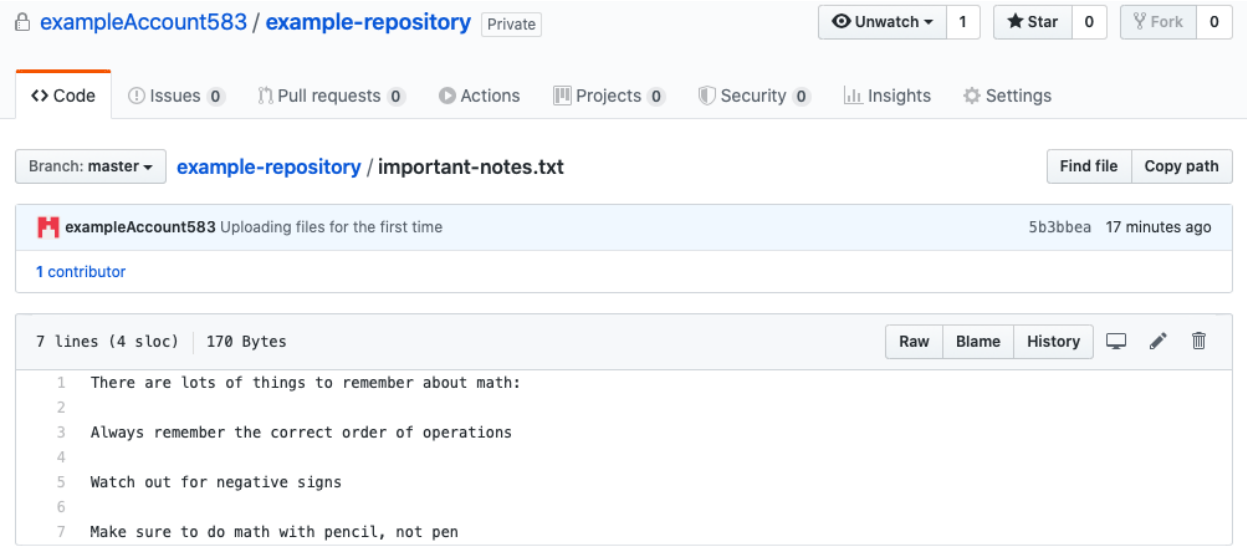
Go ahead and commit those changes using the green button and when you look at your repository you will see that the new file was uploaded. Notice that the most recent commit message for each file is shown along with how long ago it was committed.



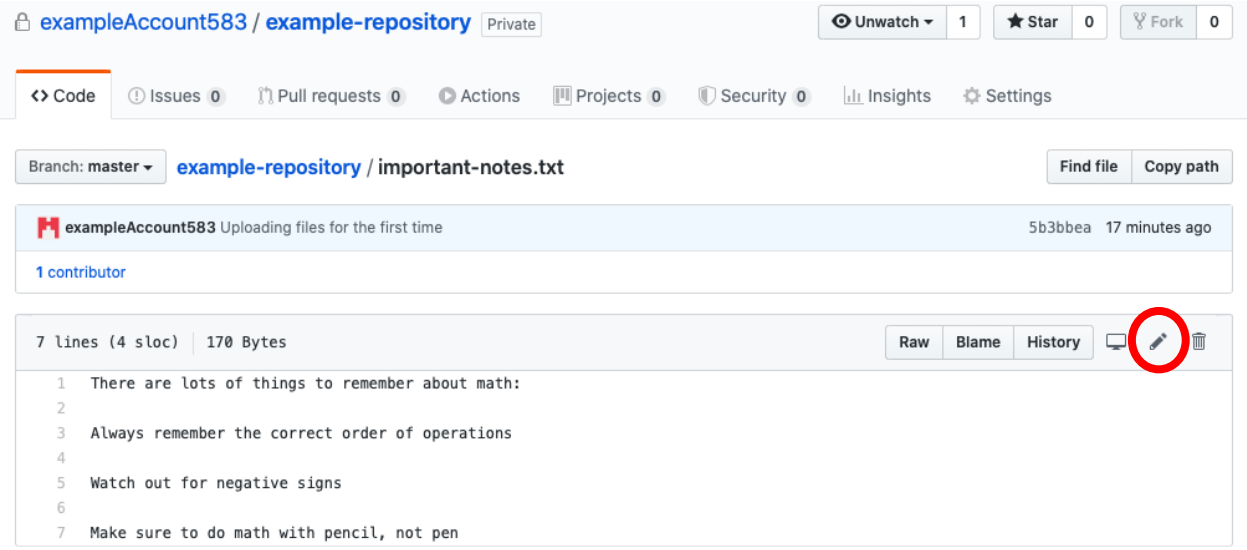
2.1.1.3 Editing Files Online

Although it is generally bad practice, you can actually edit files on GitHub's website. This is risky because it means that the files online are changed in ways that the files on your computer aren't. Better practice would be to make changes on your computer and then upload the new files, as shown in the previous section. Nevertheless, should you ever find yourself in an appropriate situation to edit files online, here's how to do it.

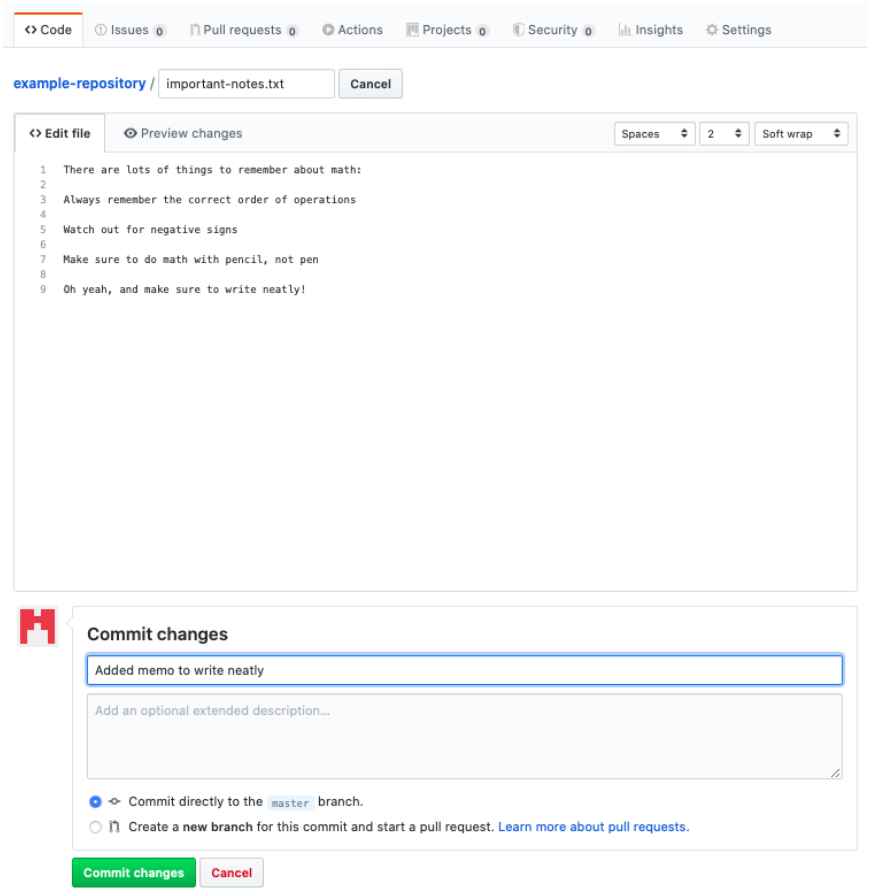
Click on one of your files to open it. For example, after clicking on the “important-notes.txt” file you’ve that uploaded you’ll see the contents of the file:



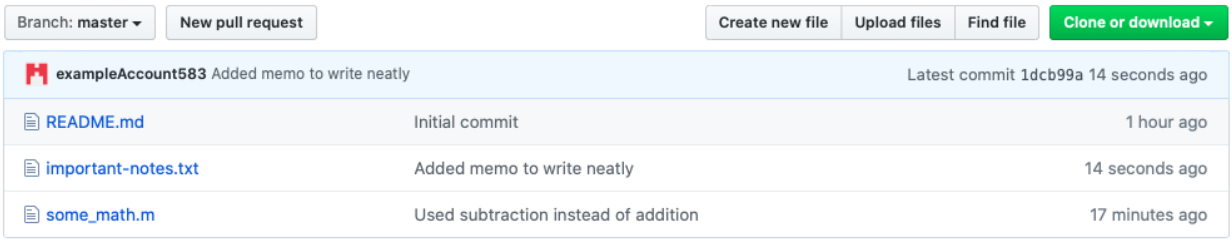
To edit the file, click on the pencil icon on the right-hand side of the page:



Make any desired changes, add a commit message, and then press the green “Commit changes” button.

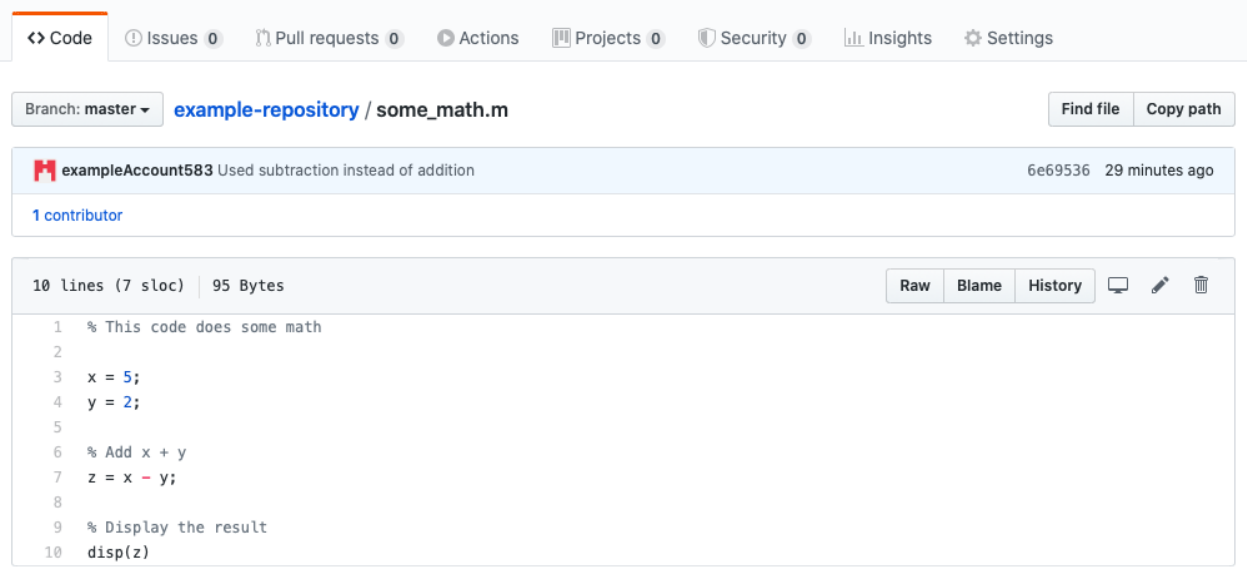


You will again see the changes you've made when looking at your repository:

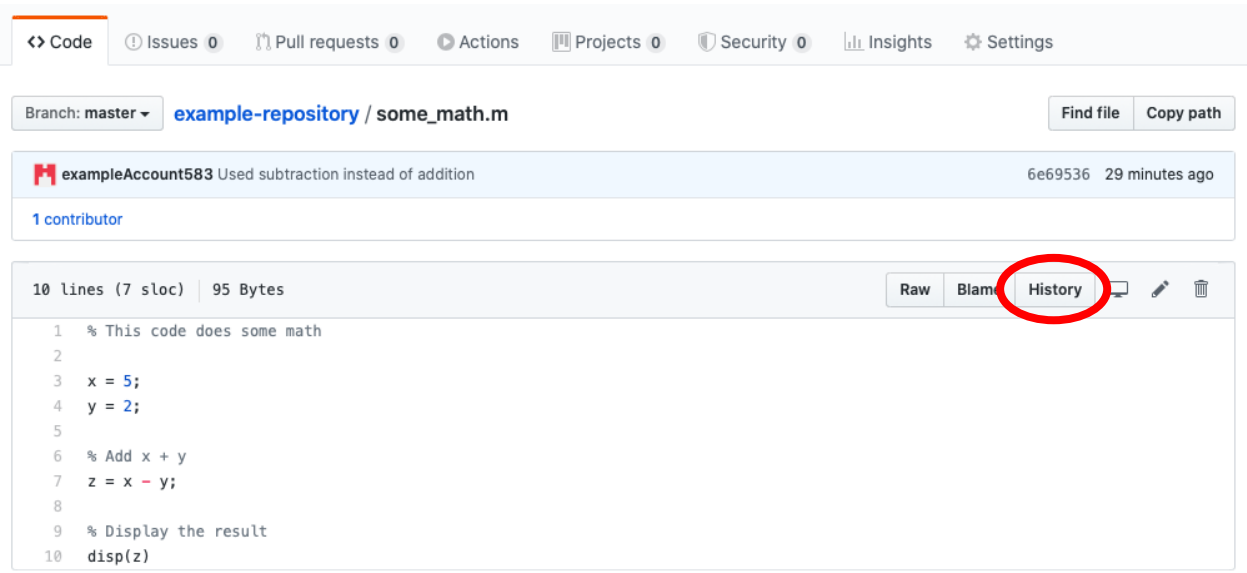


2.1.1.4 File History and Comparing File Versions

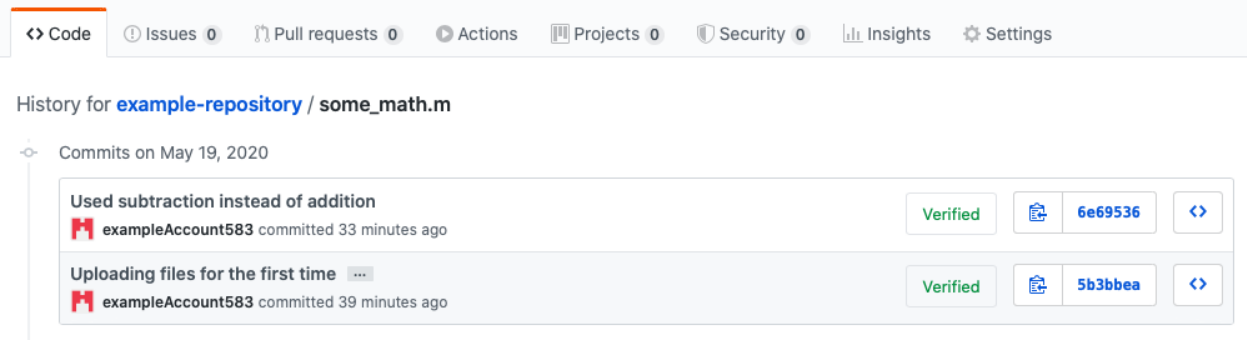
One of the most important Git benefits is the ability to look at and even compare old versions of your files. If you want to look at the history of a file, click on it in your repository. Doing this for "some_math.m" leads to the following page:



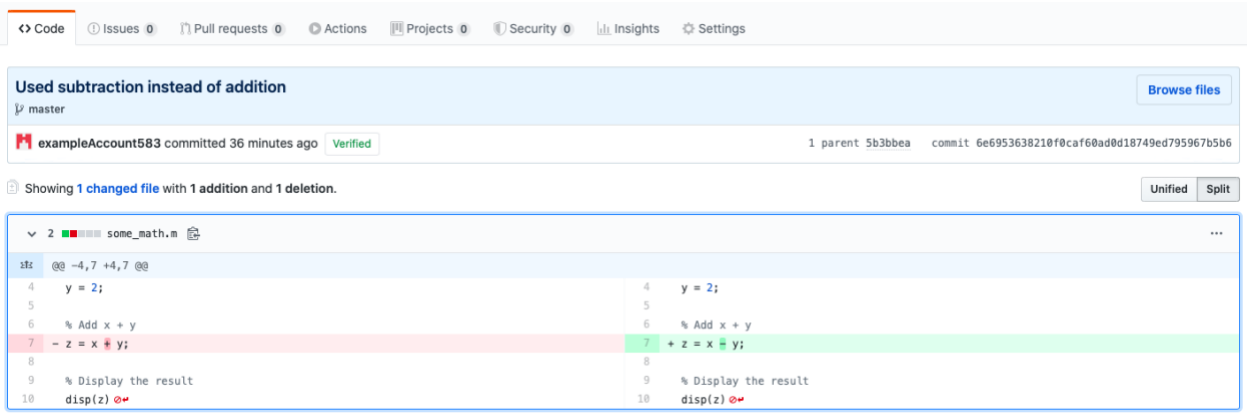
If you want to see the file history, click on the “History” button:



This will bring you to a page where you can see the history of this particular file. We see here that this file has changed twice (being created technically counts as a change).

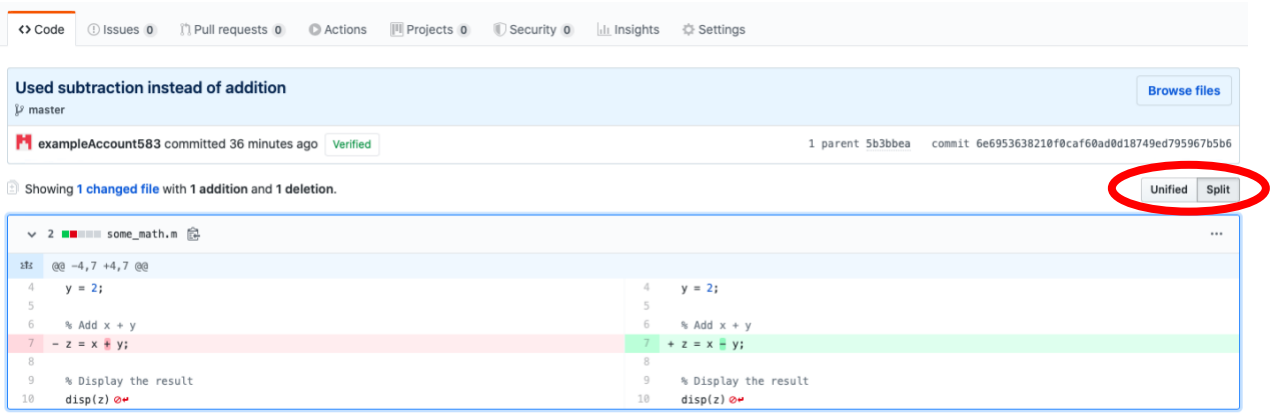


Click on one of the file versions to compare it with the version that came before it. This will bring you to a page that looks something like this:



The version of the file on the left-hand side of the page is the older version and the version on the right-hand side of the page is the newer version. You can see here that in this example the only change between the two files was turning the addition symbol into a subtraction symbol, as described in the commit message.

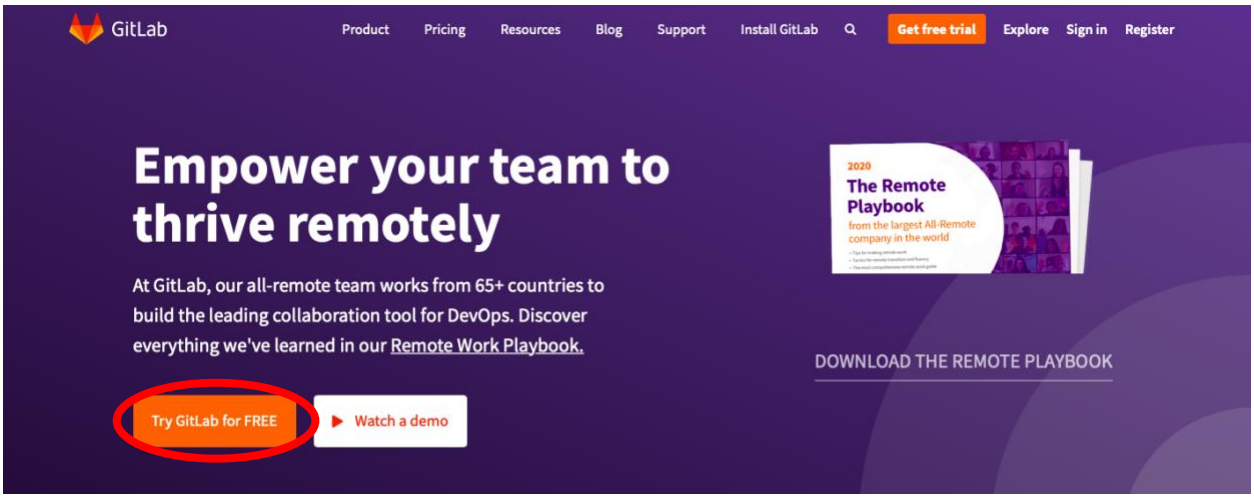
The files can either be viewed in “Unified” or “Split” mode. To switch between these views, click on the buttons on the right-hand side of the page:



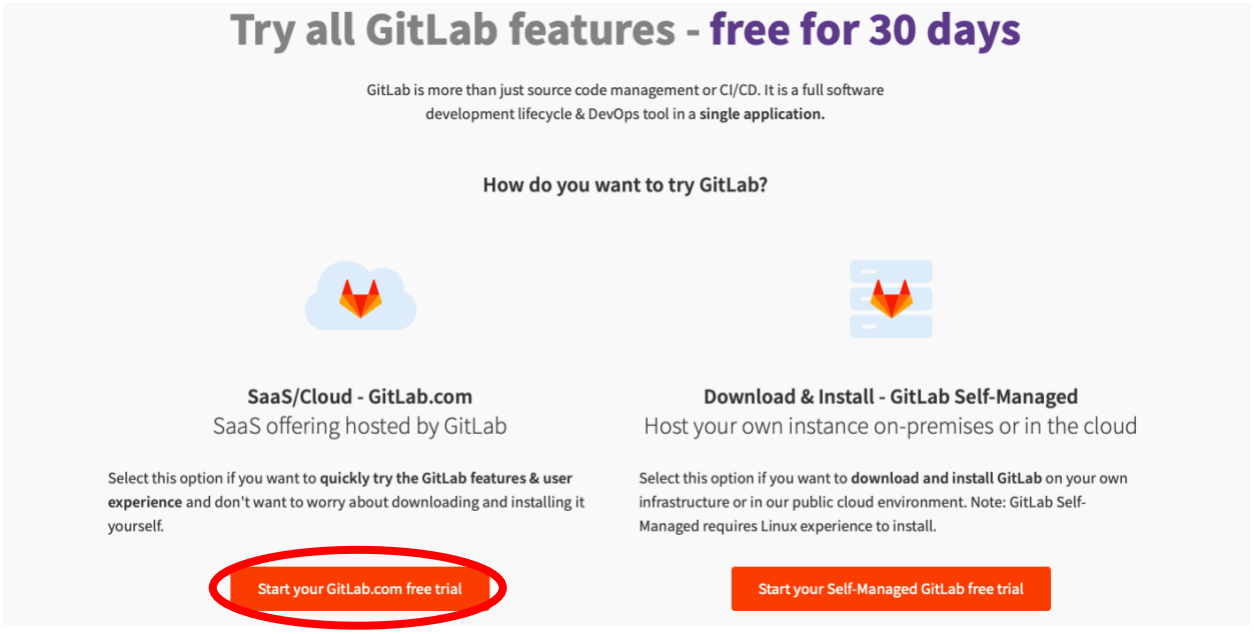
There you go! You can now begin to use GitHub to store your files and help keep your work clean. There are so many other things you can do with GitHub besides what is covered here. Refer to Section 5.3.2 in the Appendix for more helpful GitHub resources.

2.1.2 GitLab

GitLab is an online Git provider similar to GitHub. You can access many of the key Git features without ever typing code into your terminal. To create an account on GitLab for yourself[5], go to https://about.gitlab.com, and click on the "Try GitLab for FREE" icon.



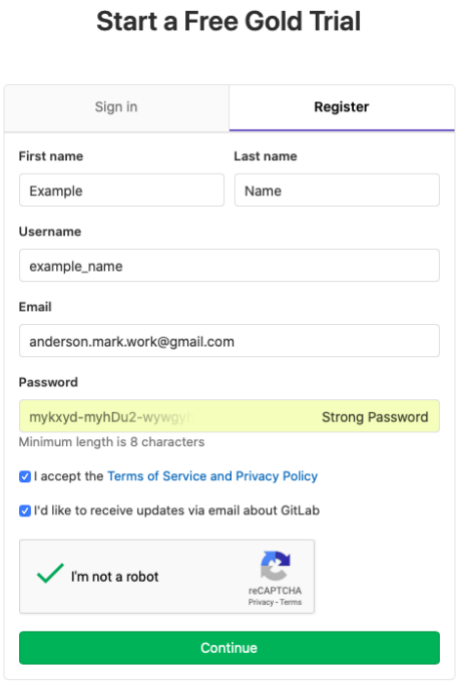
This will bring you to a page where you select which version you want to install. You can choose to download GitLab onto your computer, but for now we will choose to keep it online.



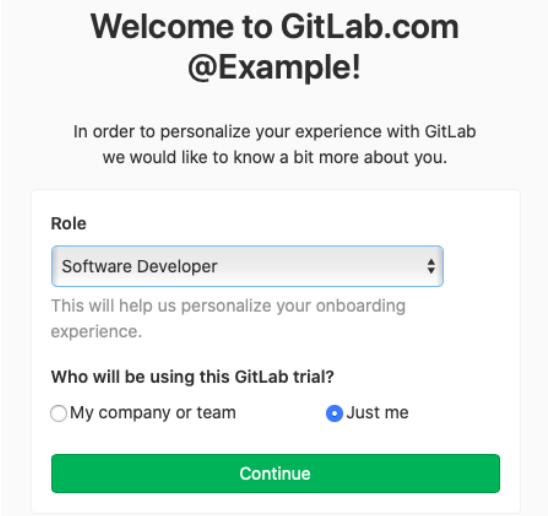
You can ignore the notices about losing functionality after 30 days because we will choose the very functional free version. After clicking on the "Start your GitLab.com free trial" icon indicated

[5] BYU physics students already have a GitLab account on http://git.physics.byu.edu and can sign in with their netID and password and can do this tutorial using that account if desired.

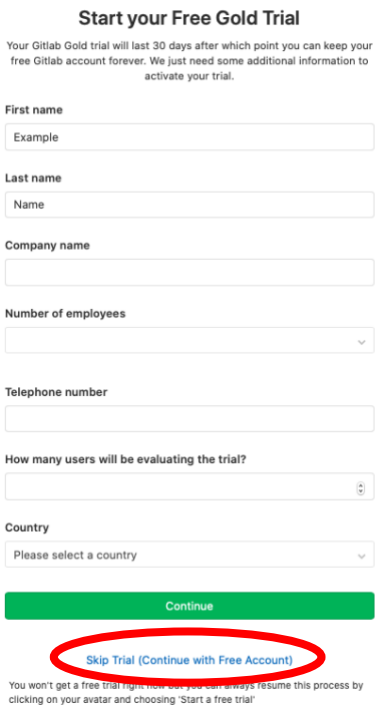
above, you will be led to a page to either sign in or register. If you do not have an account yet, then click on the “Register” tab. Fill in the information appropriately and then click on “Continue”.



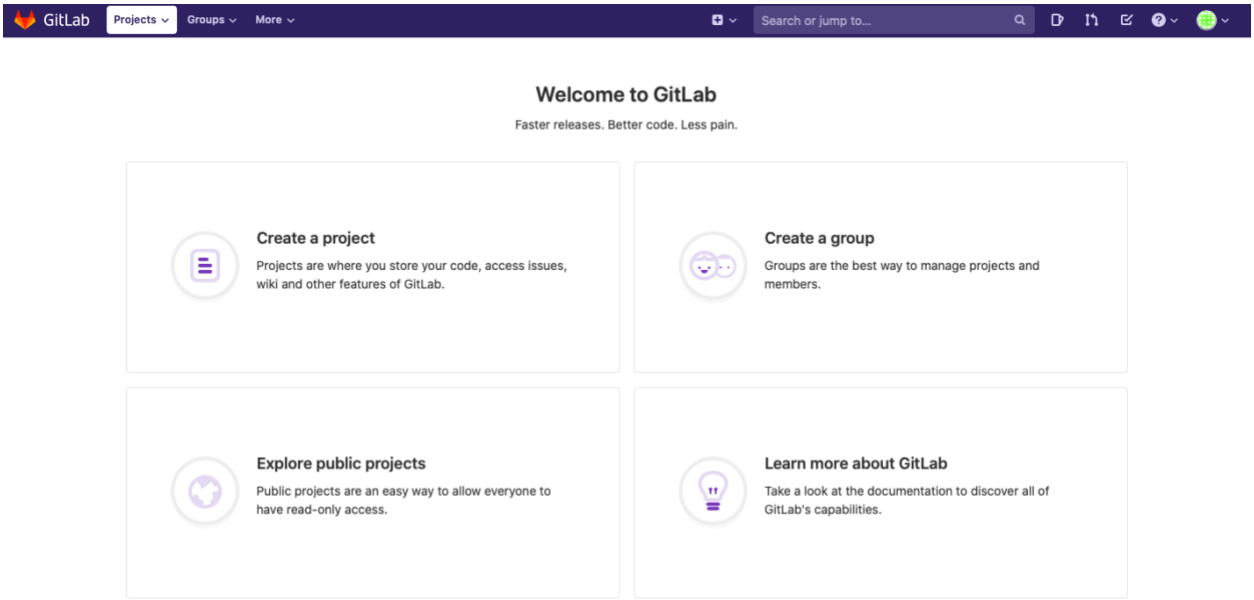
You will be directed to a page where they ask some information about you and why you’re using GitLab. Fill this out appropriately and then click “Continue”.



On the next page they will ask you to fill out some more information about yourself as part of the “Gold” level free trial. You may fill this out or skip the “Gold” version free trial and just use a free account by clicking the blue text at the bottom of the page. This tutorial uses the free account, so everything shown here is available for free to any user.

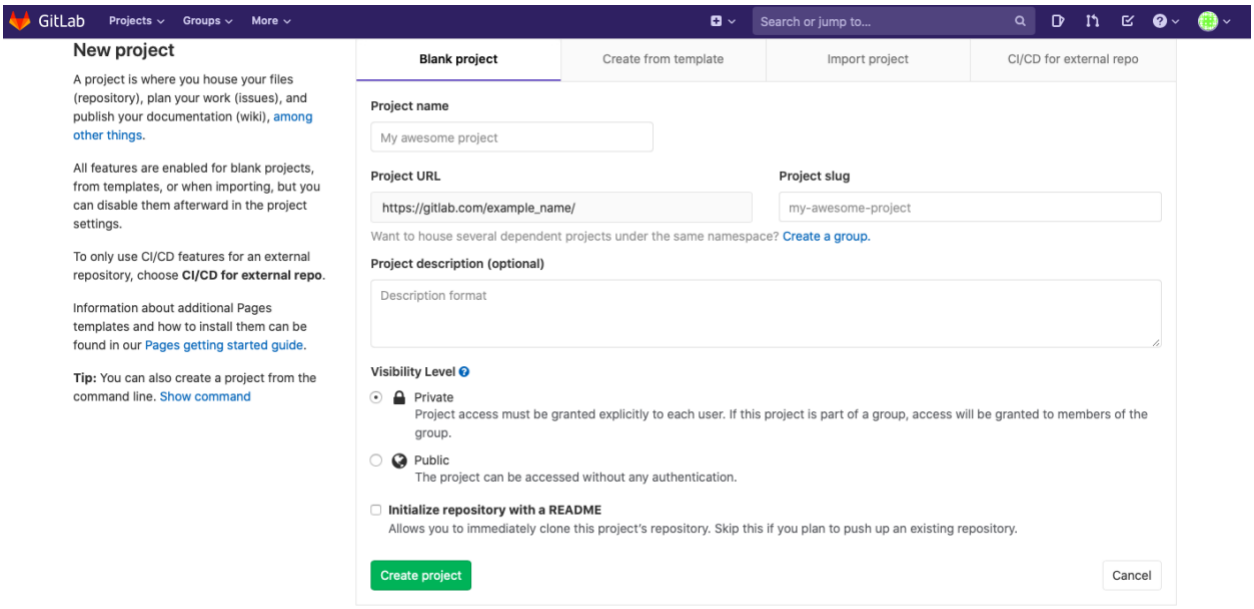


You will be redirected to the following page, and your account has been created.



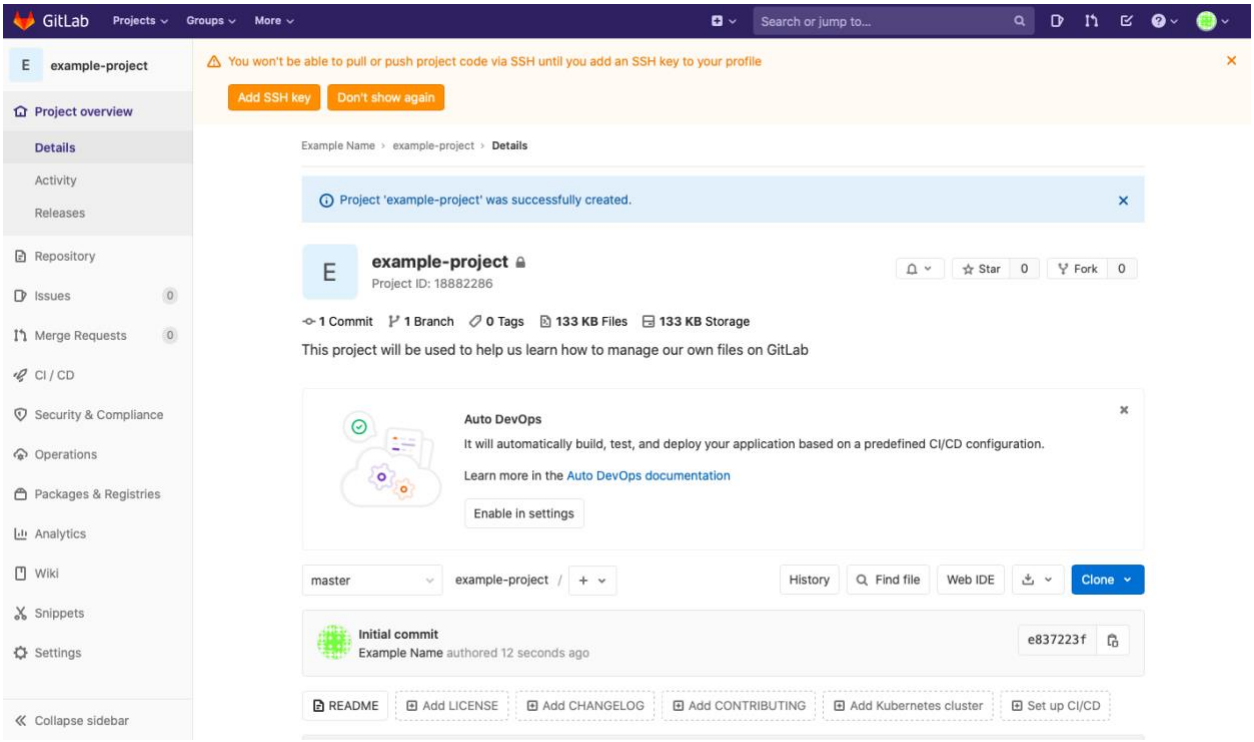
2.1.2.1 Creating a Repository

In GitLab, repositories are called “projects”, and we will use the word “projects” in this tutorial. Click on the “Create a project” button to create a new project. You will be directed to the following page:

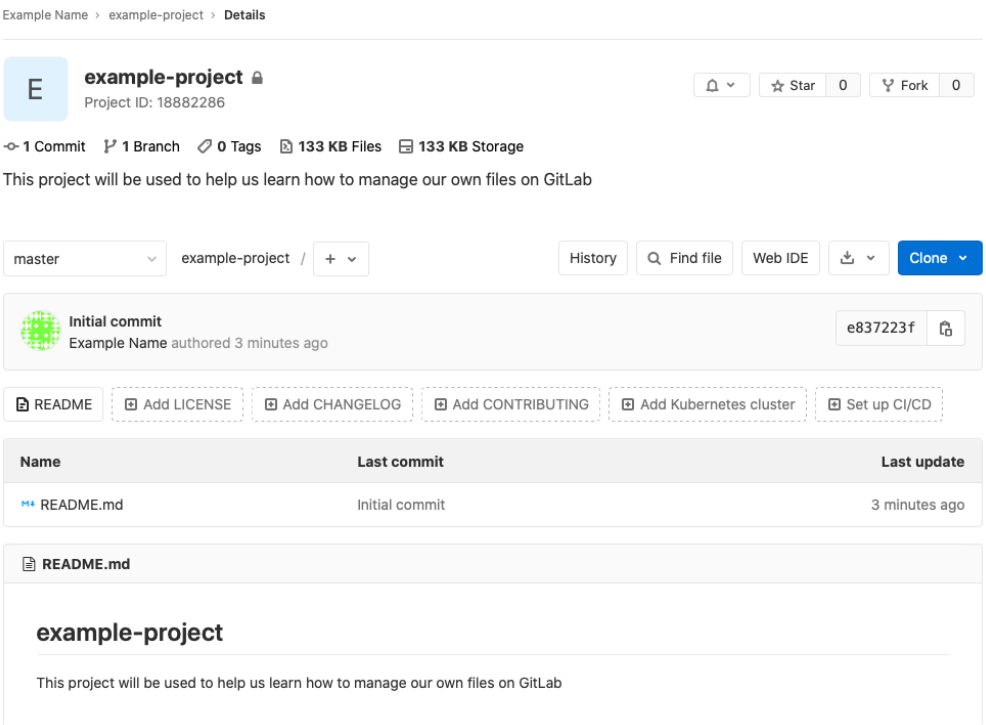


Fill out the boxes. The “Project name” box is where you officially name the project. Make sure to choose a name that is short and sweet, but descriptive enough that nobody will question what it is. The “Project slug” box will be filled out automatically according to the “Project name” box but will replace any spaces with dashes. For simplicity, it is often desirable to give your project a name that does not have any spaces so it will automatically be the same as the project slug. The “Project description” box is a place to include more detail about the project and should be filled out each time you make a new project. GitLab defaults to making the project private, so that only you and people you give permission to can see the project. Lastly, you want to check the “Initialize repository with a README” box. This will provide people viewing your project with a description of the project when they view it. When you’re ready, click on the green “Create project” box.

You will see a page similar to the following:



The dialogue box at the top that talks about SSH keys can be ignored for now. Click on “Don’t show again” (this feature can be enabled later but not needed for this tutorial). After exiting from the blue dialogue box and the “Auto DevOps” dialogue box, you will see the following:



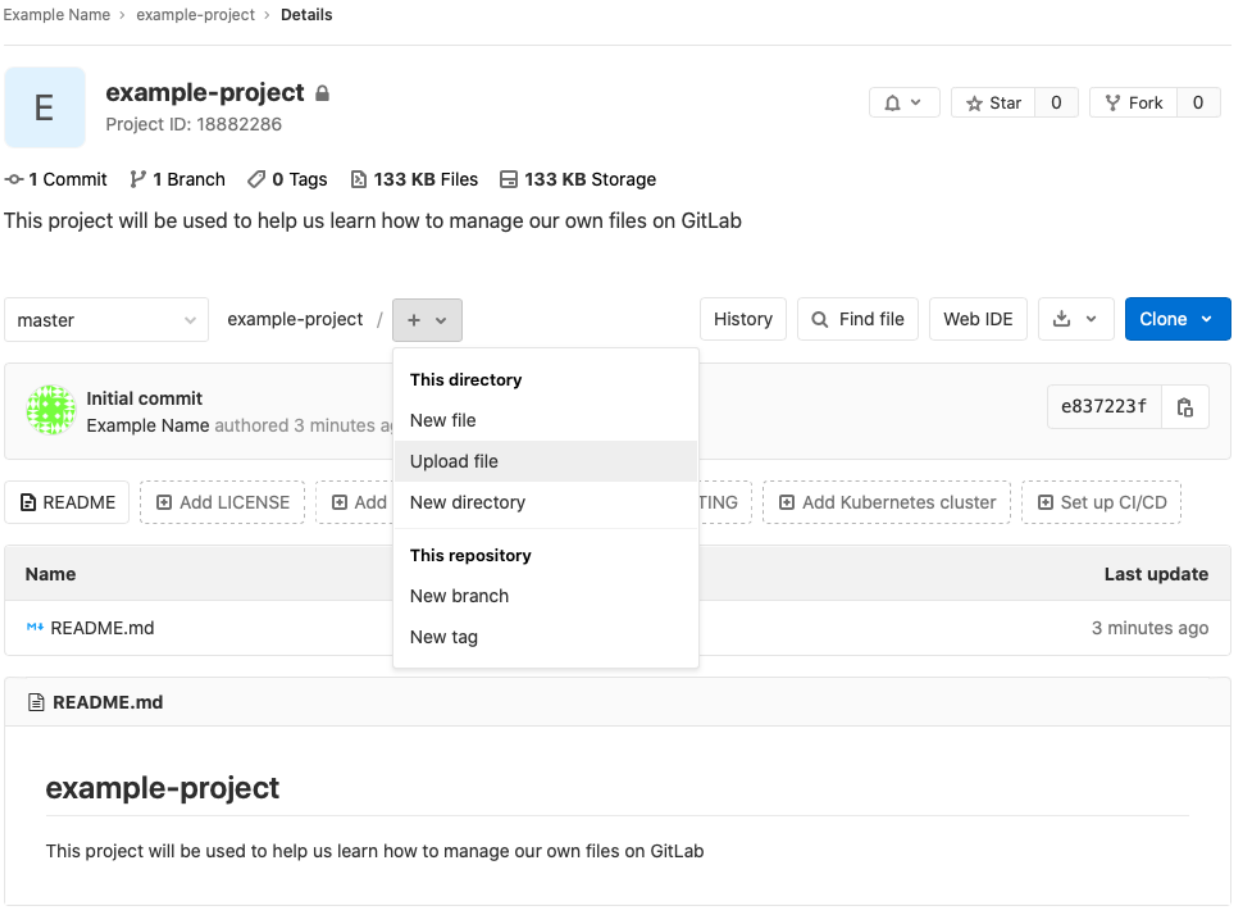
You have now successfully created a project on GitLab. This will be a place to store your files and where you can see different versions of your file over time. Eventually, you can also add others to the project so you can work in a team.

2.1.2.2 Uploading Files

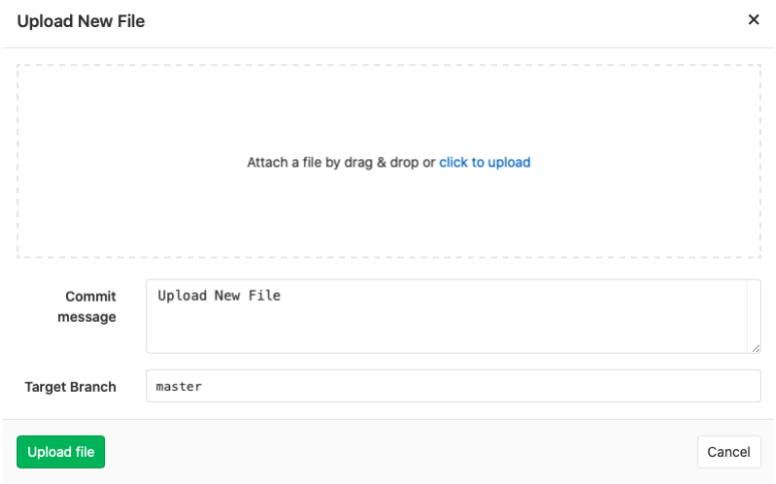
Let's say that you have two files in your project that are saved on your computer:

some_math.m
important-notes.txt

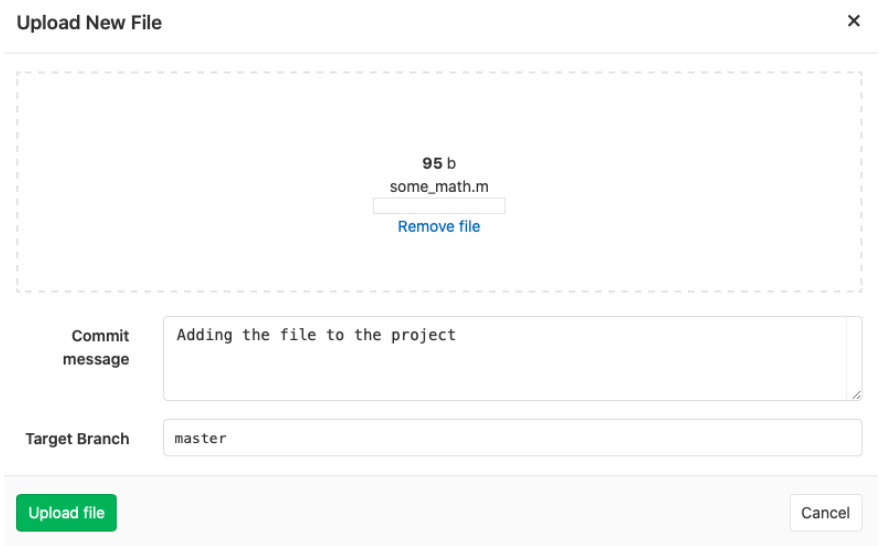
You would like to upload them to GitLab so that you can track all of the changes that you make to them over time and maybe someday collaborate with others on making these files better. To upload these files, click on the plus sign icon and select "Upload file":



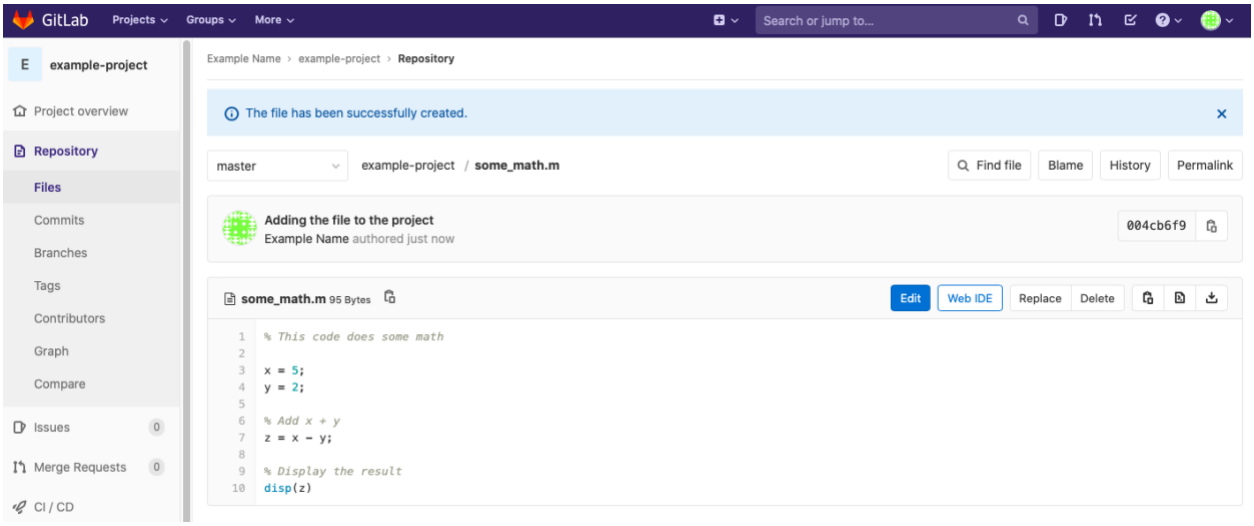
The following box will appear:



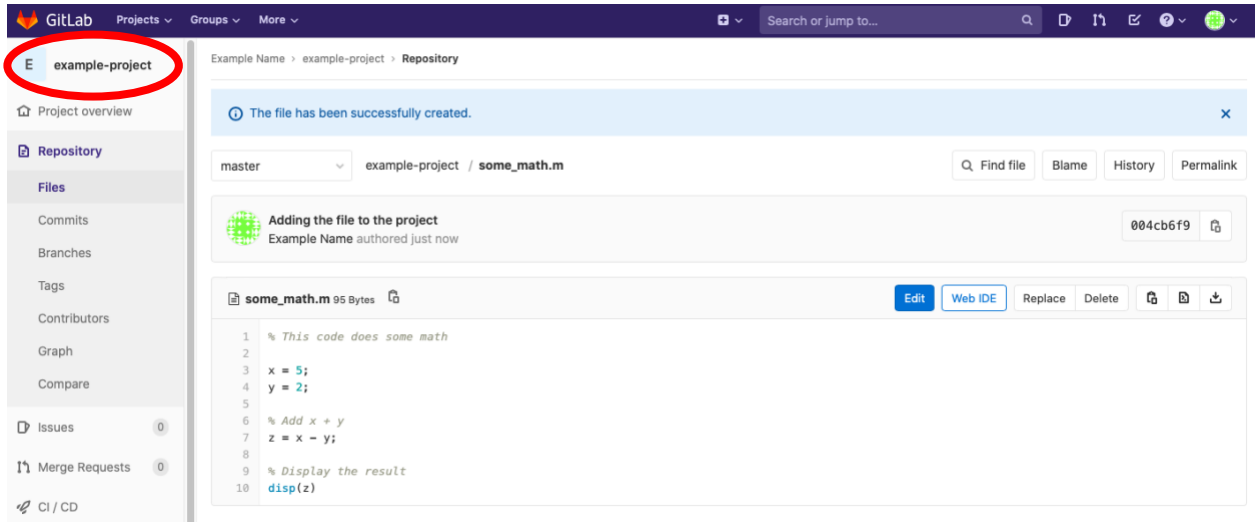
Click on the blue “click to upload” text and select a file you would like to upload (file uploads must occur one at a time, but you can add an entire folder at once if you want). Then type a message in the “Commit message” box. This message will appear next to these files in the project so make sure that it’s a short yet descriptive message. Click on the green “Upload file” icon to complete the upload. Do this for all files that you want to add to the project.



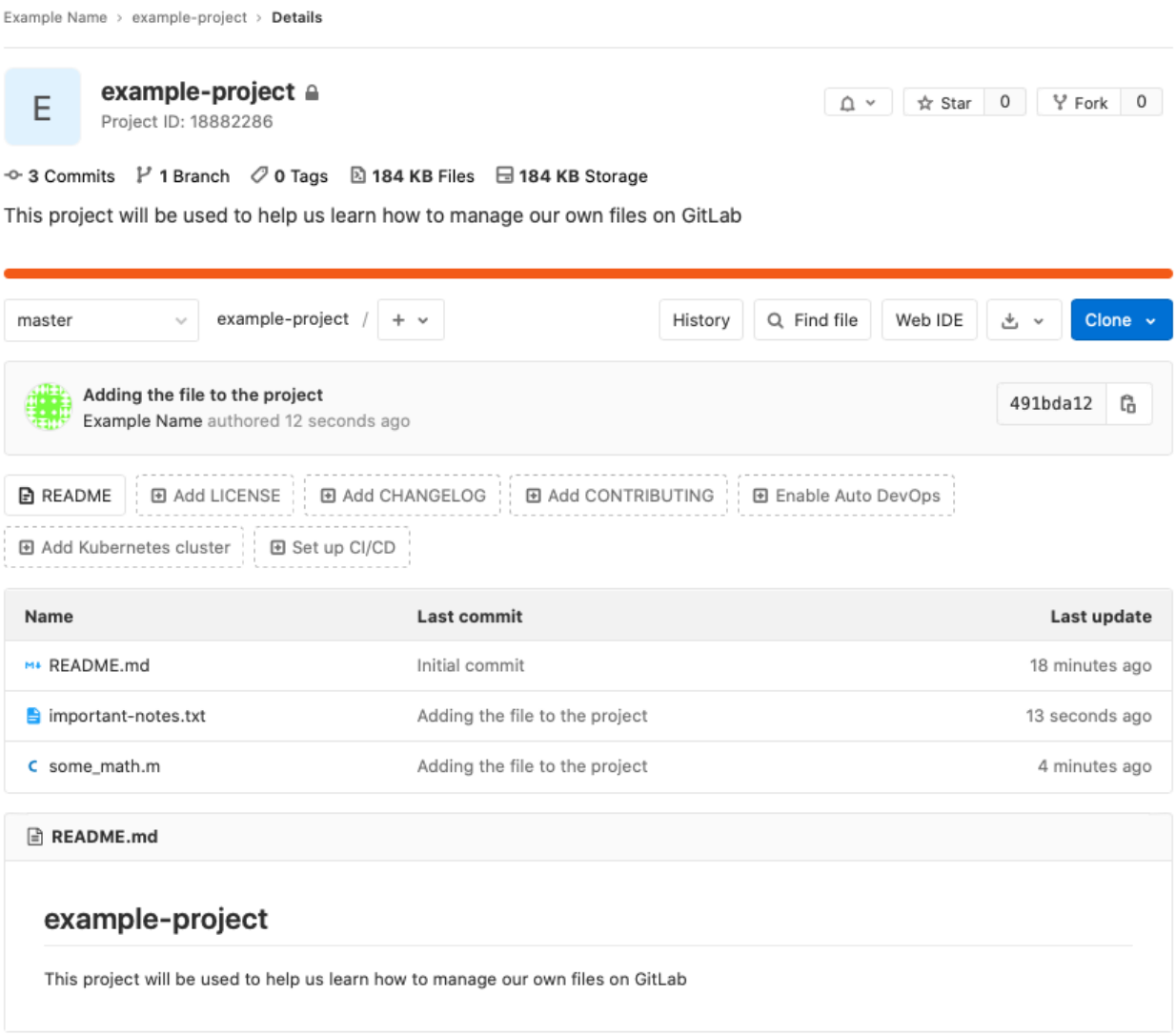
GitLab might open the file immediately and you may see a screen that shows the contents of the file, like this:



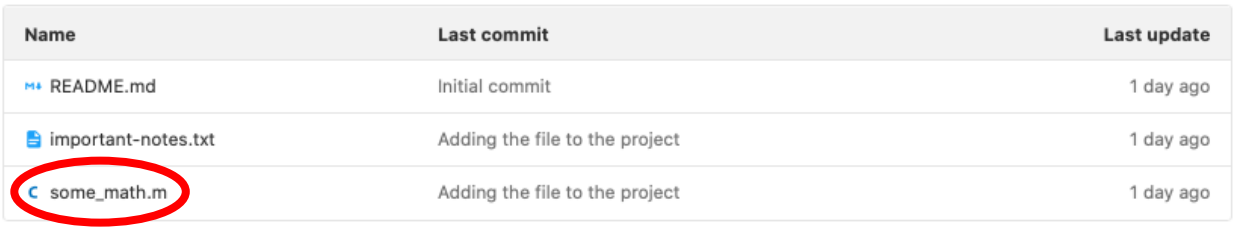
To return to the main project page, click the project's icon in the upper left corner of the web page:



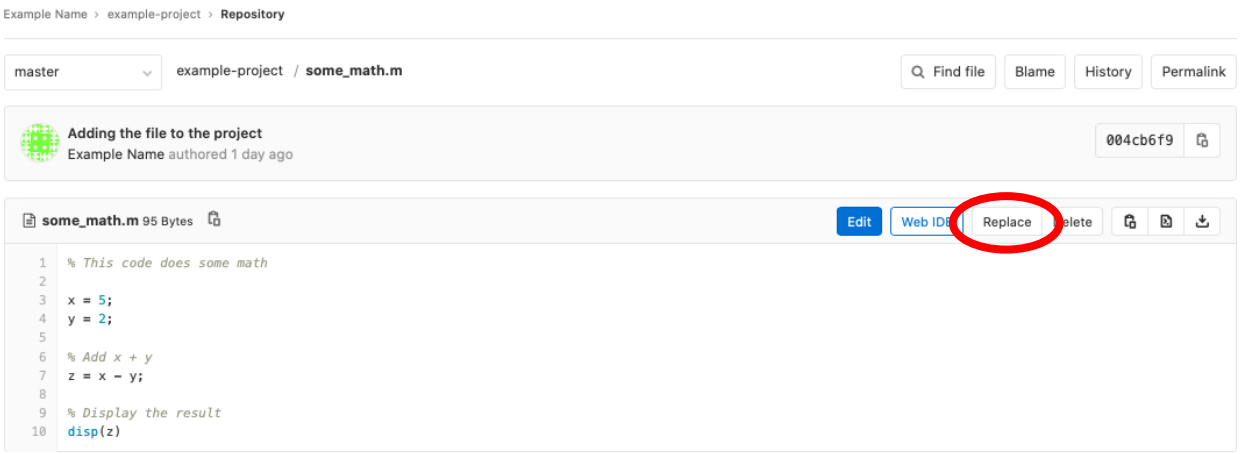
Once your files are added, the project's home page will reflect those additions:



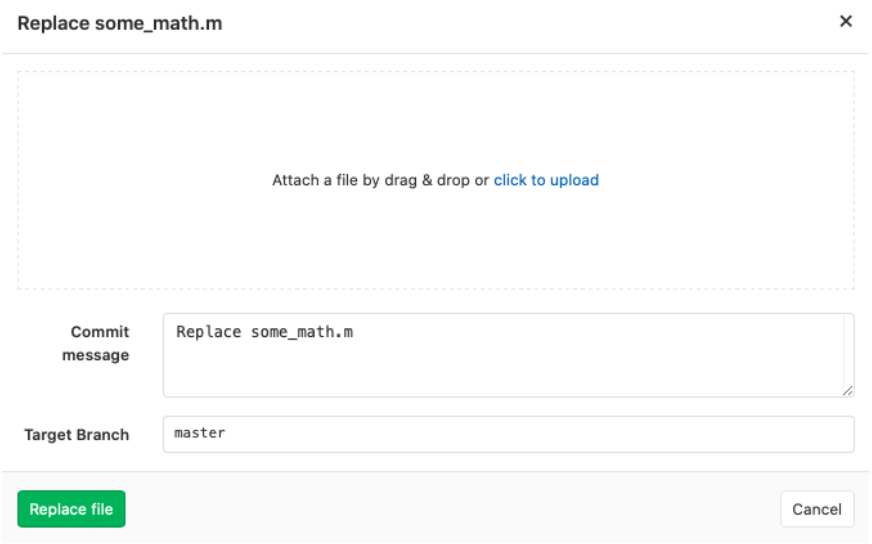
Now let's say that you have changed some_math.m and would like to store the new version on GitLab. Click on the file you would like to update:



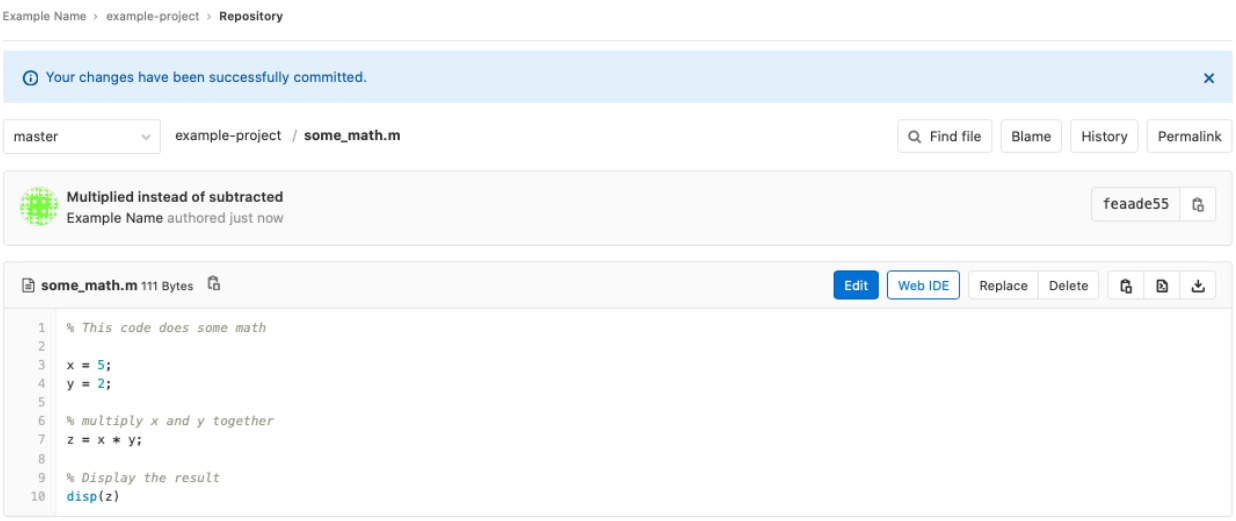
You will be directed to a page that shows you the contents of the file. Click on the "Replace" icon.



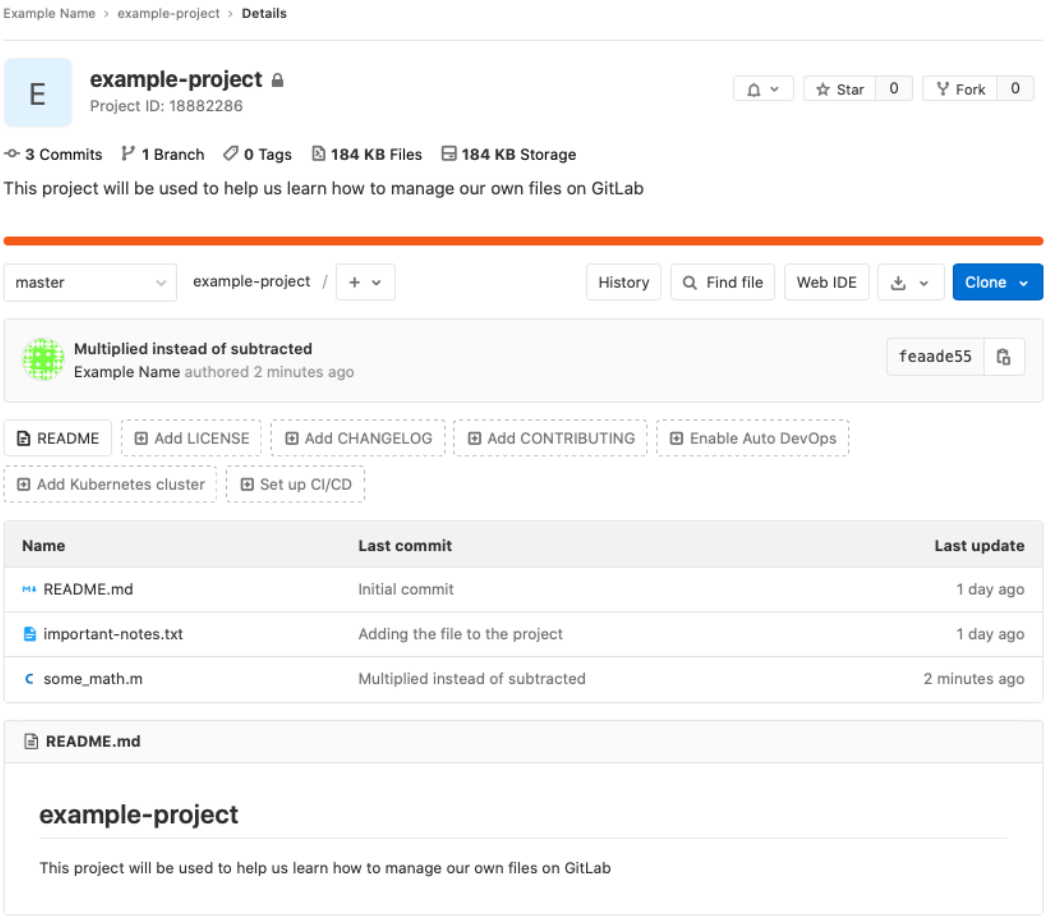
The following box will appear:



Click the blue text to upload the new version of the file, write a simple commit message, and then click on “Replace file”. You will then see a page that shows the new version of the file:



Return to the main project page, and you will see the updated file with its commit message:

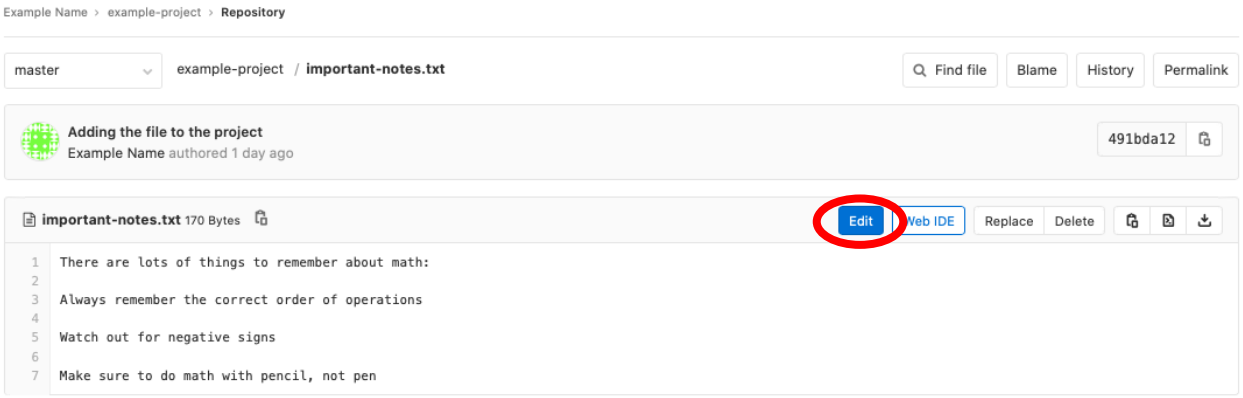


Congratulations, you can now upload files and new file versions to GitLab. This will help you to keep your files safely stored online.

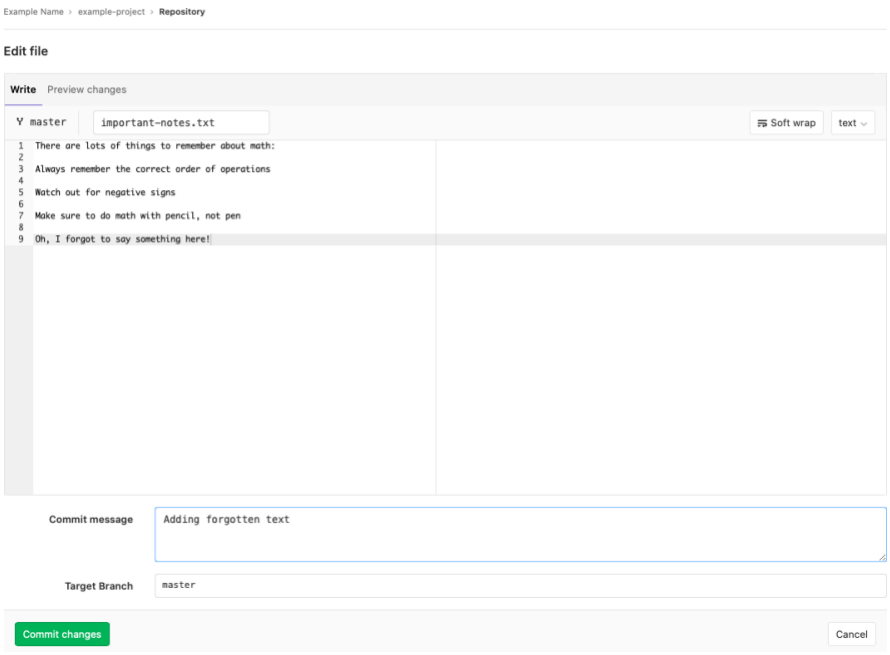
2.1.2.3 Editing Files Online

Although editing files online is generally a risky practice, it is possible. It is risky because it means that the files on your computer will not have those same changes, and then if you try to upload a new version from your computer the changes you made online will disappear. However, if you find yourself in a situation where it is appropriate to edit a file online, this is how you do it.

Click on the file that you would like to edit; in this example we'll edit the important-text.txt file. This will take you to a page that displays the contents of the file. Once here, click on the blue "Edit" button.



Make some changes, and then type an informative commit message before clicking on the green "Commit changes" button.



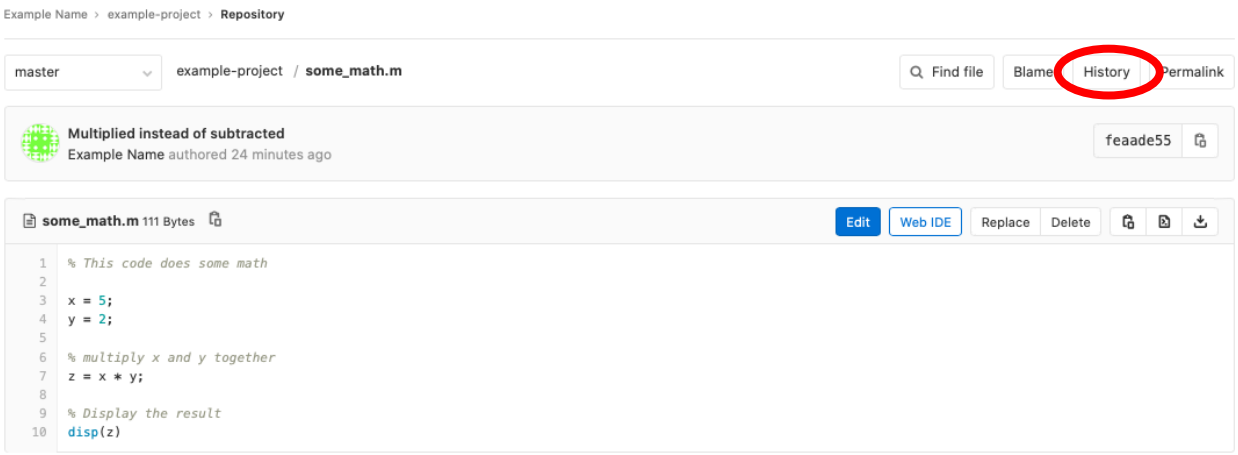
Navigating back to the main project page, you will see the updates:

Name	Last commit	Last update
README.md	Initial commit	1 day ago
important-notes.txt	Adding forgotten text	just now
some_math.m	Multiplied instead of subtracted	15 minutes ago

You can now make edits purely online.

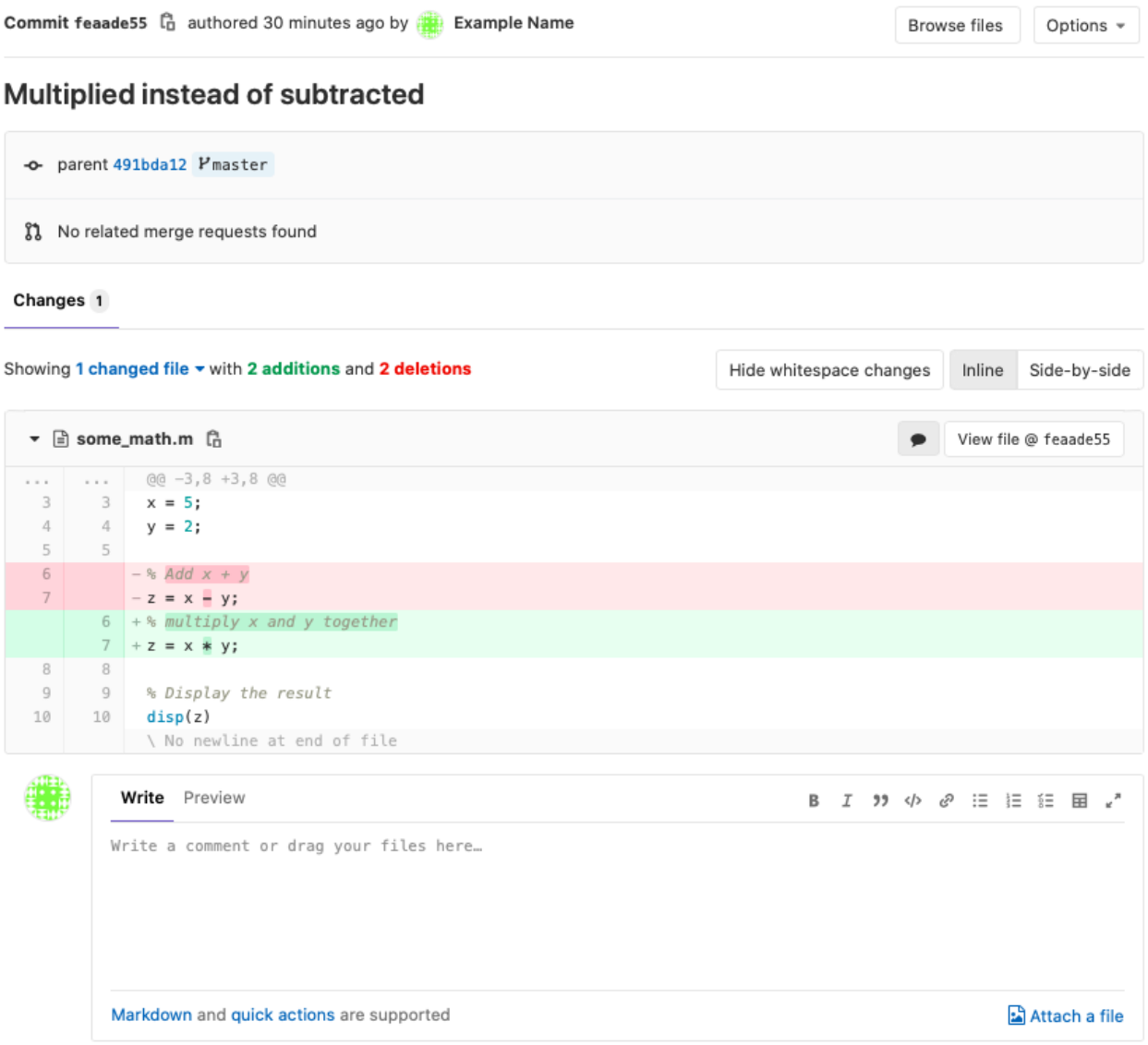
2.1.2.4 File History and Comparing File Versions

Now let's say that you want to see how your files have changed over time. This is useful if some recent changes you made have introduced new bugs or if you want to see when certain changes were made. To do this, first click on a file in the main project page to view its contents. This will lead you to the following page, where you will then click on the "History" icon:



You will see a page with the different versions of that file. Click on one of them.

You will now see a page that shows you the differences between the files. If you want to see the differences side-by-side, then click on the "side-by-side" icon on the right-hand side of the page.

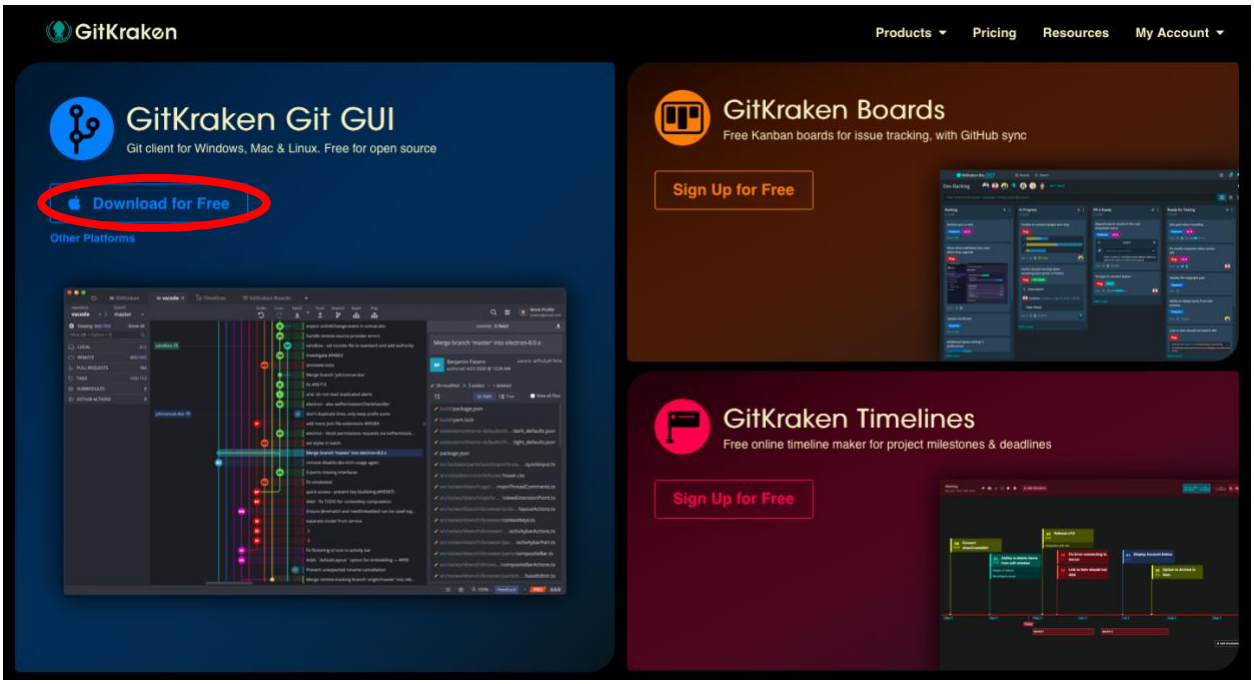


And there you go; you can now begin using GitLab for basic version control with your files. Read Section 3.3 for more information about working in groups using GitLab.

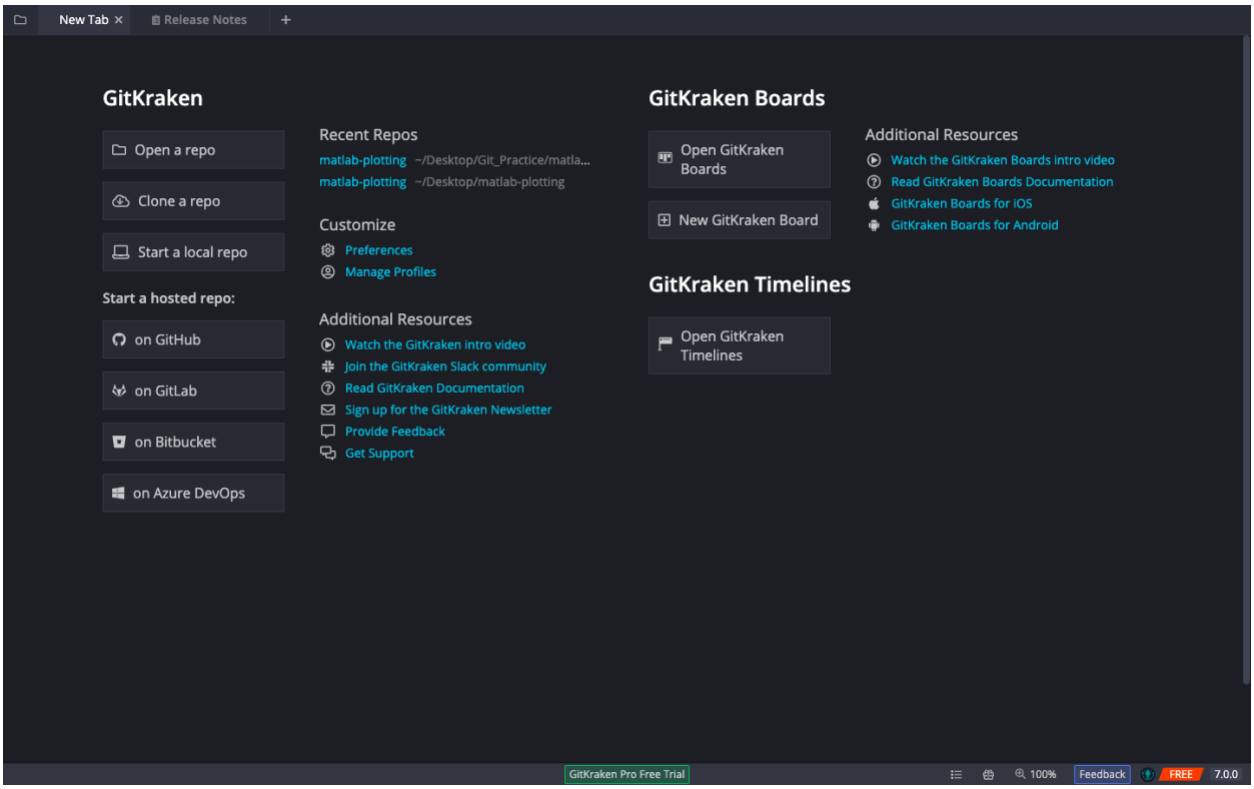
2.2 Local (On Your Computer) Providers

2.2.1 GitKraken

GitKraken is an application that you can install on your computer. GitKraken will allow you to use Git in a GUI interface, which means that you can navigate Git with your mouse in a user-friendly environment. To install, go to https://www.gitkraken.com and select the "Download for Free" option. The web page should automatically detect your operating system and pull up the proper download. If not, select "Other Platforms" to find the download for your operating system.



You may be brought to a web page thanking you for downloading the file. Once the file is downloaded, install GitKraken using the download on your computer. Once it's installed, open it. Your computer may warn you about it being an application downloaded from the internet. Open it anyway and you'll see the following screen:



GitKraken is now successfully installed on your computer. Before we continue with this tutorial, you will need to connect your GitKraken account to either a GitHub or GitLab account so that we can access repositories that are stored online. To do this, click on the "Manage Profiles" button underneath the "Customize" section.

For students and faculty, you can upgrade to a Pro version of GitKraken by associating your GitHub account with your university email address, discussed in Section 5.1 of the appendix. For others, you can upgrade your account to Pro according to GitKraken's pricing.[6] The key difference is that if you use the Pro version you can have private repositories that only you can see. If you decide to stick with the free account that's just fine, just make sure that all of your online repositories are public rather than private. You can always get the student upgrade later, and as of the present writing it is advertised to last for one year.

2.2.1.1 Creating a Repository

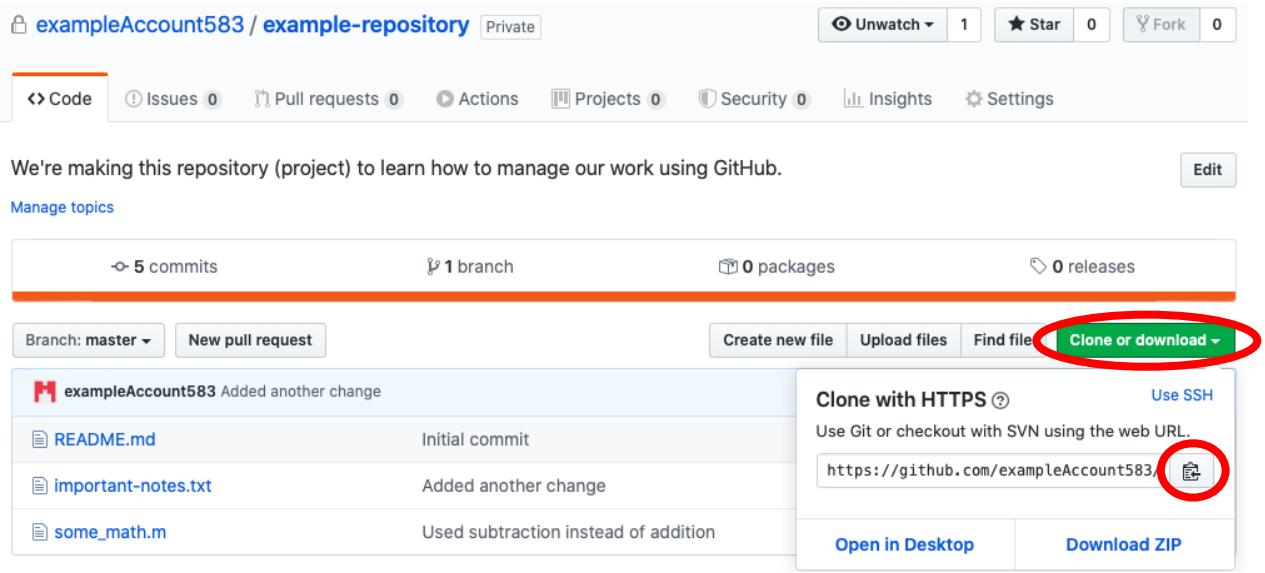
GitKraken provides lots of options for creating a repository. You can open a repository that you've already made, clone a repository from an online provider like GitHub or GitLab, or start a repository that stays only on your computer and is not synced up to an online provider. Lastly, you can create a repository that is automatically synced up to an online provider. This last option is called "Start a hosted repo". These are all good options that you can learn more about as your skills with Git and GitKraken grow. In this tutorial, we will take an existing repository from GitHub and "clone" it to our computer. Cloning means that we will make a copy of the project on our

[6] https://www.gitkraken.com/pricing

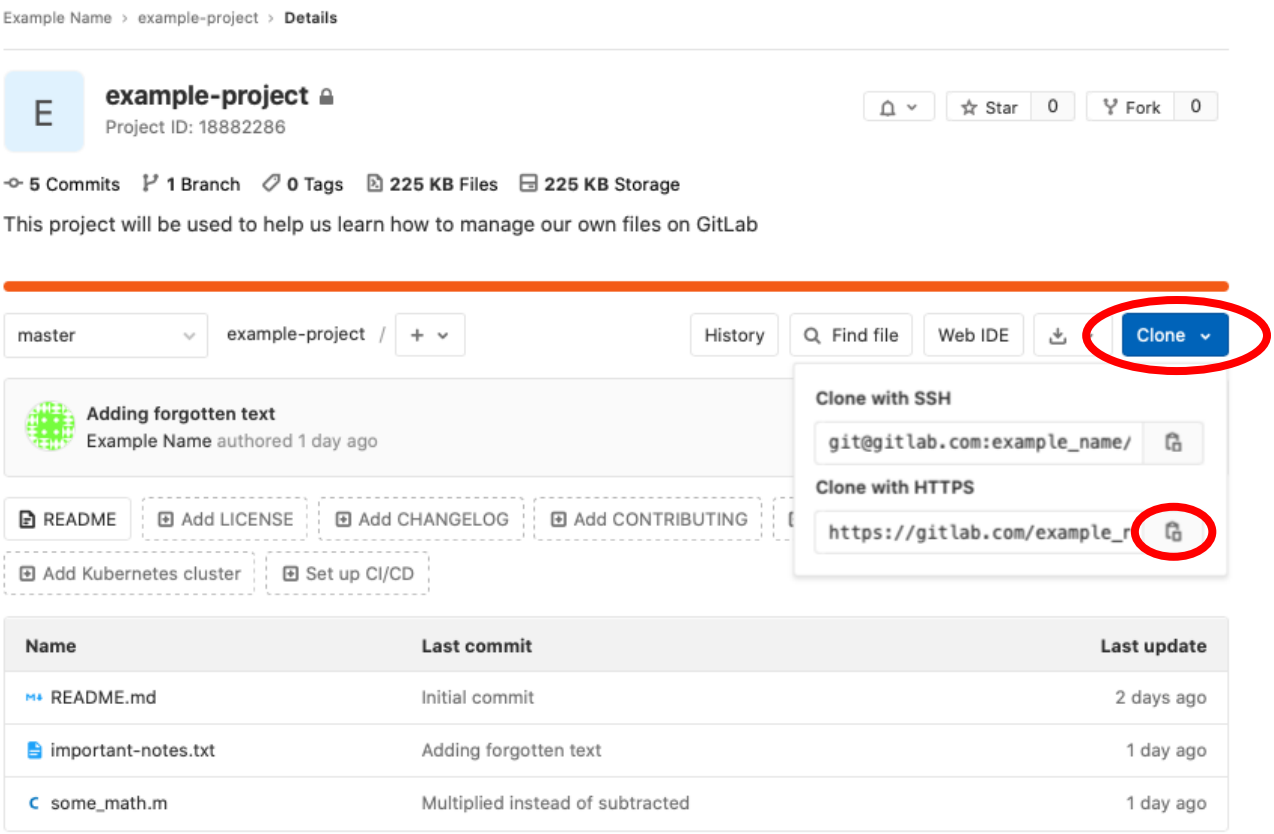
computer, where we can make our own edits, test them out, and then “push” those changes back to the online repository where they will then become available for other users.

2.2.1.2 Cloning a Repository

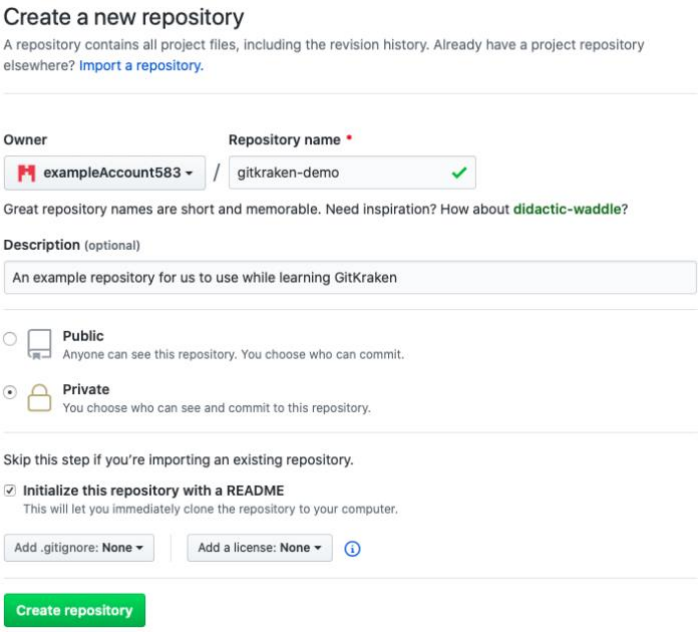
To clone a repository, open the repository on either GitHub or GitLab and click on the “clone” icon. This will open up a box where you should copy the link to clone with “HTTPS”. For GitHub, this is shown in the following screenshot:



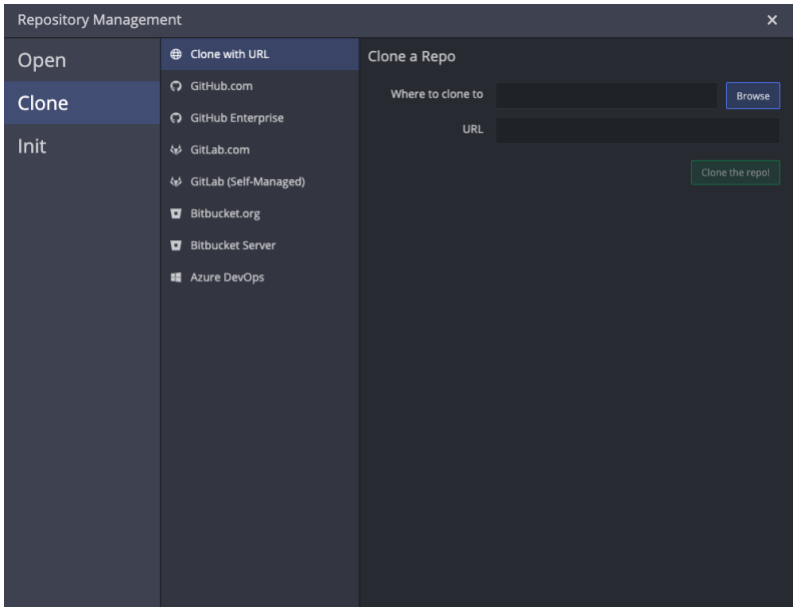
If you want to clone a repository from GitLab, this is shown in the following screenshot:



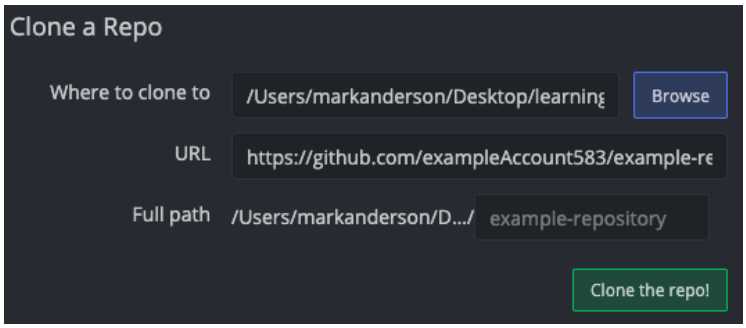
For this GitKraken tutorial, I have gone into GitHub and created a new empty repository called “gitkraken-demo”. You can follow along closely with this tutorial by creating a similar repository either on GitHub or GitLab. For a tutorial of how to create a repository on GitHub or GitLab, refer to Sections 2.1.1.1 and 2.1.2.1 respectively. For your reference, below is the repository creation page on GitHub:



Once the repository is created, click on the green “Clone or download” icon and copy the HTTPS link. Then return to GitKraken and click on the “Clone a repo” icon on the left-hand side of the welcome screen. You will see the following box:

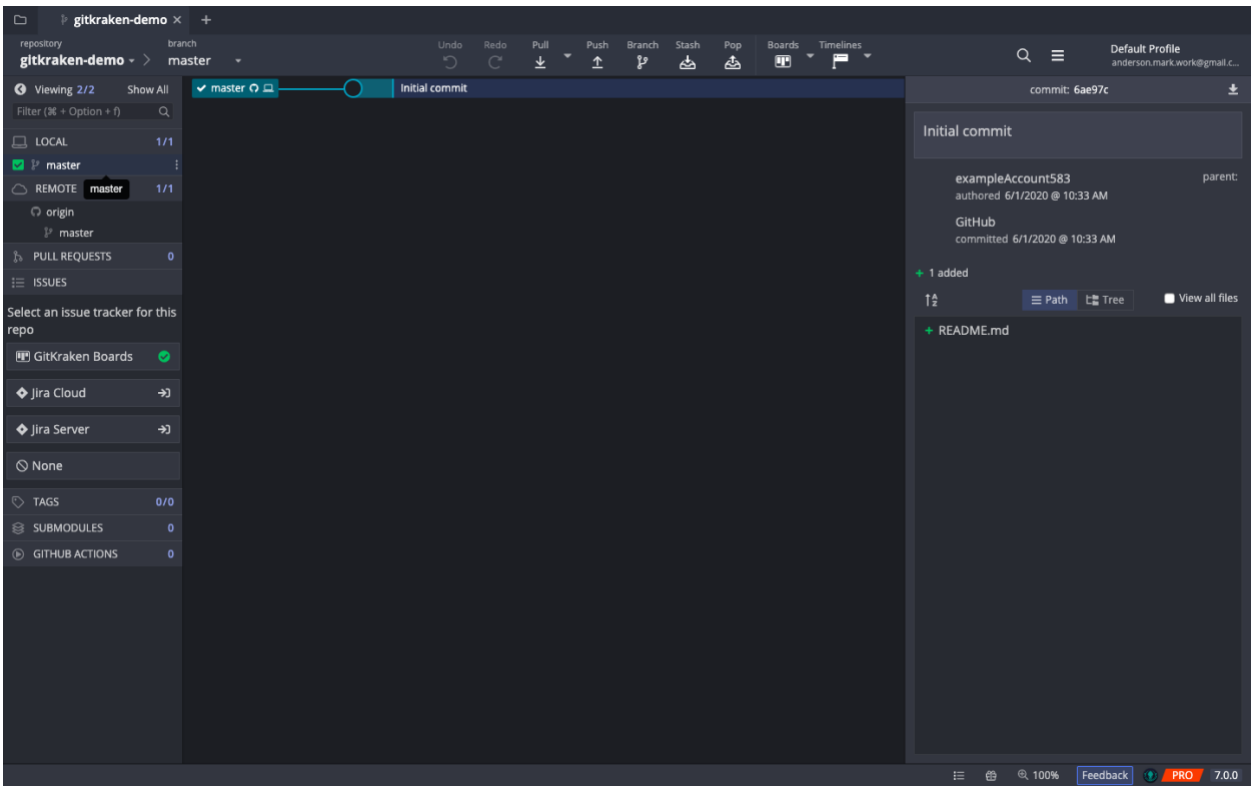


Click on the “Browse” button to select a place on your computer to store the repository. Then paste the “HTTPS” link that you copied from either GitHub or GitLab and paste it in the “URL” box.



Click on the green “Clone the repo!” button. After the repository is successfully cloned to your computer, you will see a banner across the top of the screen that says:

Go ahead and click on the “Open Now” button to open the repository. You will be brought to the following page:



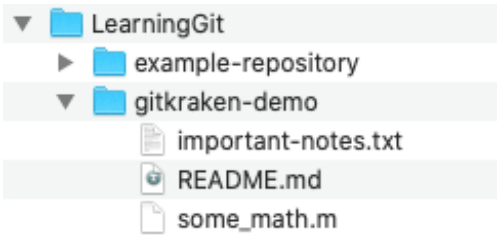
This page may look daunting with all of the buttons everywhere, but we will go through everything you need to know to get started with GitKraken.

2.2.1.3 Getting files into GitKraken

Let’s say that you have two files that you would like to keep track of with GitKraken:

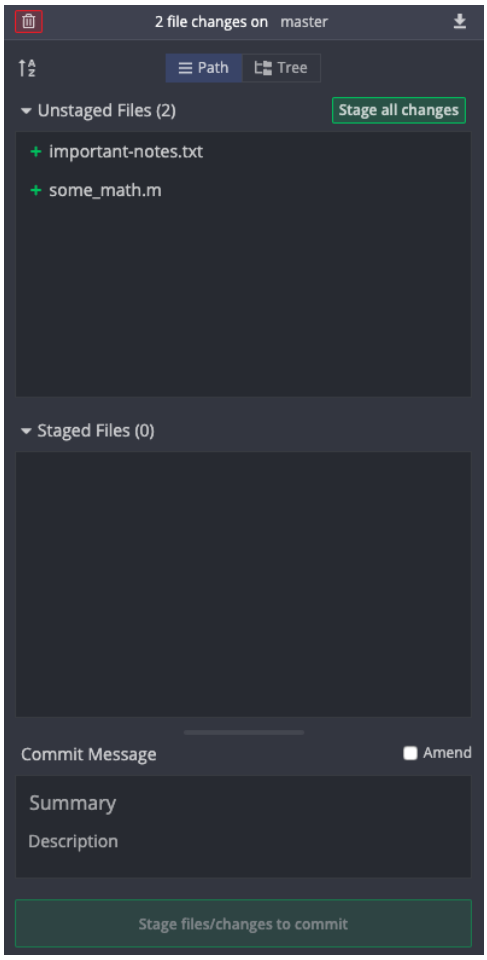
some_math.m
important-notes.txt

When you cloned the repository to your computer it created a folder named after the repository. Find that folder and put the files that you want to keep track of in that folder. Here is an example of what this might look like:



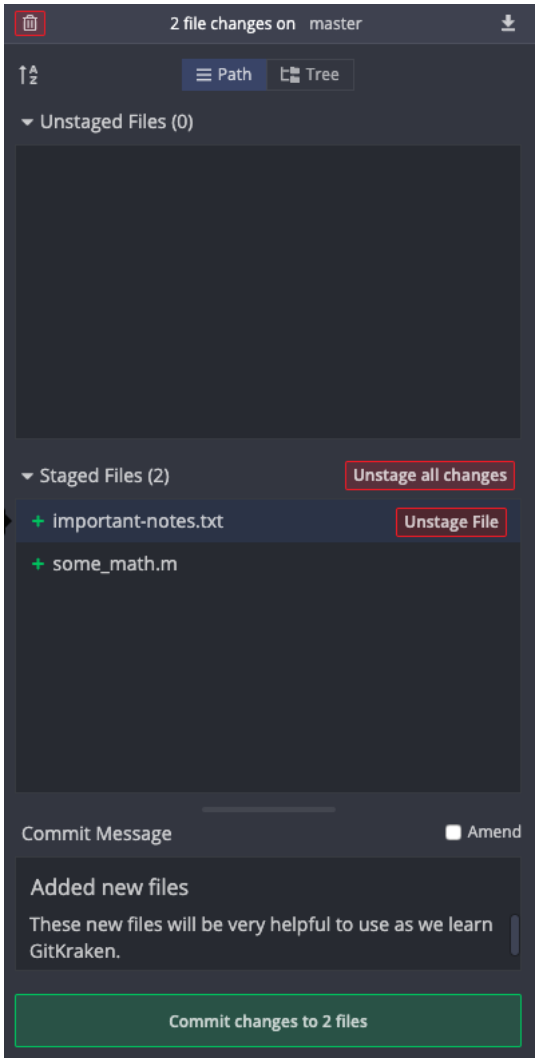
GitKraken is aware that there are now new files in the repository and when you return to the GitKraken screen you will see some changes on the screen. In particular you will see this in the "tree":

The little "+2" means that there are two file changes (adding the files counts as changing them). The box next to the "+2" is grayed out because the changes represented there have not been committed to Git yet. To commit them, click on the grayed-out box. On the right-hand side of the screen you will see:



Right now, the two new files have not been staged. If we try to do a commit right now, they will not be saved because we never told Git that we wanted them to be part of the commit. If we want them to be committed then we need to add them to the staging area, which tells Git that we want them to be committed to the repository on the next commit. To stage a file, hover over it and press the green “Stage” icon that appears. Or if you want to stage all of the files press the green “Stage all changes” icon.

It is very important to include a commit message explaining what you changed about the files (in this case we just added them, but if we had modified existing files then we would summarize those changes). In the “Summary” section put a very brief summary of the changes and if you need more room to explain the changes then type the detailed explanation in the “Description” section. Once this is complete the right-hand-side of the screen should look something like:



Once you're satisfied that these are the changes you want to make and that you've described them well, click on the green "Commit changes" icon at the bottom of the right-hand-side of the screen to commit the files. Now the center display will look like this:

Each line represents a commit, and the commit message for that commit is displayed next to it. The box that says "master" with a computer and checkmark next to it represents the version of the files that are on your computer. The other "master" box indicates the version of the files that are on GitHub (or GitLab, depending on where the online version of your repository is). It's important to note that the repository on your computer and the repository online are now out of sync. This bring us to the following section.

2.2.1.4 Syncing Changes with a Remote Repository

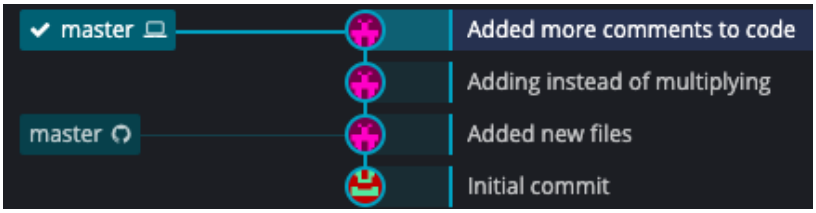
We need a way to re-sync our remote (online) repository with our local (on our computer) version of the repository. First, we must make sure that we download any changes that the remote repository has that we don't have on our computer. This is unlikely if you are working alone, but very likely if you are working in a team. To do this, go to the actions toolbar at the top of the screen:

Click on the "Pull" button to download any changes that the remote repository has that your local repository does not. After doing this (and perhaps making sure that your changes don't interfere with someone else's changes) click on the "Push" button to send the changes that you have made to the remote repository. Once successful, you will see that the tree looks like this:

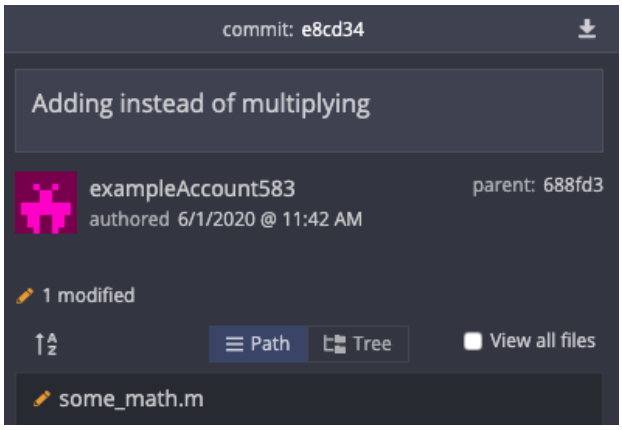
Now your remote and local repositories are back in sync. If you go online and look at the repository you will see that the changes you made (adding those two files) are officially part of the repository online. You do not need to push your changes to the remote repository after every commit, instead you can do it after every few commits or whenever is most useful for your particular project.

2.2.1.5 Comparing Past Versions of the File

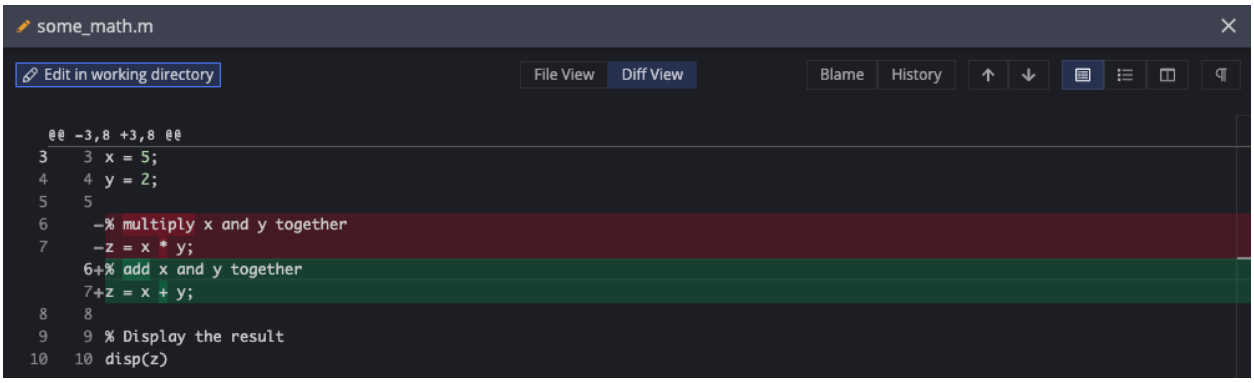
Let's say that you've made some changes to some_math.m and you have committed them to your local repository (the one on GitKraken) and your tree looks like this now:



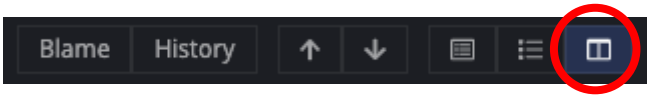
To see how the files have changed between commits, click on a commit and go to the right-hand-side of the screen:

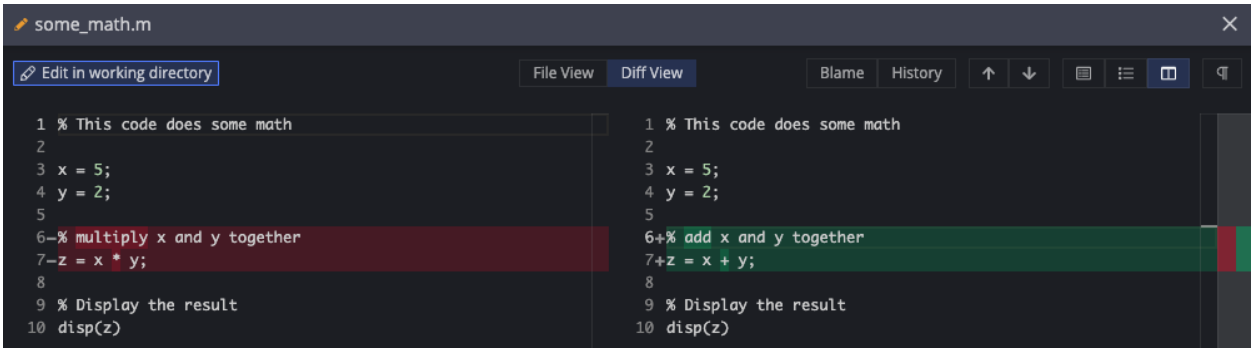


Click on the file that you want to view (in this case some_math.m) and the center of the screen will now show the changes that happened:



The red regions show what was removed and the green regions show what was added. In the upper right of this display there are some options for different views. The split view is a convenient way to view the file differences:





There you go! You can now use GitKraken for managing your files.

2.2.2 The Terminal

The Terminal is the original interface for Git and continues to be arguably the most efficient and effective way to use Git. Understandably, interfacing with Git this way may appear daunting and you may feel discouraged. This is normal, and this tutorial will tell you everything that you need to know to start using the terminal, and once you know this basic functionality, the other features will come to light as needed.

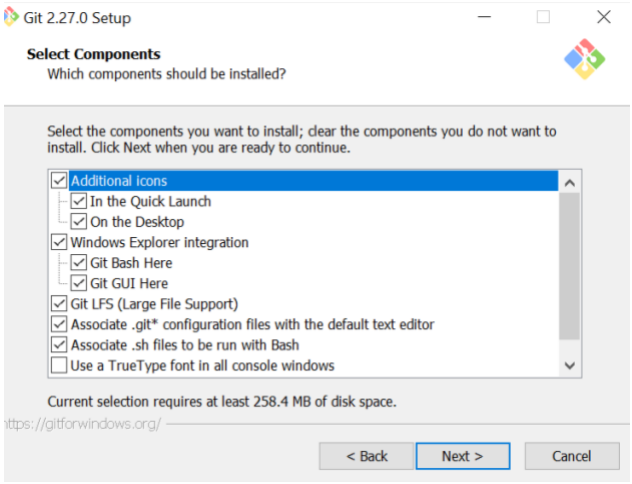
There are several useful online references for how to install git on your computer[7,8]. These websites have instructions on how to install Git on Windows, macOS, and Linux computers. In this guide we will focus on computers that run Windows and macOS. Although terminal output may vary slightly between Windows and macOS computers, all of the commands that you will enter are the same regardless of the operating system.

2.2.2.1 Windows

When using a Windows computer, we will install a program called "Git for Windows". This installs a Git along with a program called "Git Bash" that will enable us to interact effectively with Git. To do this, go to https://git-scm.com/downloads and click on the download icon. You should be brought to a new page and the download should automatically begin. If not, you can manually download it on the web site.

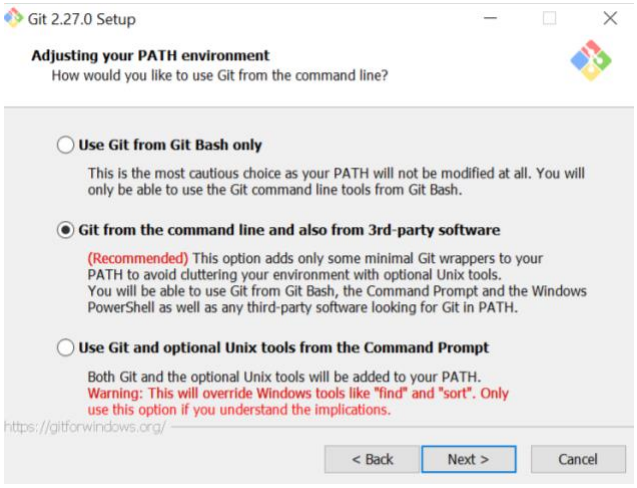
Open the download to begin the installation process. Most of the default settings during the installation are already set for what we want, but here are some recommended adjustments:

1. You want to add an icon to both the Quick Launch and Desktop locations on your computer for easy access to Git Bash.

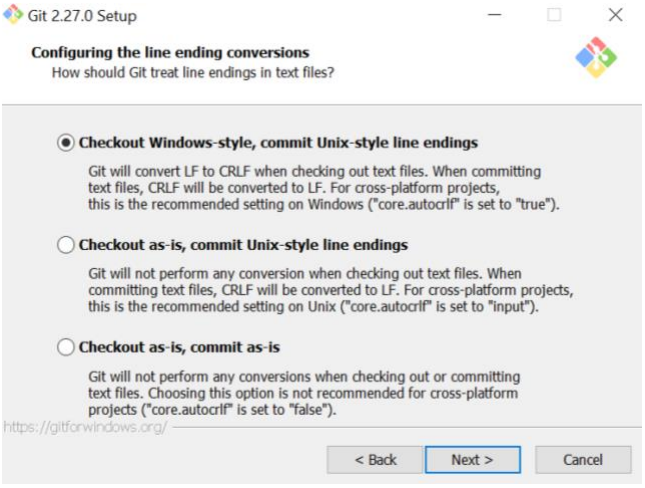


2. You want to select the "Git from the command line and also from 3rd-party software" option when it appears.

[7] https://git-scm.com/book/en/v2/Getting-Started-Installing-Git
[8] https://www.atlassian.com/git/tutorials/install-git



3. If you work with multiple operating systems, or if anyone on your team owns a non-Windows computer, you will want to select the “Checkout as-is, commit Unix-style line endings” option when it appears. This will make it so that all of your files in the repository have the same type of file ending.



After the installation is complete, you will have a shortcut on your Desktop called “Git Bash”. This will provide us with the terminal-style interface for Git. From here on, whenever we refer to the terminal, go ahead and use Git Bash. Jump ahead now to Section 2.2.2.3 to finish setting up Git.

2.2.2.2 MacOS

Go to https://git-scm.com/downloads and click on the download icon. This will pull up a web page with several installation methods. Although you can use any of the methods, the simplest two approaches are to either use XCode or Homebrew. Both work well and are described below.

2.2.2.2.1 Installing Git with XCode

Open your terminal app and type:

git --version

You may be surprised to find that if you have XCode then you might already have Git, since it comes with XCode. If you already have Git then typing the above command in the terminal will tell you which version you have, and you're good to go! If you do not have Git, your computer will prompt you to install it. You can navigate the installation without actually installing XCode, but if you have a hard time navigating the installation without installing XCode, go ahead and allow your computer to install it for simplicity.

2.2.2.2.2 Installing Git with Homebrew

If you decide to use the homebrew installation, make sure to install homebrew first[9]. After homebrew is installed open your terminal and type:

brew install git

This will install Git for you.

2.2.2.3 Setting Up Git Once It's Installed

Regardless of which method you used to install Git, to verify that you now have Git on your computer, open the terminal and type:

git --version

Your computer should respond by outputting:

git version <some numbers>

Congratulations! You have successfully installed Git on your computer. Now it's time to tell Git your name and email address. This information is important because every time you do something with Git your name and contact information will be associated with it, and this will make working in groups easier for you and your team. To tell Git your name, type:

git config --global user.name "example name"

where you put your name in the quotes. Then, to tell Git your email address, type:

git config --global user.email "example@email.com"

Replace the example email with your own email in quotes.

[9] https://brew.sh

2.2.2.4 Navigating the Terminal

The terminal lets you navigate your computer using only your keyboard. As you become accustomed to using your computer this way it will become a very efficient tool. This subsection provides some of the basic navigation commands so that we can begin using Git.
When you first open the terminal, you will be inside of your home directory. To see what directory you're in use the "present working directory" command. Type:

pwd

Now that we know what directory we're in, we can use the "list" command to tell us what subdirectories are in the current directory so that we can navigate to any location on our computer. Type:

ls

This will provide you with a list of the directories within the current directory. To move into one of those folders use the "change directory" command. For example, if there was a folder called "Work" in the current directory and you wanted to move into it, type:

cd Work

If there is a space in the directory name, you'll want to wrap the directory name in parenthesis. Again, use the "list" command to tell you what directories and files are within this new directory, and repeat until you have navigated to where you want to go. If you get lost, you can use the following two commands:

cd (this command, if left empty, will return you to the home directory)
cd .. (this command, if followed by two periods, will take you up one directory)

The best way to understand these commands is to practice navigating around your computer using them. There are plenty of other commands out there that you can learn if you want but are not necessary for this Git tutorial.

2.2.2.5 Creating a Repository

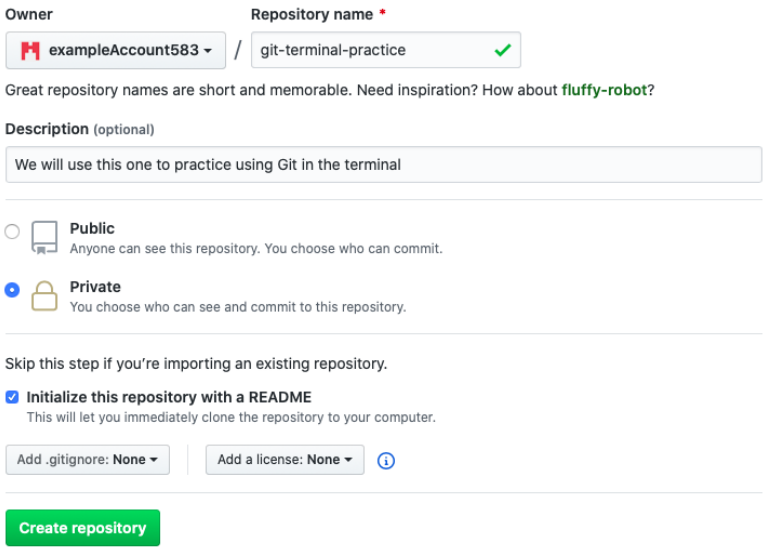
Let's say that you have some files in a folder on your computer that you want to turn into a Git repository. This can be done one of two ways:

1) Create the repository on your computer and then push it to an online service
2) Create the repository online and then clone it into the folder on your computer

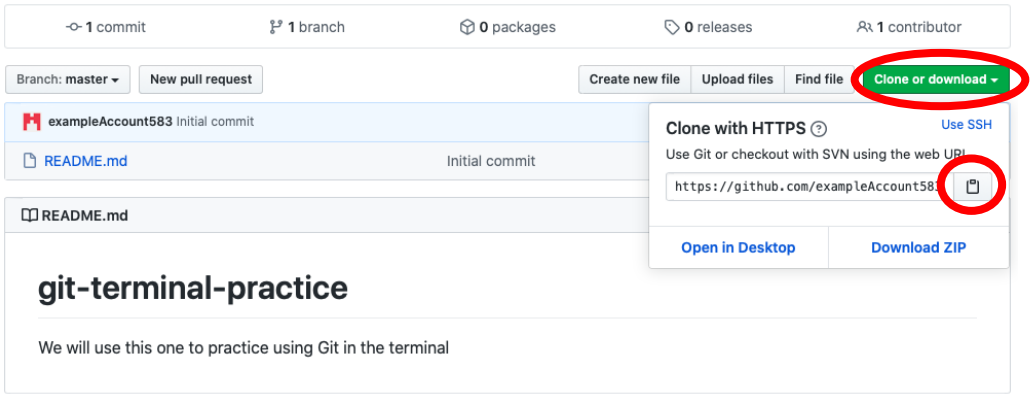
The second method is the most straightforward for us to use here. We will use GitHub, but the principle is exactly the same for GitLab: create an empty repository and clone it into the folder on your computer that contains all of your files.

2.2.2.6 Cloning a Repository

Go ahead and create an empty repository on either GitHub or GitLab and give it the name "git-terminal-practice" or something similar to that. For your reference, here is the repository creation information for the repository used in this tutorial:



To clone a repository on GitHub, click on the green "Clone or download" icon and copy the link provided. On GitLab, click on the blue "Clone" icon. In both cases, make sure you copy the HTTPS link. On GitHub, this looks like:



To clone the repository, use the terminal to navigate to a folder that you want to put the repository into. Cloning the repository will create a folder named after the repository in whatever directory you are in. To clone, use the "git clone" command:

git clone https://github.com/exampleAccount583/git-terminal-practice.git

If this is your first time using Git in the terminal, you will be prompted for your account's username and password. If you have used Git before in the terminal with this account

(exampleAccount583 in this instance), then the repository will be cloned without the need for a username and password. Lastly, if you have used Git with a different GitHub account from your terminal, your terminal might get confused and not let you clone the repository because the username and password that you have in the system does not match the one to clone the repository from this new account. For help with this, see the appendix in Section 5.2. Below is an example of how to clone a repository for the first time, where commands that I typed are highlighted in green and the rest is the terminal output:

```
markanderson@MarkAndComputer ~ % pwd
/Users/markanderson
markanderson@MarkAndComputer ~ % cd Desktop
markanderson@MarkAndComputer Desktop % git clone https://github.com/example
Account583/git-terminal-practice.git
Cloning into 'git-terminal-practice'...
Username for 'https://github.com': exampleAccount583
Password for 'https://exampleAccount583@github.com': Password goes here
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
markanderson@MarkAndComputer Desktop %
```

The repository is now cloned! You will generally want to have a better place to store your repositories than your Desktop, so consider creating a designated “projects” or “repositories” folder somewhere on your computer. If you open the folder you will see that anything that was on the online repository has now been downloaded to your computer.

2.2.2.7 Uploading Files

Let’s say that you have two files that you now want to add to this repository:

some_math.m
important-notes.txt

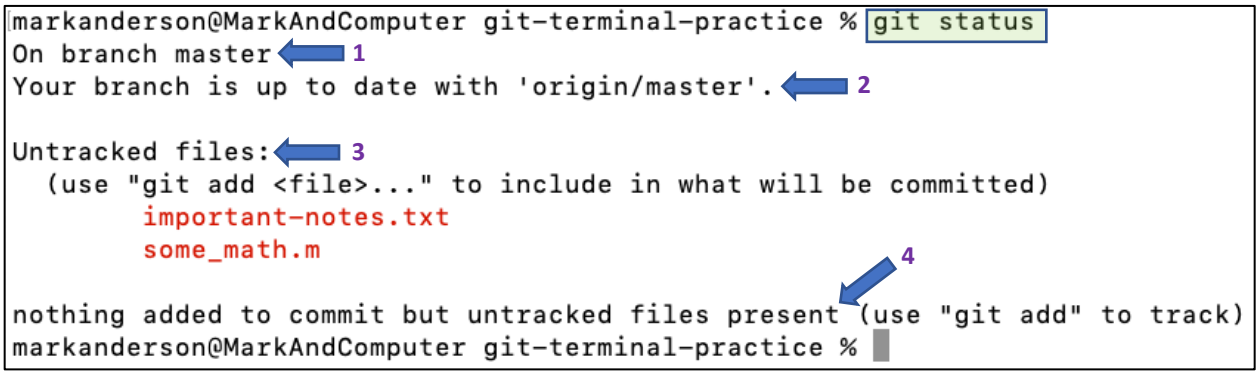
If you made these files before cloning the repository to your computer then go ahead and paste them into the repository folder. If you made them after cloning the repository, you can just save them directly into the folder. However you went about it, you now have two files in your “git-terminal-practice” repository folder and you want to make sure that Git keeps track of them. First, use the “cd” command to move into your repository:

cd git-terminal-practice

Now if you use the “ls” command the terminal will show you the repository’s contents.

2.2.2.7.1 The “git status” Command

One of the first commands to learn is the “git status” command. This command will tell you the current state of your repository:



1. You are on the main branch of the repository, more on branches in Sections 1.2.4.
2. The branch you are on (the master branch) is currently up to date with the online repository. This means that you have not made any new commits that the online repository isn’t aware of, and vice versa.
3. Git sees that there are new files in the folder and is telling you that it sees them but is not yet tracking them. Often you will actually have files in your repository folder that you don’t want Git keeping track of, so this is a useful feature.
4. Git is again telling you that there are untracked files, and to use the “git add” command to add them to the staging area so we can commit them to the repository.

2.2.2.7.2 Staging

The staging area is a place to say, “Let’s bundle together all the files that I want to commit to Git on my next commit”. You can add several files, or just one, all at once or at different times. The next time that you commit the files to Git, you will commit all of the files that are currently in the staging area. To add a file to the staging area, use the “git add” command:

git add some_math.m

If you want to add all of the files in your project at once, then use a period instead of a specific file name. Type:

git add .

Those files are now set aside and ready to be committed. Here is an example:

```
markanderson@MarkAndComputer git-terminal-practice % git add some_math.m
markanderson@MarkAndComputer git-terminal-practice % git add important-notes.txt
markanderson@MarkAndComputer git-terminal-practice % git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:  1
  (use "git restore --staged <file>..." to unstage)  2
        new file:   important-notes.txt
        new file:   some_math.m

markanderson@MarkAndComputer git-terminal-practice %
```

1. You have made changes to the repository (added files) and staged the files that have been changed (being added technically counts as being changed)
2. Git is telling you how to remove the files from the staging area. This is useful if you decide you're not quite ready to commit a file. If you wanted to remove important-notes.txt from the staging area (we won't actually do this), then you would type:

git restore --staged important-notes.txt

2.2.2.7.3 Committing

To commit all of the files that are in your staging area, use the "git commit" command. This will tell Git to take a snapshot of your files in the staging area as they are at this moment and save that snapshot forever. This will let you see how your files change over time and even restore old versions if you need them later.

git commit -m "type a message here"

The "-m" lets you add the message in the same line as the commit command. Otherwise it will open a window that prompts you for a commit message and that window that can be quite confusing. If you find yourself in that window type <insert windows shortcut here> ("cmd + w" on macOS) to escape from it. The commit message denoted by the text within the quotes is very important because in that short message you describe the changes that you made to the files being committed, which will be useful to you in the future as you look at the file histories. Below is an example commit:

```
                                                                    1
markanderson@MarkAndComputer git-terminal-practice % git commit -m "Added math and notes files"
[master ad2d475] Added math and notes files
 2 files changed, 21 insertions(+)  2
 create mode 100644 important-notes.txt
 create mode 100644 some_math.m
markanderson@MarkAndComputer git-terminal-practice % git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.  3
  (use "git push" to publish your local commits)  4

nothing to commit, working tree clean  5
markanderson@MarkAndComputer git-terminal-practice %
```

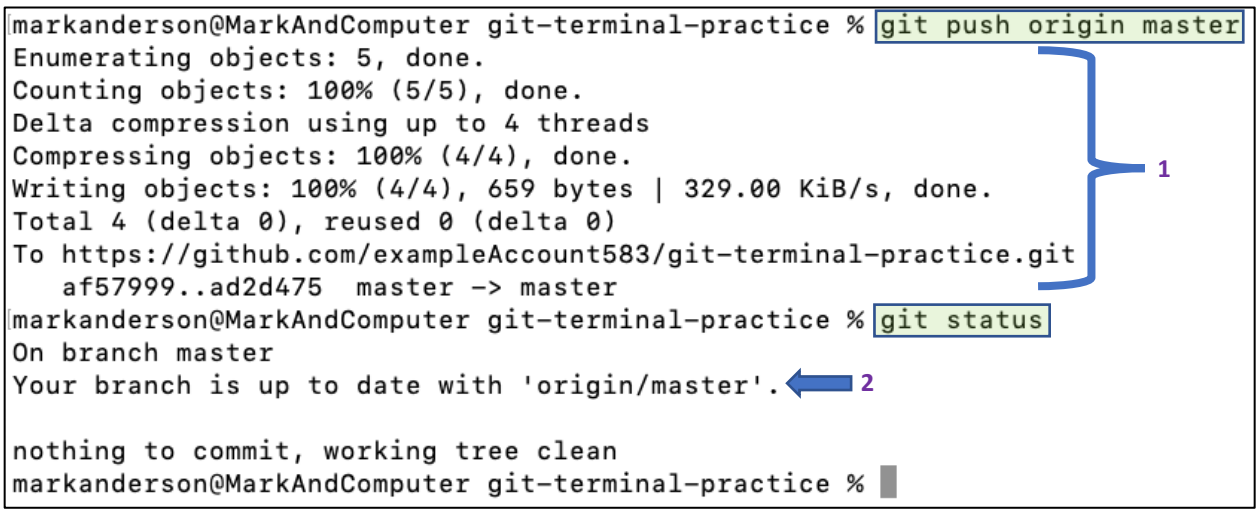
1. A unique number associated with the commit.
2. How many changes were made (adding new files will count every line of each file as a change).
3. The repository on your computer (the local repository) is ahead of the online repository (the remote repository or "origin/master") by one commit. In other words, you have made changes to the repository, but only on your computer. If you want to sync those changes with the online repository, you will need to "push" the changes, discussed in the next section.
4. Telling you how to "push" the local repository to the online repository
5. Git is unaware of any new changes to your files since your last commit.

2.2.2.7.4 Pushing

Now that the files are committed, they are forever remembered by Git in whatever state they were in at the time of the commit. These changes are saved locally on your computer, so if you want them to be permanently saved online then you need to push the commits to the online repository. This is done by typing:

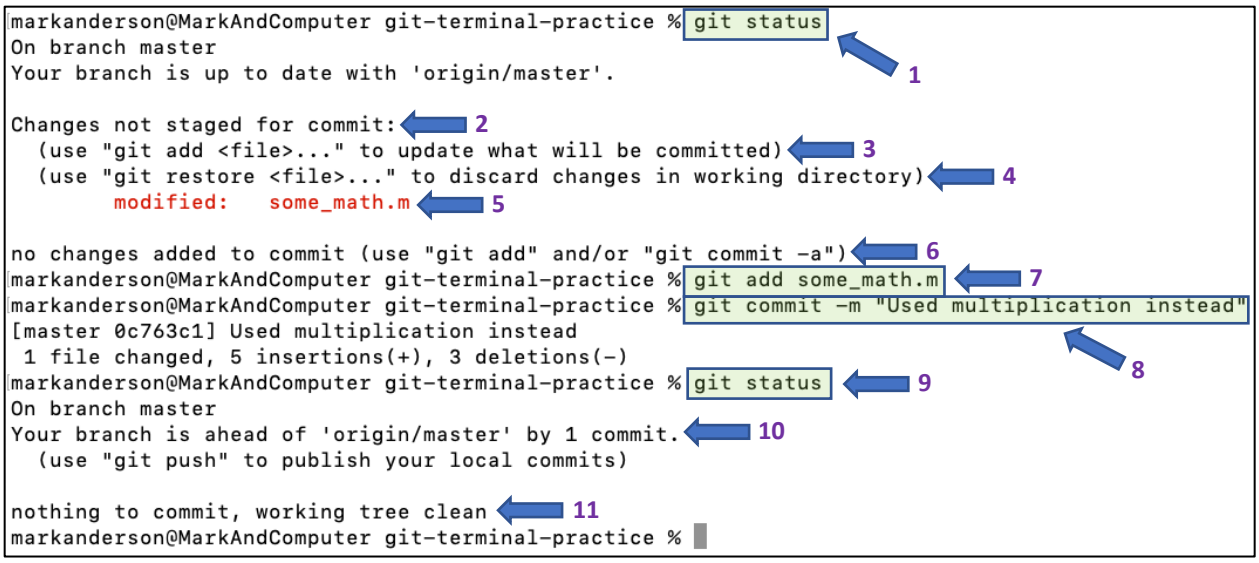
git push origin master

Now your changes are synced with the online repository. When you do this, all commits that haven't been uploaded to the repository will be uploaded, so you don't need to do this every time you make a commit, but rather at least once a day should be sufficient, or whenever you decide is appropriate. Below is an example push:



1. Git is uploading the files to the remote repository
2. Your local repository and the remote repository are now in sync

Let's say that you have made some changes to some_math.m and would like to commit those changes to Git. Go ahead and commit those to Git as discussed above. Here is an example:



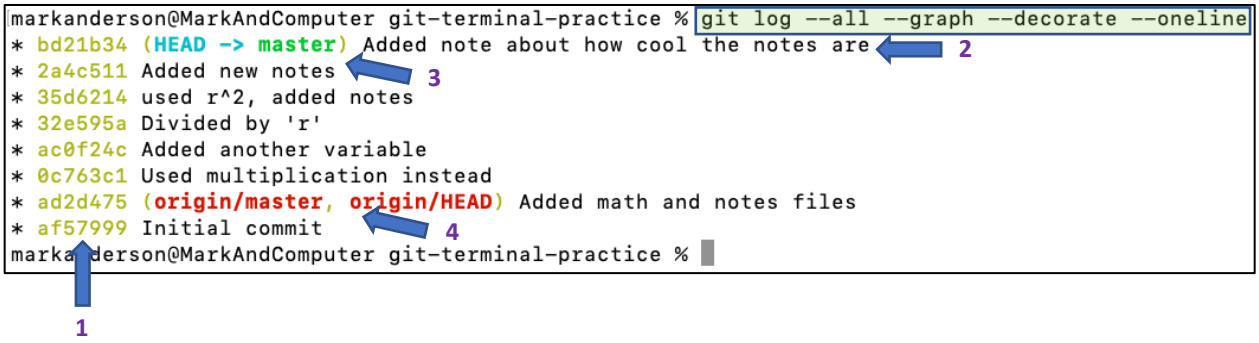
1. Begin by checking the repository status
2. There are changes in the repository, but they are not staged. This means that if you do a commit right now, those changes will not actually be committed.
3. A reminder for how to add files to the staging area
4. This tells us how to restore a file to how it looked on the last commit. This is useful if we have made changes, decided we didn't like them, and then wanted to revert the file to the way it was on the last commit.
5. The file that has been modified in our repository is some_math.m. There could be many modified files, but in this example we just have the one.
6. A reminder for how to add files to the staging area. Also included is a shortcut for how to add a file to the staging area and commit it in one line.
7. Adding some_math.m to the staging area
8. Committing all the files in the staging area (just some_math.m in this case) to Git.
9. Checking the repository status after the commit to make sure it looks as expected.
10. We are now ahead of the remote repository again by one commit. If we want, we can push our changes to the remote repository now. We can also keep making changes and commits and decide to push those changes to the remote repository later. When we do push to the remote repository, all of the commits we have made will be added to the remote repository so we can access any versions of our files that were ever committed.
11. There are no more changes in the repository since the last commit

2.2.2.7.5 Comparing Past Versions of Your Repository

To view the history of your repository, use the "git log" command. Using this command will show each commit and information about it, like it's commit message. You can also add options to the "git log" command to make the output more readable. I recommend using:

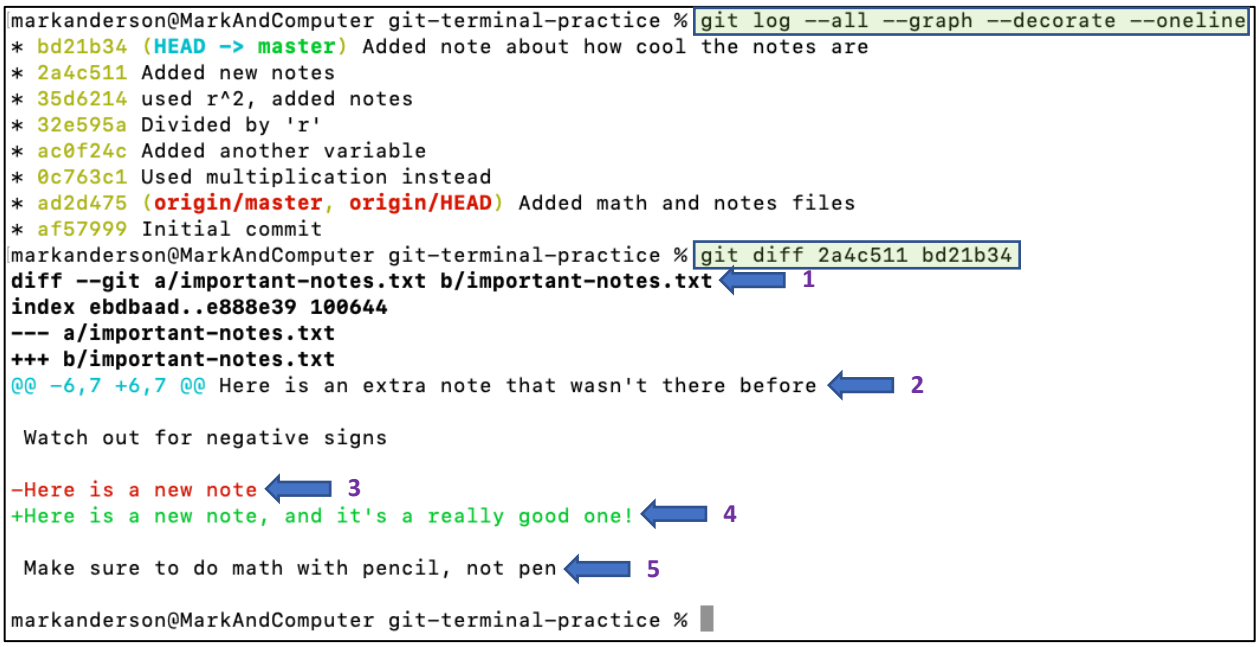
git log --all --graph --decorate --oneline

This will output each commit, along with its ID number and commit message, line by line. Because this is a longer command, I recommend making it into an alias, discussed in Section 4.4. My personal alias for this command is “hist” so I can just type “git hist” and get the proper output. Here is an example of what this output may look like:



1. The commit IDs listed with the newest at the top
2. The associated commit messages
3. The (HEAD -> master) tells you that you are currently on the master branch in your repository.
4. The (origin/master, origin/HEAD) tells you what commit the remote repository is on. If you go to the remote repository now it will have the files as they were back immediately after that commit.

We can use the commit ID numbers to look at the differences between files from one commit to another. To do this we use the “git diff” command, followed by the two ID numbers we would like to compare. Here is an example:



1. It looks like important-notes.txt was changed between these two commits
2. Git will show us some context for the changes in the file, here is the beginning of the file.
3. Here is a difference, this is what the first commit says on this line. The first commit is denoted by the first ID number after the “git diff” command
4. Here is what the second commit says on that same line. Even though no text was actually deleted, Git considers a change to a line as though the line were deleted completely and then replaced by the new text.
5. End of context in the file.

And there you have it! You can now use Git in the terminal!

3 Working in a Group

This chapter will cover working in groups on a single project.

3.1 More Detailed Workflow Description

3.2 GitHub

3.2.1 Creating Branches

3.2.2 Merging Branches

3.3 GitLab

3.3.1 Creating Branches

3.3.2 Merging Branches

3.4 GitKraken

3.4.1 Pull Requests

3.4.2 Creating Branches

3.4.3 Merging Branches

3.5 The Terminal

3.5.1 Pull Requests

3.5.2 Creating Branches

3.5.3 Merging Branches

3.6 Exercises

4 Advanced Topics

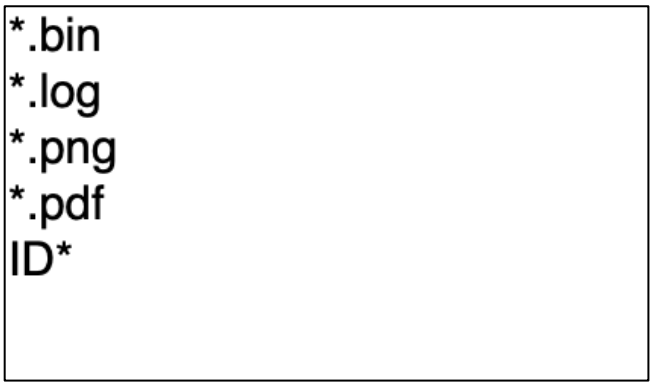
This section covers topics that go beyond the simple Git workflow. Most of this material will be taught using the terminal (command line).

4.1 Choosing Your Text Editor

4.2 Using Diff and Merge Tools

4.3 Ignoring Unwanted Files

There are many files that you may have in your repository folder but do not actually want to back up to Git. This may be because they are large data files or images or would unnecessarily clutter the remote repository if pushed online. Git provides a way to ignore such files from the repository while still letting you have them in the folder on your computer. The solution is to create a **.gitignore** file and place it in your repository. This is a file that you use to tell Git which files to ignore. A typical .gitignore file may look something like this:



The "*" symbols represent any text. For example, any file that end in ".log" will be ignored from this particular repository. Additionally, any file that starts with "ID" will also be ignored from this repository.

Notice how the .gitignore file name begins with a period. This means that it will actually be a hidden file on your computer. Your computer and Git are very aware of its presence, but if you want to make any more changes you will need to enable hidden file viewing on your computer. The shortcut for this <on windows> and shift+cmd+. on macOS. You may be surprised to find that there are actually a lot of hidden files on your computer. Use the same shortcut to disable this mode.

4.4 Creating Aliases

4.5 Using Tags

Tags are a way to create "releases" of your code. For example, if you had your code to a point where it was functional, but you know that you're going to keep working on it, you might give it the tag "v1.0.0". Git enables you and any other user to actually go and clone the repository at a particular tag, giving them the files as they were at the time that tag was created. An application of this would be if you were creating a presentation or writing a paper for which you generated figures. Experience says that eventually you will want to go back and recreate those figures for a

future project. If you tagged your code at the time that you made your figures, then you can easily go and download the code exactly as it was when you generated those figures. It would also be wise to specify along with each tag exactly how the data were stored so you can re-use the same file system for your data, further minimizing the effort to regenerate those figures.

To create a tag, type:

git tag -a <tag> -m “Message”

For example, this could look like:

git tag -a v1.0.0 -m “First Release”

To see a list of all the tags in your repository just type:

git tag

To delete a tag, type:

git tag -d <tag>

Tags are not automatically pushed with a push command. To push a tag to a remote repository type:

git push origin <tag>

4.6 Submodules

Submodules are used to more or less embed a repository inside of another repository. A benefit of using submodules is that you can have some sort of “core files” repository for an overarching project and embed that “core files” repository inside of all of the other repositories in the overarching project. This way, you can ensure that everyone has a consistent file structure on their computers so that the code works universally. However, you may want to limit who has access to the “core files” so that you don’t have any unexpected changes that will affect other projects. Using submodules to embed the “core files” into the other repositories lets the repositories access the “core files” without granting editing rights to the “core files” for team members working in the repository. As needed, you can release updated versions of the “core files” and have everyone update their versions of the repository to include this update to the “core files”.

To add a submodule to a repository, navigate to the target repository on your terminal. Then type:

git submodule add <url/ssh>

git commit -m “added submodule”

git push

If changes have been made within the submodule (e.g. a new revision of “core files” has come out) then you can update that online by typing:

git submodule update --remote --merge

git push

Other team members can now download the latest updates by opening their terminal and navigating to the repository on their computer and typing:

git pull

If they are cloning the repository then they need to change the clone command to account for the folders within the repository:

git clone --recursive <url/ssh>

4.7 Using Git with Other Software

4.7.1 Visual Studio Code

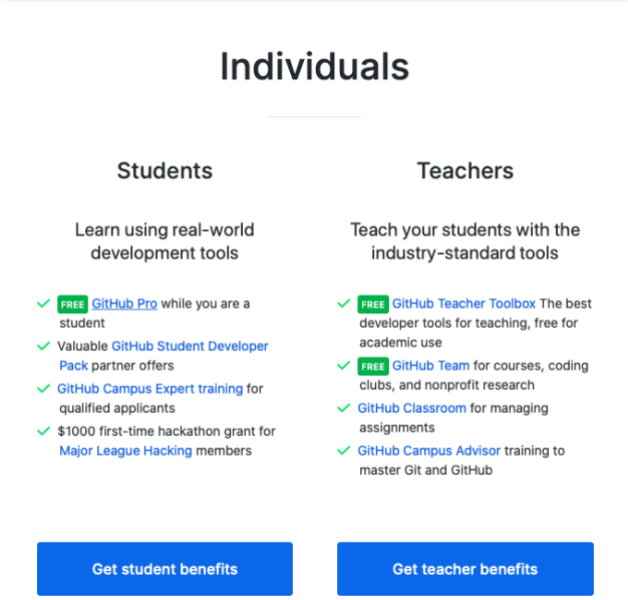
4.7.2 MATLAB

5 Appendix

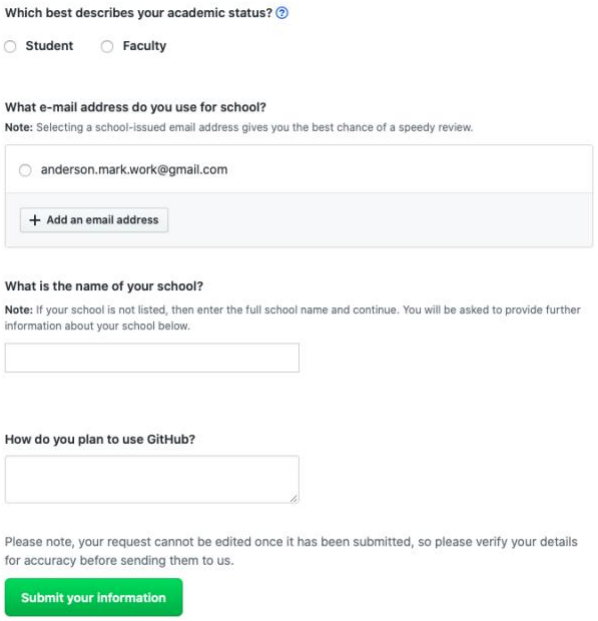
5.1 Student Upgrade to GitKraken Pro

5.1.1 Getting the GitHub Student Developer Pack

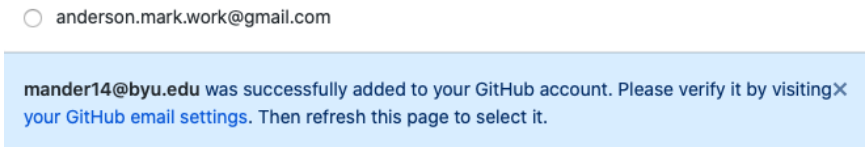
As a student, you can get a free upgrade to GitKraken Pro by associating your GitHub account with your personal university email address. Go to https://education.github.com/pack and select the blue “Get the Pack” icon. This will bring up the following page:



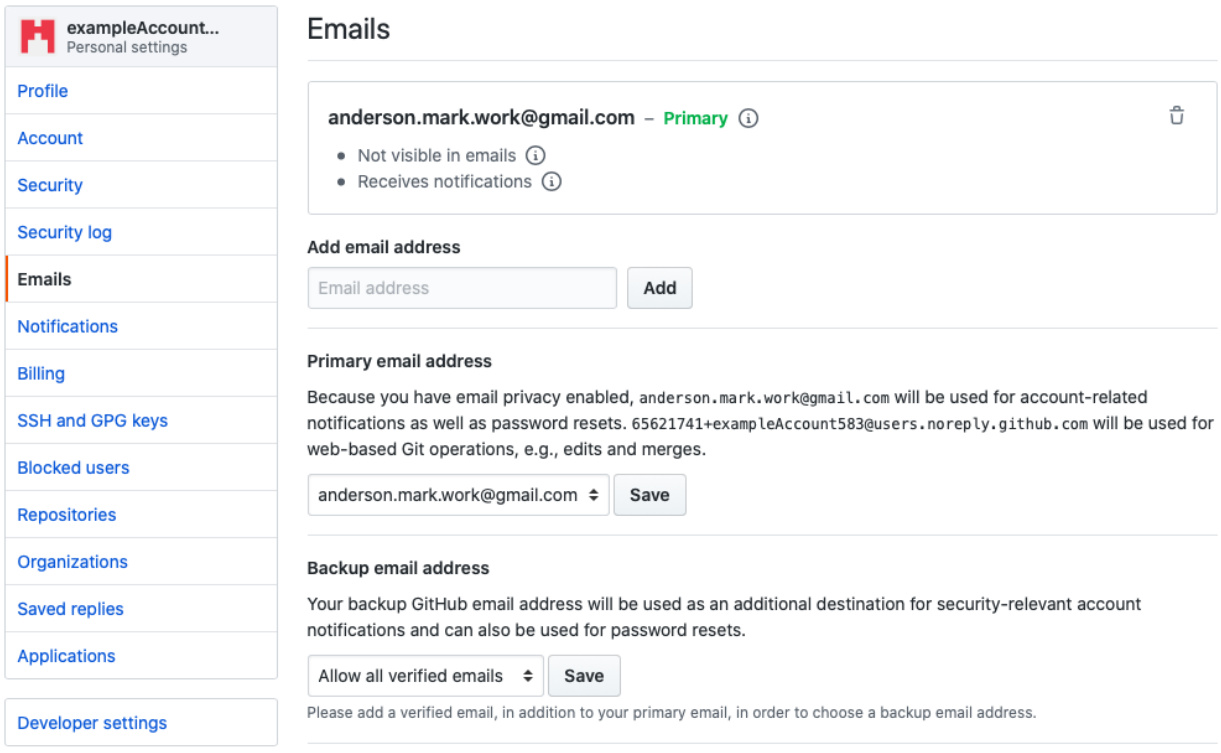
Select “Get student benefits”. You will see the following:



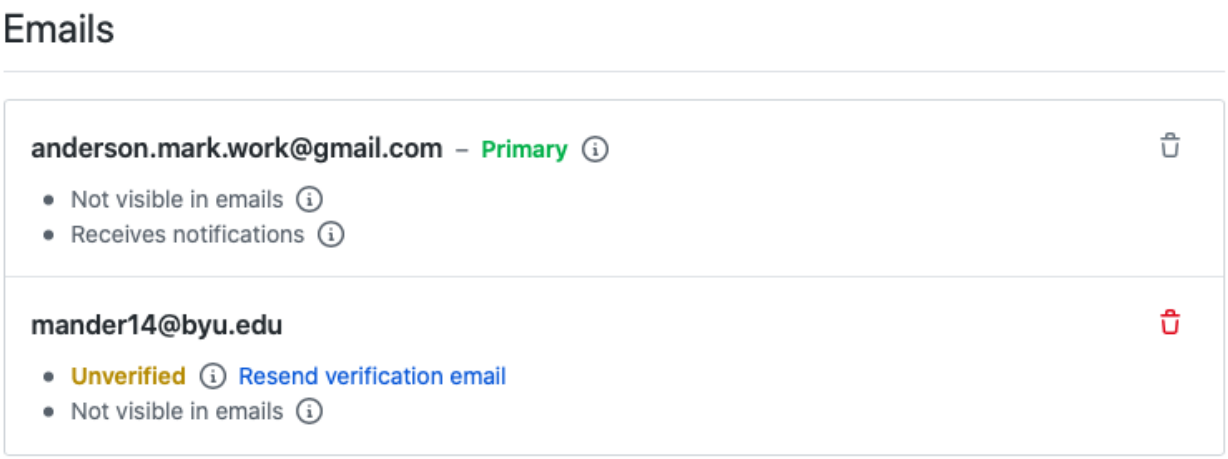
Add an email address, you may see the following after adding it:



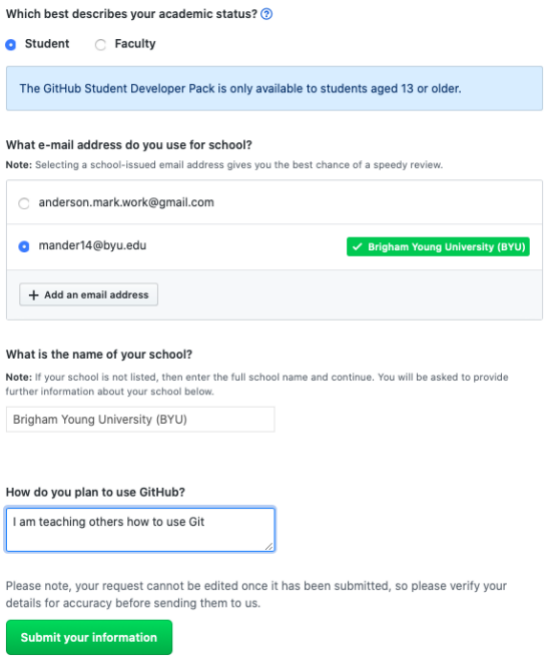
Right-click on the “your GitHub email settings” text and open it in a new tab. You will be brought to the following page:



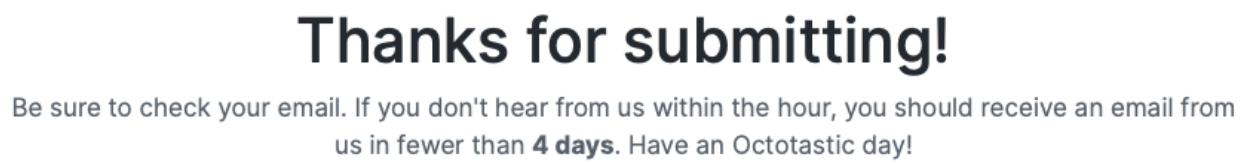
Add your school email address in the “Add email address” field. You will see the following at the top of the page:



GitHub will send a verification email to your school email address. Once you verify the email address, you're ready to get the student developer pack. Return to the tab for the student developer pack, or just re-navigate to it on https://education.github.com/pack. Once filled out, the application should look something like this:



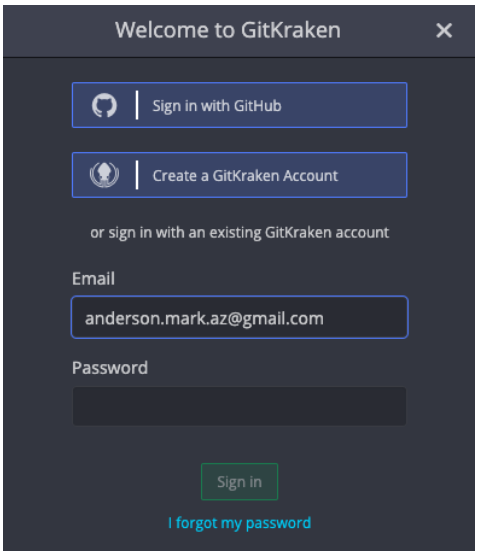
Click on the green "Submit your information" icon and you will see the following notice on the next page:



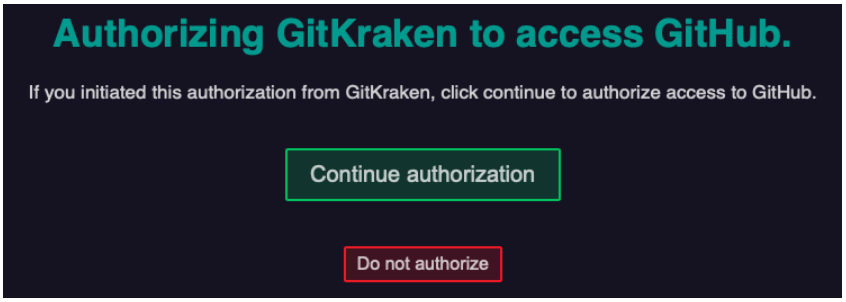
It looks like it can take a while for your request to be accepted, but while making this tutorial the email saying that the offer was accepted was received about a minute after submitting. The email will have lots of information for you to explore. Welcome to the student developer pack.

5.1.2 Getting GitKraken Pro Through the Student Developer Pack

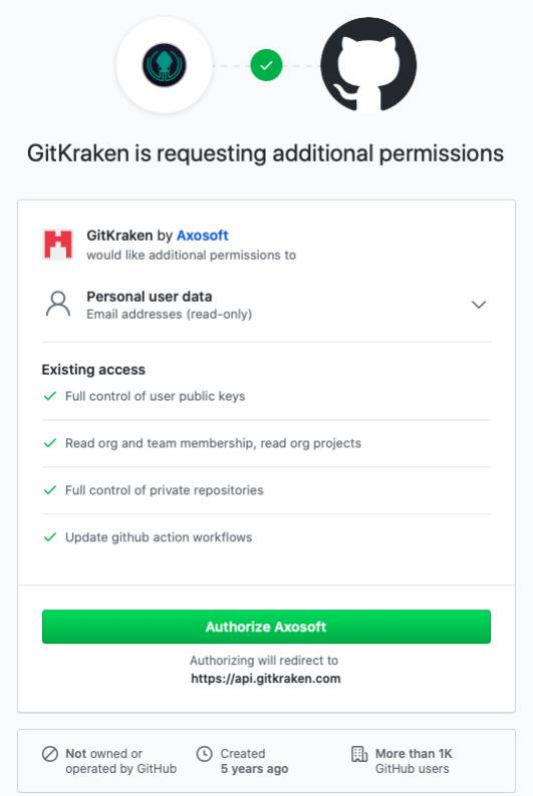
First, sign into GitHub on your browser. After this, open GitKraken and go to "File > Sign into a different account". You will see the following popup:



Select the "Sign in with GitHub" option. You'll be brought to the following web page:



Click on the green "Continue authorization" icon. You will be brought to the following page:



Click on the green “Authorize Axosoft” icon (Axosoft is the company that makes GitKraken).

You may have to wait a few minutes for the authorization to be successful, but once it is successful you will see a screen that says it was successful. Afterwards, open GitKraken and in the bottom right corner you should see something like this:

Where the orange “FREE” label has now been replaced with the orange “PRO” label. While making this tutorial it was about thirty minutes before the label changed to “PRO” You may need to restart GitKraken. Congratulations, you now have a Pro version of GitKraken. As of the present writing, this Pro version for students is advertised as a one-year deal.

5.2 Resetting your Git Credentials in the Terminal

5.2.1 Windows

5.2.2 macOS

5.3 Further Learning

5.3.1 Git in General

5.3.1.1 Udemy.com

Udemy is an extremely useful website that provides online video tutorials on a huge array of topics ranging from playing the guitar to in-depth MATLAB tutorials on machine learning. I learned the bulk of how to use Git from a single online class. This class teaches exclusively in the terminal and is about six hours long. The teacher is extremely knowledgeable and teaches in great clarity, explaining every command. By the end of this class you will be able to use Git very comfortably in the terminal. Here is the link to the class:

https://www.udemy.com/share/101tLwAEIccF9TR3wF/

If the price seems high, don't stress. Udemy seems to do deals about every other week where the classes are offered for around $10. Just check back daily until you see that it's on sale.

https://www.youtube.com/watch?v=uR6G2v_WsRA – brings up committing almost like saving your place in a video game

https://www.youtube.com/watch?v=1h9_cB9mPT8 – Git for noobs

http://try.github.io – interactive tutorial

http://blog.martinfenner.org/2014/08/25/using-microsoft-word-with-git/ - Using git with word

https://michaelstepner.com/blog/git-vs-dropbox/ - git for researchers

https://neurathsboat.blog/post/git-intro/ - git for scientists

https://milesmcbain.github.io/git_4_sci/index.html - git course for scientists

https://biz30.timedoctor.com/git-mecurial-and-cvs-comparison-of-svn-software/ - comparison between mercurial, SVN, and Git

http://blogs.nature.com/naturejobs/2018/06/11/git-the-reproducibility-tool-scientists-love-to-hate/ - Git's hard at first, but after that it's worth it. Includes links to tutorials

https://ealdent.wordpress.com/2008/11/30/10-reasons-to-use-git-for-research/ - 10 reasons to use Git (one bad word)

5.3.2 GitHub

https://www.youtube.com/watch?v=-YVIpI4ucQw
https://www.youtube.com/watch?v=w3jLJU7DT5E

https://www.youtube.com/watch?v=BCQHnlnPusY

5.3.3 GitLab

https://www.youtube.com/watch?v=Jt4Z1vwtXT0

5.3.4 GitKraken

GitKraken has a whole YouTube channel where they both teach you Git and help you learn how to use GitKraken. The videos are very well-made and are found at:

https://www.youtube.com/channel/UCp06FAzrFalo3txskS1gCfA

5.3.5 The Terminal

https://www.youtube.com/watch?v=oK8EvVeVltE&list=PLRqwX-V7Uu6ZF9C0YMKuns9sLDzK6zoiV&t=0s

6 Glossary

.gitignore
Branch (master branch)
Clone
Commit
Merge (merge tool)
Pull
Push
README
Repository (remote repository)
SSH Key
Staging area