Introduction to Git and GitHub for Writers Workbook

February 23, 2019

Peter Gruenbaum

Table of Contents

Schedule

9:00 Introduction
9:15 Intro to Git and GitHub
10:00 Git add, commit, and push
10:45 Break
11:00 Git checkout, pull, and merge
12:00 Branching
12:45 Q&A

Preparation

For Mac Users:

If you do not have a text editor, then download Atom (for free) at: https://atom.io/

Git is already part of the Mac OS, so no need to download it.

Get a free GitHub account at http://github.com.

For Windows Users:

You can use Notepad as your text editor if you wish, but it may not update to changes automatically. If you want something more sophisticated, download Notepad++ at: https://notepad-plus-plus.org/. Click on **Download** on the left.

You will need to download the software called "Git Bash". Follow these steps:

1. Go to https://git-scm.com/downloads and click on Windows
2. Run the executable. It will ask several questions before installing.
    a. If you don't see an editor you like to use in the dropdown list, you can choose Vim as your editor
    b. Choose **Use Git from Git Bash only**
    c. Choose **Use the Open SSL library**
    d. Choose **Checkout Windows style, commit Unix style line endings**
    e. Choose **MinTTY**
    f. Leave all extra options checked.
    g. Click **Install**
3. You can check if it installed by running Git Bash. Type **git --version** and you should see your git version.

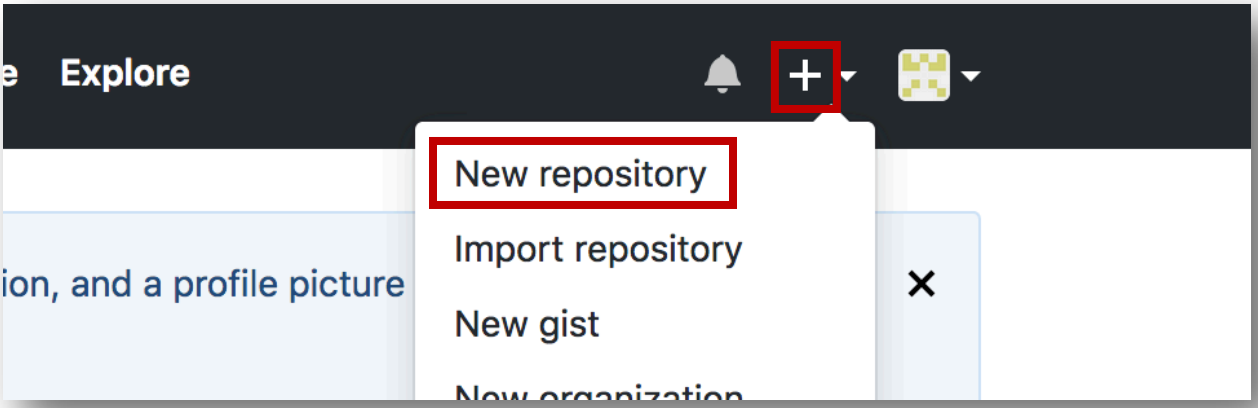
Get a free GitHub account at http://github.com.

Exercise 1: Create a repository. Use the command line.

Create a repository

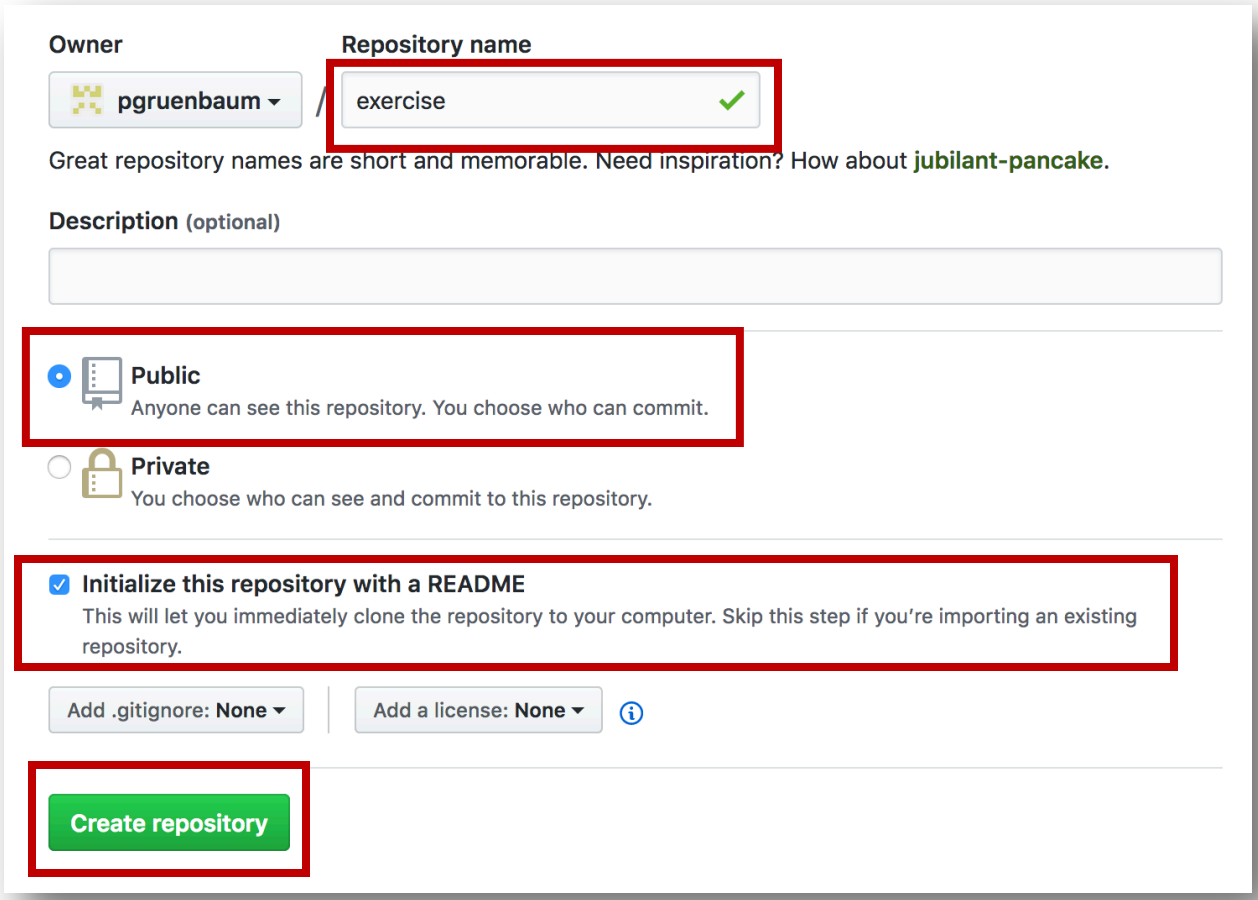
First, you need a repository in GitHub. Follow these steps:

1. Open a browser and go to http://github.com
2. Log in (if you haven't already)
3. From your home page, click the **+** sign at the upper right and choose **New repository**



4. For repository name type **exercise**.
5. Leave as public. (You can only have public repositories with free accounts.)
6. Check **Initialize this repository with a README**
7. Click **Create repository**

See screenshot on next page.



Practice using the command line

Now switch to your terminal. For Mac users, this means opening the Terminal application. For Windows users, this means opening your Git Bash application. You will now type a series of commands.

1. First, make your current directory your home directory. (Remember: "directory" basically means "folder".) Use cd (change directory) and then the tilde (~), which means "home". After each command, hit the return key.

```
cd ~
```

2. You can see where your home directory is by asking it where the current directory is. The command is pwd (print working directory)

```
pwd
```

3. Typically, the home directory is in the Users directory, under another directory which is your user name. Make a new directory called "learngit". Use mkdir (make directory).

```
mkdir learngit
```

4. Do an ls (list) command to see that the directory exists. You will see other files and directories there as well.

```
ls
```

5. Now change directory into your new directory.

```
cd learngit
```

6. List the current directory. You'll see that you are now inside the learngit directory.

```
pwd
```

7. Do a list. The directory is empty, so you shouldn't see anything at all.

```
ls
```

8. To change back up one level, try this command. The two periods mean "up one level".

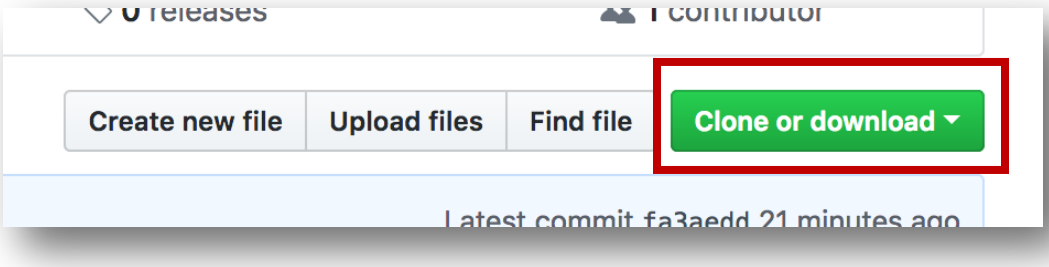
```
cd ..
```

9. Do a `pwd` to see that you are back in the home directory.
10. Move back into the `learngit` directory. This time just type `learng` and then tab. It should finish the rest for you. (Don't worry about the slash at the end.) Press return to execute the command.
11. Do a `pwd` so that you know you are there.

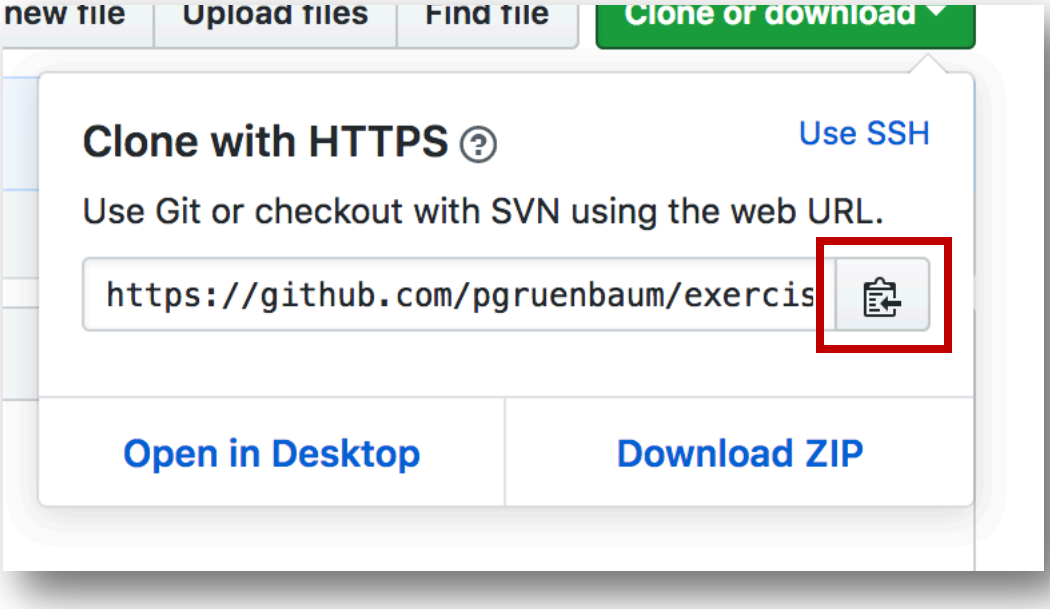
Clone the repository

Now that you are in your new directory, let's clone the repository. Follow these steps:

1. Go back to github.com and click the **Clone or Download button**.



2. You will see a URL. Click the clipboard icon to the right of it. This copies the URL into your clipboard.



3. Return to your terminal. Type the command `git clone`, a space, and then do a paste (Command+V for Mac, right-click, then **Paste** for Windows.) Your command should look something like this:

```
git clone https://github.com/pgruenbaum/exercise.git
```

4. Press return and it will get to work cloning your repository so that you have a local copy.

```
Cloning into 'exercise'...
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
```

5. Type `ls`, and you should see a new directory called “exercise”.
6. Change directories into the exercise directory.
7. Type `ls`, and you will see one file, called README.md, which we had generated automatically when we created the repository.

Congratulations! You cloned your first GitHub repository!

Exercise 2: Create, add, commit, and push

In this exercise, you will create a new file, which you will add to your Git repository, and then push it up to GitHub.

Create

Follow these steps:

1. Open your text editor and create a file with three lines in it:

```
Git and GitHub are used for version control.
Git manages your files.
GitHub hosts your files.
```

2. Save it as **sample.md** in the **exercise** folder that's in your **learngit** folder. The "md" suffix means that it's a Markdown file. Markdown is a simple markup language. You don't need to know Markdown for this exercise.

Staging

The file is created, but that's it. That means that it's unstaged. To stage it, go to your terminal app and:

1. Type `ls`. This lists all the files. You should see `sample.md` listed.
2. Type the following. This tells you the state. Note that it lists `sample.md` as an "untracked file", meaning unstaged.

```
git status
```

3. Type the following to stage all unstaged files in this directory and any subdirectories. Note the period at the end.

```
git add .
```

4. Type `git status` again. Now notice that `sample.md` is listed as a file to be committed.

Committing

Let's say you've decided your file is fine and should be committed. Follow these steps:

1. Commit all staged files using the message "Added sample doc". The command looks like this:

```
git commit -m "Added sample doc"
```

2. It returns some information about what was committed, basically saying that `sample.md` was created.

Pushing

So far, all of the work you have done is local. The push command uploads your changes to GitHub. Follow these steps:

1. Go to github.com and verify that there is only one file: README.md.
2. Return to your terminal and type this command to upload the file. The “origin” means to push it back to where you cloned the repository.

   ```
   git push origin
   ```

3. You’ll see a bunch of information about how the changes were uploaded. As long as there is no error message, then it succeeded. Return to github.com and refresh the page. You should now see **sample.md** listed.
4. Click on **sample.md.** GitHub can display Markdown, so you should see your three sentences. Because there were no spaces between the lines, Markdown interprets them as one paragraph.

Exercise 3: Making changes

Now let's see what happens when we make changes to the file.

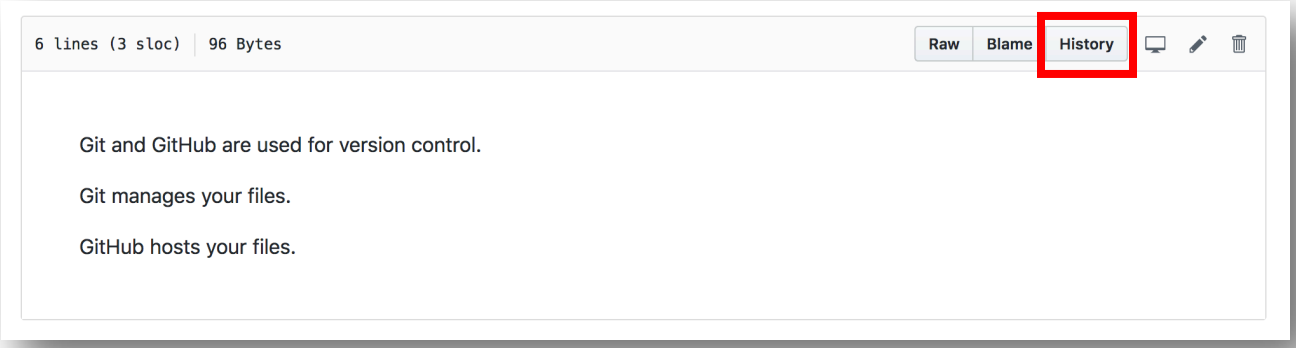
1. In your editor, change your file by adding blank lines between each of the lines, then save the file. In Markdown, this means that each line will be its own paragraph.

```
Git and GitHub are used for version control.

Git manages your files.

GitHub hosts your files.
```

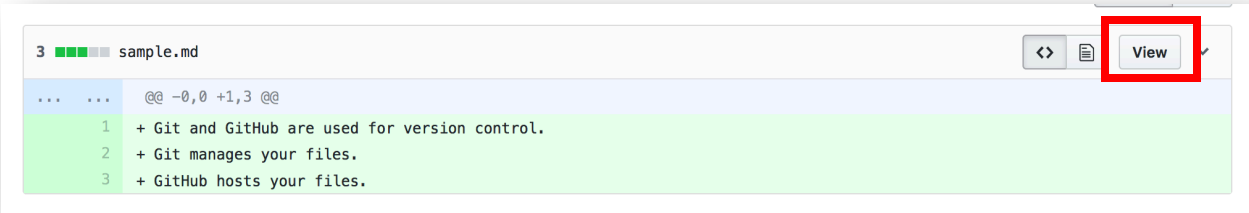
2. Go back to your terminal and type `git status`. Note that the file is listed as not staged, and that `git status` returns that it has been modified.
3. Now stage it. You can do this with:

```
git add .
```

4. Type `git status` again. Now notice that `sample.md` is listed as a file to be committed.
5. Commit all unstaged files (which is just sample.md) with a message "Added blank lines to sample". See last exercise on how to do this.
6. Type `git status` once more. Note that it says, "Your branch is ahead of 'origin/master' by 1 commit." This means that you have a local commit that's not on GitHub yet.
7. Now upload the commit to GitHub. Again, refer to the last exercise to see how to do this.
8. Go back to github.com and refresh the page that shows example.md. Now each line is its own paragraph.
9. One of the cool things about GitHub is how easy it is to go through your history of commits. Click the **History** button.



10. Now click **Added blank lines to sample**. This shows all the changes you made for that commit. You'll see green lines with + in them. This indicates a new line that's been added with this commit.
11. Go back one page and click **Added sample doc**, the first commit you made.
12. Click **View**.



13. Now you can see the file as you committed it that first time, where it's all one paragraph.

Exercise 4: Going Back in Time

It's one thing to see your previous commits in GitHub, but we can also use Git to see the previous commits locally.

1. Start by typing `git log`. This will show you a history of all the commits you've made.

```
Peters-MacBook-Pro:exercise Peter$ git log
commit 9dc45b86b630fe100e7a12d626f67459f6d1d49b (HEAD -> master, origin/master,
origin/HEAD)
Author: Peter Gruenbaum <peter@sdkbridge.com>
Date:   Thu Aug 16 17:09:05 2018 -0700

    Added blank lines to sample

commit 09882bee5f6cfab189556748185c4518527bbdeb
Author: Peter Gruenbaum <peter@sdkbridge.com>
Date:   Thu Aug 16 14:19:28 2018 -0700

    Added sample doc

commit fa3aedda078e73c4e9751ed6e1d8ef09e7725fd2
Author: pgruenbaum <peter@sdkbridge.com>
Date:   Tue Aug 14 14:07:10 2018 -0700

    Initial commit
```

2. That's a lot of information, especially if you have a long history. Try this command just to see the IDs and commit messages: `git log --oneline`

```
Peters-MacBook-Pro:exercise Peter$ git log --oneline
9dc45b8 (HEAD -> master, origin/master, origin/HEAD) Added blank lines to sample
09882be Added sample doc
fa3aedd Initial commit
```

3. Let's go back in time to when you made the commit "Added sample doc". That will be the original file with three lines and no spaces in between. Select and copy the ID that is in front of the "Added sample doc" comment. Then put it at the end of a `git checkout` command. See the screenshot below to see how I did it. Notice that it says it moved HEAD (the commit you are currently working on) to the "Added sample doc" commit. Also note that it warns you that you are "detatched", meaning that you shouldn't make changes at this commit.

```
Peters-MacBook-Pro:exercise Peter$ git checkout 09882be
Note: checking out '09882be'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b <new-branch-name>

HEAD is now at 09882be... Added sample doc
```

4. Now go back and look at your editor. The file has magically changed to be your previous version, with no blank lines in between each line.

Note: If you are using Notepad, it may not automatically refresh. You may need to re-open the file to see the changes.

5. Type `git checkout master` to bring yourself back to the present. Verify that the editor shows the blank lines again.

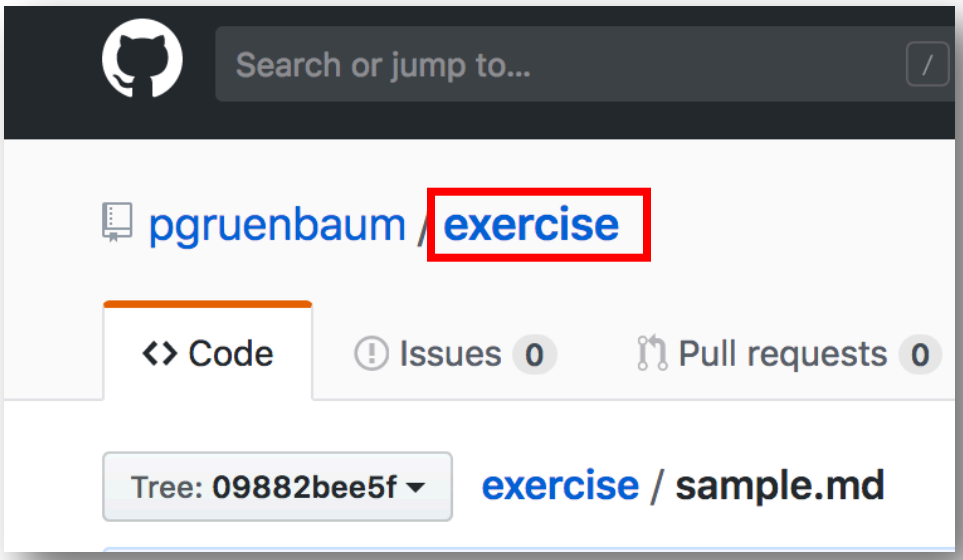
Exercise 5: Pulling

One of the most important uses of Git is that it is a tool for collaboration. We can simulate collaboration by having you make changes on GitHub, as if those changes were done by another member of your team.

Simple Pull

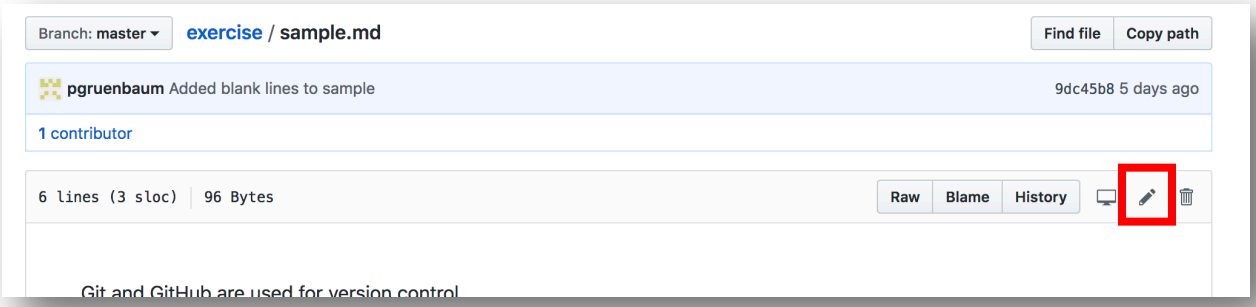
First, make a change directly in GitHub.

1. Go to your browser, where GitHub.com is open.
2. After your username, click on **exercise**. This will bring you to the top level of the repository.

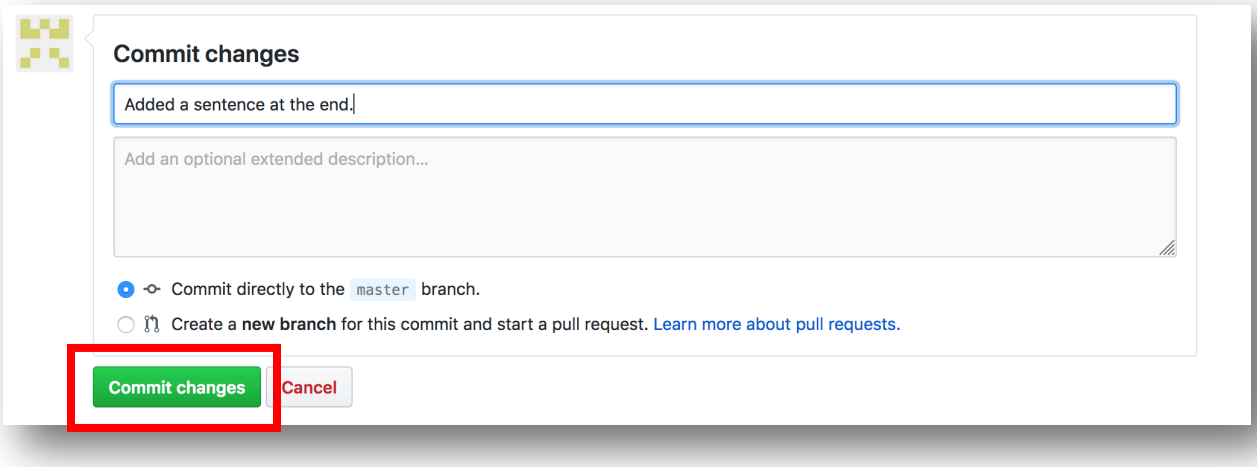


3. Click on **sample.md**.
4. Click on the edit pen icon to edit the file directly.

Note: It is generally *not* recommended to edit files directly in GitHub. You should make your changes locally and then upload them using push.



5. Add a sentence at the end of the third line, such as “But there are alternatives to GitHub.”
6. Scroll to the bottom and add a commit description, such as “Added a sentence at the end.” Then click on **Commit changes**. This will add a commit to the repository.



7. Return to your editor. The editor does not show the changes. This is because the change is remote (on GitHub), but not local. To download the changes, go to your terminal and type:

```
git pull
```

8. You should see a message that one file has changed. Return to your editor. You should see the sentence at the end of the last line now.

Pull with Merge

Git was able to tell that you hadn't made any changes since the change was made on GitHub, so all it had to do was replace your version with the GitHub version. Simple! But what if you had made changes? Let's try it.

1. Go back to GitHub, and edit the file again. This time, add another sentence at the end, such as "GitHub is the most popular." Again, add a commit description and commit.
2. Return to your editor. Change the first line to: "You can use Git and GitHub for version control." (Might as well get rid of that passive voice!) Save.
3. Open your terminal and type `git pull` to grab the changes from GitHub.
4. What happened? You should be getting an error. This is because your changes need to be committed in order to do a pull. This ensures you are at a good stopping place before you try to bring in someone else's changes.
5. In your terminal, type the following to commit your changes and then do a pull:

```
git add .
git commit -m "Changed first sentence"
git pull
```

6. Git now notices that it needs to merge your changes with changes that were made in GitHub. Fortunately, your local version changed the first line, and the GitHub version changed the last line, so it's clear what Git should do, which is use your change for the first line and use GitHub's change for the last line. If you are on a Mac or you chose Vim as your editor, the terminal will show you a file to edit using the Vim editor. If not, you might see another editor appear.

7. You have the opportunity to add some more information here, but usually you don't really need to. It's usually fine to leave it as the default message, which just indicates that a merge has taken place. Note that Vim is an old, non-intuitive editor. All you need to know to save and quit the editor is to type these keyboard strokes and hit return:

   ```
   :q
   ```

8. Go back to your editor. You should now see both your local changes and the GitHub changes.
9. Return to the terminal and type `git status`. Note that it says that your branch (that is, local copy) is ahead by 2 commits. The first commit was where you changed the first line. The second commit was the merge.
10. Type `git push origin` to upload the changes to GitHub. Refresh your GitHub page to see that remote file looks just like your local file.

Pull with Conflict

I mentioned before that Git was able to tell that you had changed the first line locally and the last line on GitHub, so it knew which lines to take when merging. What if you had changed the same line? In this case, GitHub does not know what to do, and it calls this a "conflict". Let's generate one so that you can see what happens.

1. Go to GitHub and edit your file. Change the second and third sentences, adding the words "locally" and "remotely" so that it now says:

```
You can use Git and GitHub for version control.

Git manages your files locally.

GitHub hosts your files remotely. But there are alternatives
to GitHub. GitHub is the most popular.
```

2. Add a commit message and commit.
3. Return to your editor and change the last line so that it flows a little better:

```
GitHub hosts your files. Note that there are alternatives to GitHub,
but GitHub is the most popular.
```

4. Commit your changes and then do a `git pull`
5. Note that it says: "Automatic merge failed; fix conflicts and then commit the result." Go back to your editor. Note that it has updated to say:

```
<<<<<<< HEAD
GitHub hosts your files. Note that there are alternatives to GitHub,
but GitHub is the most popular.
=======
GitHub hosts your files remotely. But there are alternatives to
GitHub. GitHub is the most popular.
>>>>>>> 63565ee5d7ca5ce9bbad9c1a2d1cb167d7854242
```

6. Git is showing you the conflict as two versions: the first version (HEAD) is your local version, and the second version (6356…, which is the ID of the GitHub commit) is the GitHub version. Edit these lines so that (1) you have added the word "remotely" to your version and (2) so that you have eliminated all of the <<<< ==== and >>>> characters.

```
GitHub hosts your files remotely. Note that there are alternatives
to GitHub, but GitHub is the most popular.
```

Note: Certain editors, like Atom, add user interfaces to make this easier. Click the "Use me" button for the HEAD version and then add remote.

7. Save. Add your changes, commit, and then push. Refresh your file and you should see your changes up on GitHub.

Congratulations! You resolved your first conflict!

Note: Often merges will have multiple conflicts in each file, and multiple files that have conflicts. You simply go through this process for each of them before committing.

Exercise 6: Branching

For this exercise, you are going to create a branch and make changes on that branch, create a pull request, and merge it.

Create a branch

Follow these steps to create a branch and move between branches.

1. Use the following command to create a branch called “exciting”:

   ```
   git checkout -b exciting
   ```

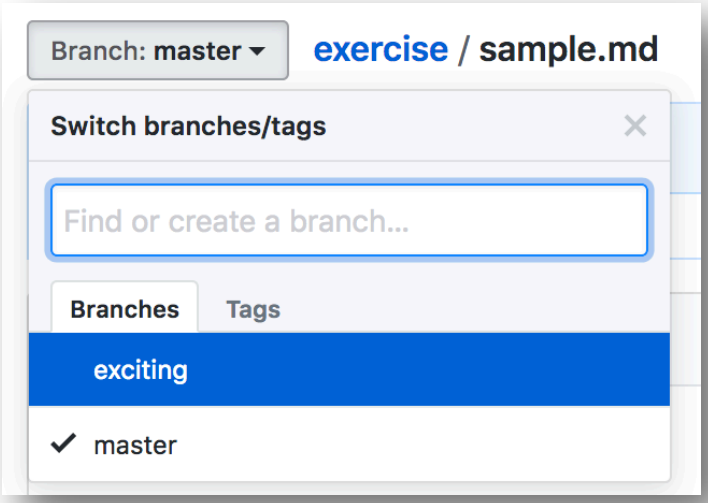
2. To list all your branches and see which one you are on, type:

   ```
   git branch
   ```

3. You should see two branches: master and exciting. Note that exciting has a star by it, meaning you are on that branch. Switch back to the master branch:

   ```
   git checkout master
   ```

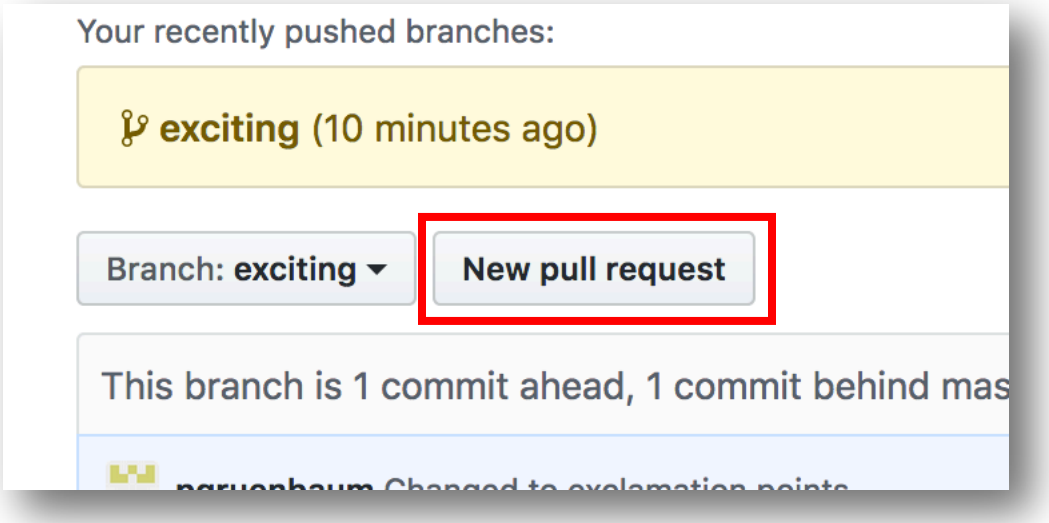
4. Type `git branch` to verify you are on the master branch.
5. Type `git checkout exciting` to return to the exciting branch.
6. Type `git status` to verify you are back on the exciting branch.
7. In your editor, change all of the periods to exclamation points.
8. Add and commit your changes. Note that this commit is now on the exciting branch, but not the master branch.
9. Switch back to the master branch and then take a look at your editor. Note that your changes are no longer there.
10. Switch back and forth and notice how the changes update themselves.
11. Return to the exciting branch. Do a `git push` to upload your changes.
12. Note what happened: Git doesn’t know which branch to push to, because the exciting branch doesn’t exist on GitHub yet. Copy and paste the suggested command, which will create the branch on GitHub and upload the code to it.
13. Return to GitHub and refresh the page. Click on **Branch: master** and you will get a dropdown that shows all available branches. Note that exciting is one of them. Select it, and you will see your changes with the exclamation points.



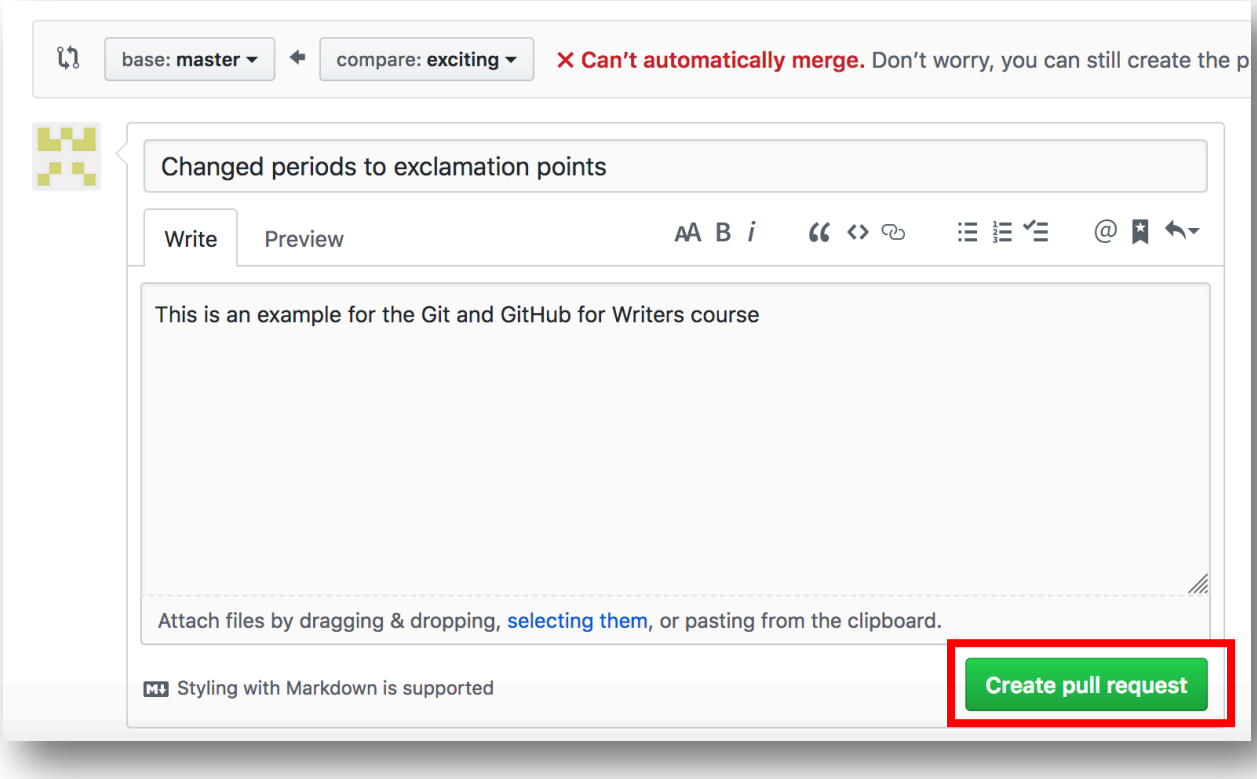
Pull request

Very often you will make your changes on a branch and then want to merge those changes into the branch your originally branched off from. You can do this with a pull request. But before we do this, let's create a conflict by changing something in master before doing the pull request.

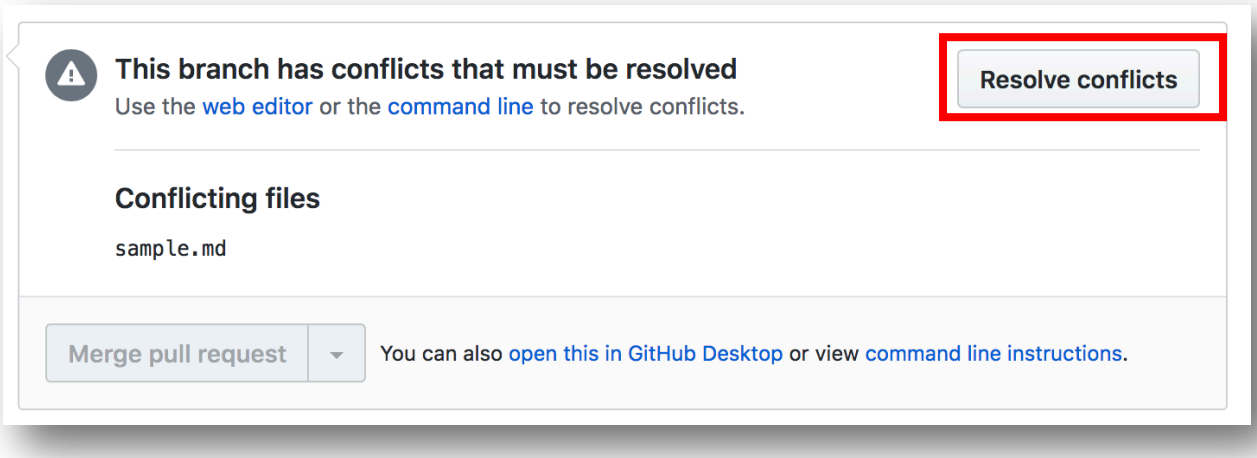
1. In your terminal, switch to the master branch.
2. In your editor, change the first period to a question mark.
3. Save, add, commit, and push.
4. In GitHub, click on **exercise** after your username to bring you back to your repository's home.
5. Choose the **exciting** branch. Then click on **New pull request**.



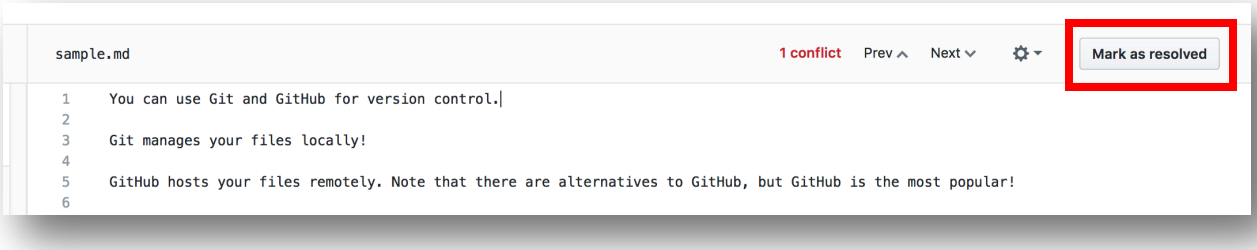
6. Notice that it says, “Can’t automatically merge” in red. This is because of the conflict we created. But we can still create the pull request. So, if you like, you can add a comment that describes the pull request (that’s optional). Then click on **Create pull request**.



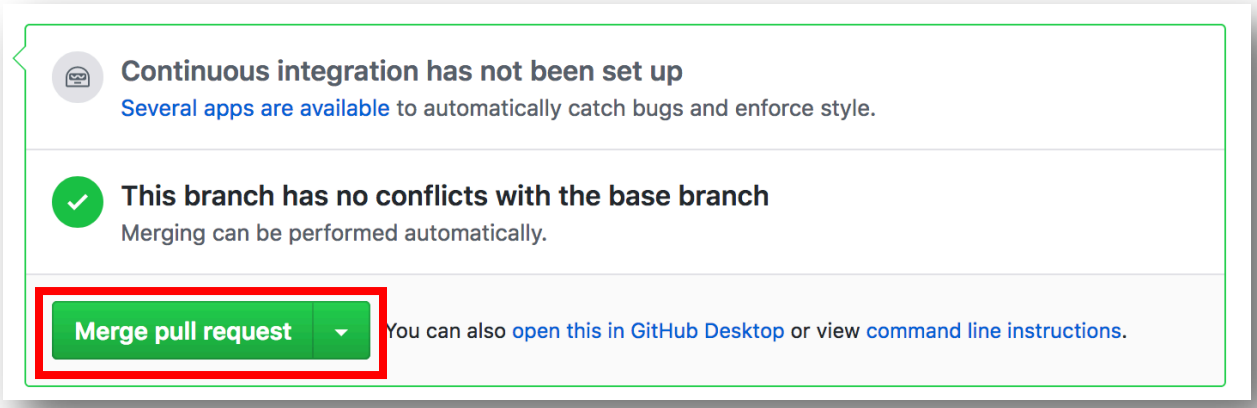
7. Next, you’ll see that it says you have conflicts that must be resolved. Click on **Resolve Conflicts**.



8. Edit the conflict to be however you like. (Perhaps it’s time to put a period back.) Make sure to remove all the <<<< ==== >>>> lines. Once you have, you can click on **Mark as resolved**.



9. If there were more than one file with conflicts, it would then present you with the next one, since there is only one, it now presents you with a button **Commit merge**. Click it.
10. Now that the merge is committed, you actually have to accept that pull request. Click on **Merge pull request**. Once again, you can add a comment. (The default is the merge comment.) Click **Confirm merge**.



11. Return to your terminal and do a pull. In your editor, you should see the version that you resolved on GitHub.
12. Lastly, return to GitHub. Note that it says "You're all set—the exciting branch can be safely deleted." Since you have merged, you no longer need that branch. Click on **Delete branch**. (This is a good practice so that you aren't storing branches that you'll never look at again, but you want to be absolutely sure that the merge went successfully.)

Discounts for Online Classes

SDK Bridge offers five online courses to learn API Documentation. This workshop covers part of the first and second courses. For taking this workshop, we are offering 50-75% discounts on all five courses using coupon PUGETSTC.

Learn API Technical Writing: JSON and XML
~~$59.99~~ $9.99 with coupon

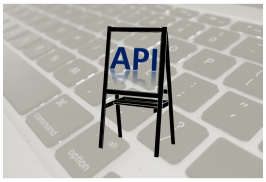
https://www.udemy.com/git-and-github-for-writers/?couponCode=STCONLINE

Learn API Technical Writing: JSON and XML
~~$25~~ $13 with coupon

https://www.udemy.com/api-documentation-1-json-and-xml/?couponCode=PUGETSTC

Learn API Technical Writing 2: REST
~~$40~~ $19.99 with coupon

https://www.udemy.com/learn-api-technical-writing-2-rest-for-writers/?couponCode=PUGETSTC

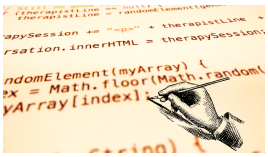


The Art of API Documentation
~~$25~~ $12.99 with coupon

https://www.udemy.com/the-art-of-api-documentation/?couponCode=PUGETSTC

Learn Swagger and the Open API Specification
~~$35~~ $15.99 with coupon

https://www.udemy.com/learn-swagger-and-the-open-api-specification/?couponCode=PUGETSTC



API Documentation: Basic Programming
~~$45~~ $11.99 with coupon

https://www.udemy.com/coding-for-writers-1-basic-programming/?couponCode=PUGETSTC