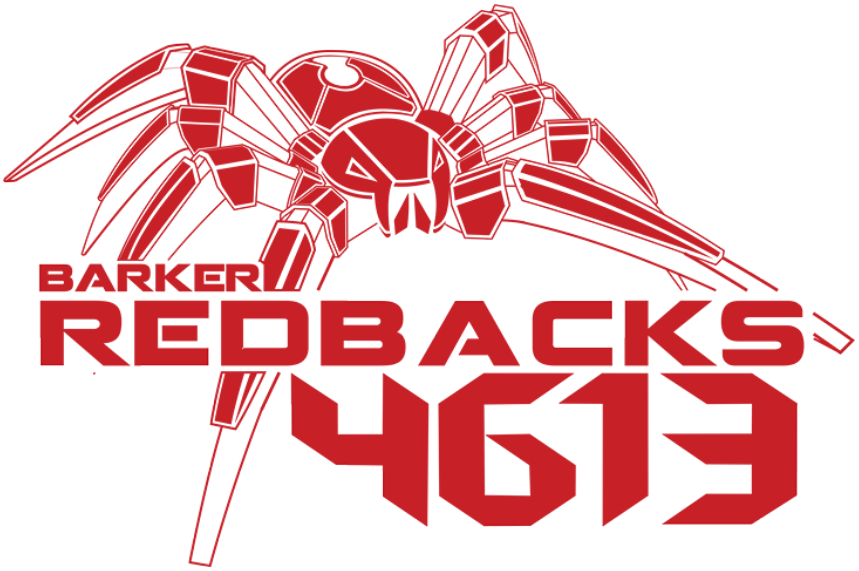

Tom's Git Guide

Tom Schwarz[1]

Janurary 2019
v1.0

[1]with contributions from Ben Schwarz

Contents

1 Introduction

This guide was written to explain how to use git (and GitHub) quickly and easily. While it definitely is not a particularly good guide, I couldn't find one online that explained, practically, how to use git with the right context and simply enough. So I wrote this. For an explanation of how Git works (which you *must* know before you read this (knowing "git commit", "git pull" and "git push" or their equivalent GUI buttons dosen't count)) see Git & Github in plain english (seriously - do read it).

2 Solving merge conflicts (in VS Code)

Merge conflicts occur when git is trying to merge converged histories, and in the process finds both groups have edited the same file. For example, this can occur when:

- You run "git pull" after you have commited changes to a file, and someone else has commited and pushed different changes to the same file
- You merge a branch back to master, after you changed a file in the branch and someone else changed the same file in master
- You are updating your branch to have new changes from master - ie the reverse of above

Luckily solving merge conflicts is much less spooky than it appears:

1. Firstly you will be notified that a merge conflict has occured either by some sort of error message after running ""git merge conflicting-branch" that looks like "CONFLICT (content): Merge conflict in file.txt / Automatic merge failed; fix conflicts and then commit the result." or the following error dialog box if using the GUI:

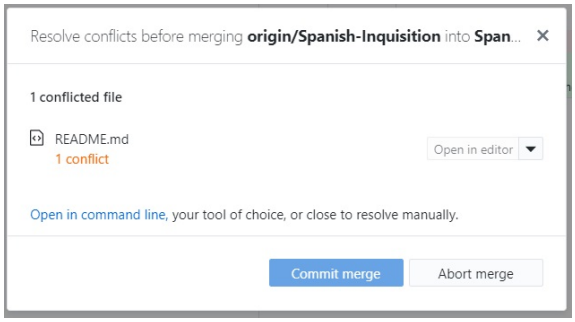

Figure 1: The horror! The horror!

2. To solve the conflict, simply open the conflicting file(s) in your favourite text editor, and make the file look exactly how you want it. In this case we will use Visual Studio Code, but any text editor works:

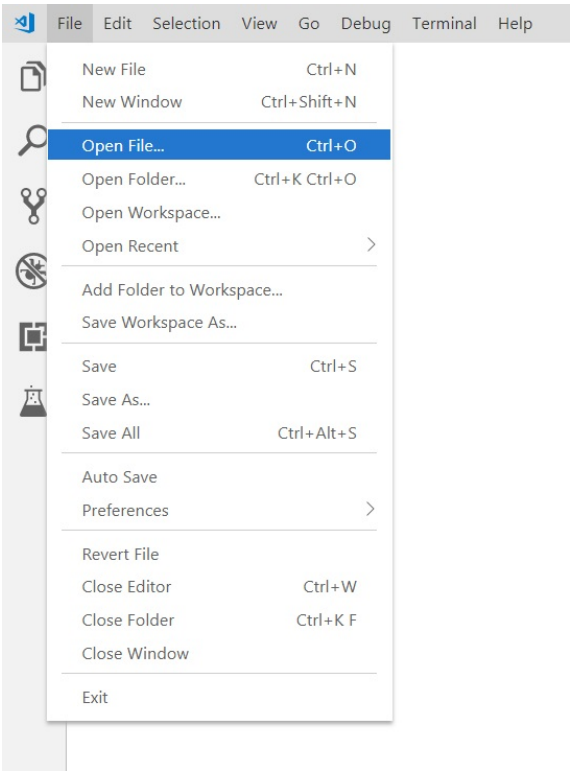

Figure 2: Open the file in VS Code by going File >Open File... and selecting it. It can be any type of (text) file - not necessarily just .java or whatever you commonly edit in VS Code.

The general structure of the file will be:

Preceding context code	
<<<<<<< HEAD	This indicates the start of the conflicting section, and that this first section corresponds to the HEAD commit - ie your commit. VS Code has helpfully labelled this as the (Current Change).
Blah Blah text	
=======	This indicates the dividor between your code, and the alternative commit code
Blah Blah conflicting text	
>>>>>>> 123456	This indicates the end of the conflicting section, and gives the reference hash for the alternative commit code. VS Code has helpfully labelled this as the (Incoming Change)
Ending context code	

Here is an example file:

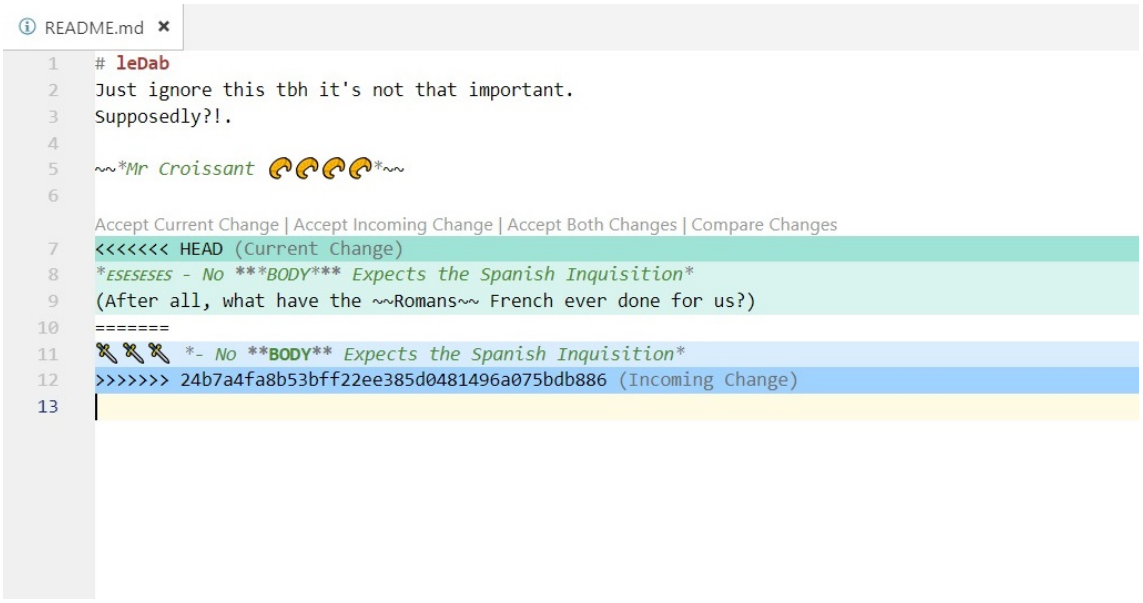

Figure 3: The opened file. VS Code has added the colouring and light grey help text

3. Now edit the file to look how you want. You can use VS Code's handy buttons to choose the current change or incoming change or both changes (or get a side-by-side diff). You can also just ignore it all together and touch up the file yourself. Generally I click "Accept Both Changes" to have it remove the <, = and >stuff and then fix up the code myself - but do what makes sense in the situation

   Note: Often, merge conflicts are minor issues that you can just quicly touch up. However, if you don't communicate with other team members its possible that you can have an actual logical merge conflict -eg you both edited the same function to do something you want. In this case you will need to re-code the function, and the code that uses it, to solve the conflict (even though git might say its been solved before then, you still need to update the dependant code)

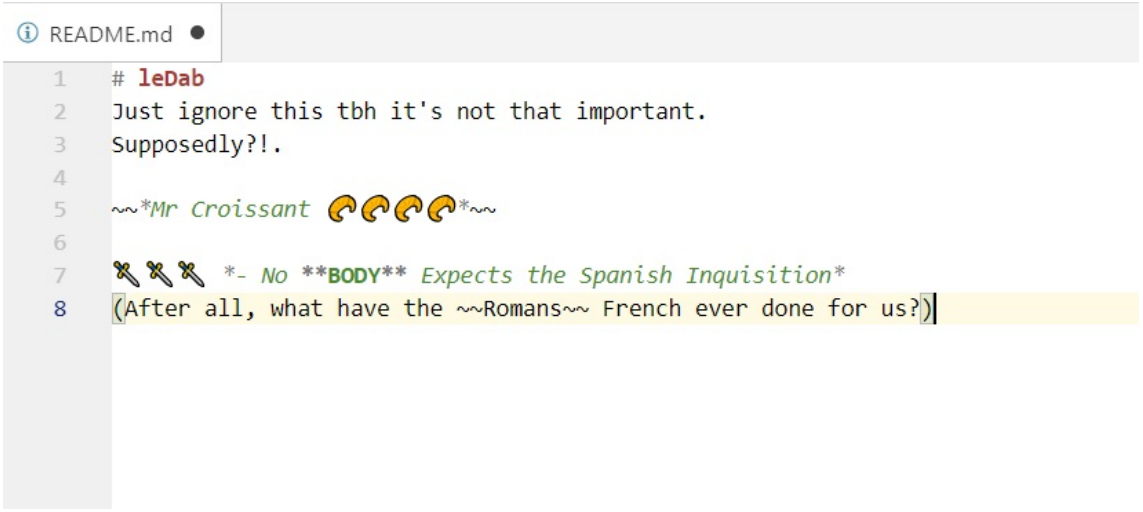

Figure 4: Note that the merge has been fully resolved, and the file looks exactly as we want it.

4. Save and test your code. (This is a very important, non-skippable step. Don't skip it.)

5. Run "git commit" to finish the merge, or go back to the GUI and click "commit merge" (Alternatively you can run git merge –abort to stop merging, and return the

branch to how it was before you tried to merge).

For more help on solving merge conflicts, see this Atlassian Git merge conflicts article.

3 Branching & Pull Request Workflow

This explains a simple workflow, but in extreme detail. A quick outline of this workflow:

1. Create a branch and work on your new changes there
2. Create a pull request to incorporate changes into master
3. Review & approve the pull request, making changes as necessary
4. Delete the branch

3.1 Creating a branch

1. Make sure your git repo is up to date with origin (ie Github) - either by running "git pull" or in the GUI clicking the "Fetch origin" button
2. Create the branch by running "git checkout -b your-branch-name" or in the GUI going "Branch >New Branch" (also available from the "current branch" button next to fetch origin).
3. Choose a suitable branch name, compliant with relevant conventions/guidelines - it should be short and descriptive. You should create a new branch for every self-contained feature - do NOT create large monolithic branches for miscellaneous updates; even if it means having a branch with just one commit.
4. Add this branch to origin either by running "git push –set-upstream origin your-branch-name" (the extra option connects your new branch to a new reflected branch in origin) or hitting the "Publish branch" button in the GUI.

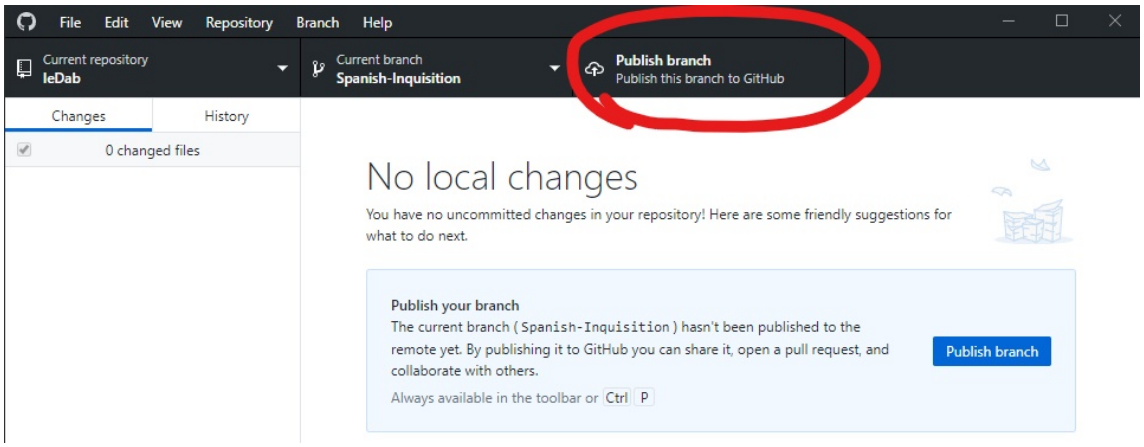

Figure 5: The publish branch button in the GUI

5. Now you can start working on your feature, updating code, making commits and pushing them up to Github as normal!

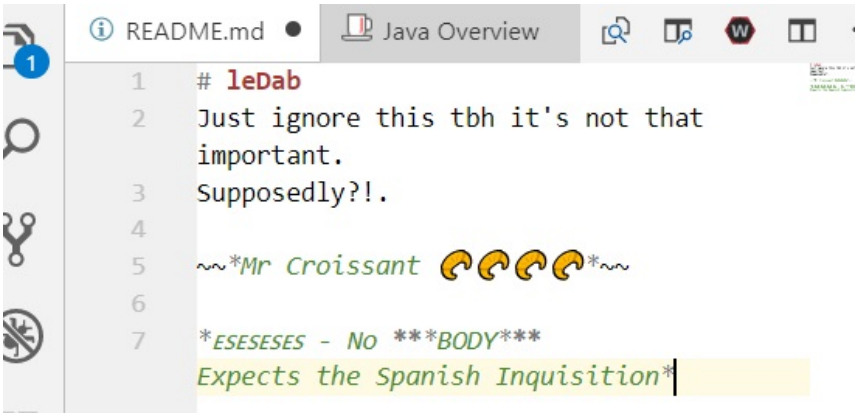

Figure 6: These important files have been modified (&commited) in the branch

3.2 Creating the pull request

1. First, we must encorporate any changes that have been made to master while we have been working on our branch.
   (a) Update the local copy of master using "git checkout master" and "git pull" or in the GUI clicking the "Fetch origin" button
   (b) Next, merge these changes into your branch by checking out your feature branch and running "git merge master" or in the GUI going "Branch >Update from master". You may need to resolve merge conflicts - if so, see the Solving merge conflicts section.
2. sync your merge changes with origin ("git push" or the sync button)
3. Go to the github website for the repository, changing to your feature branch

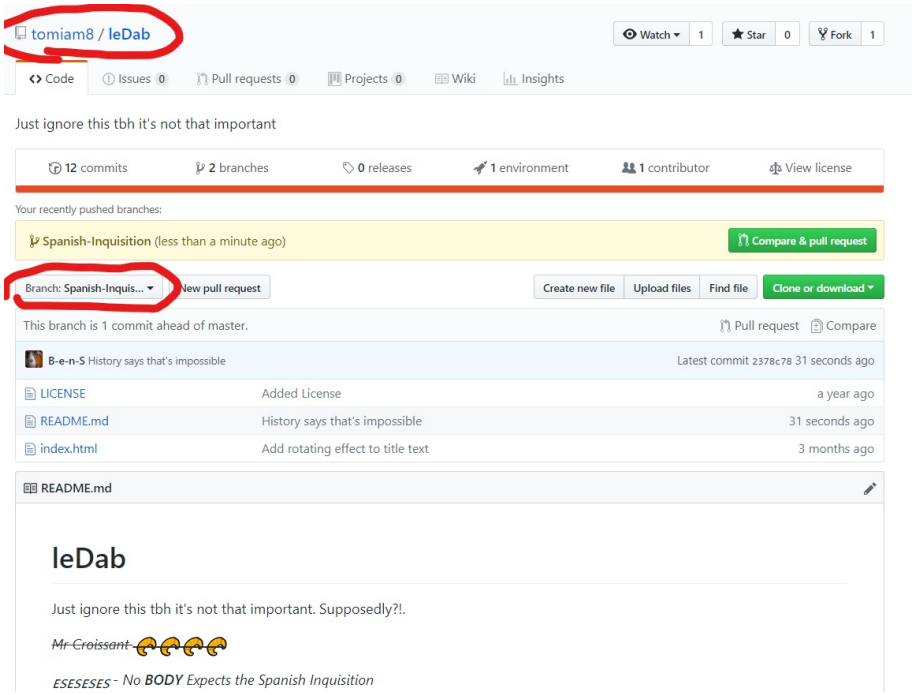

Figure 7: Note that we have selected the repository, and the feature branch

4. Create the pull requst
   (a) Click on the "pull request" button

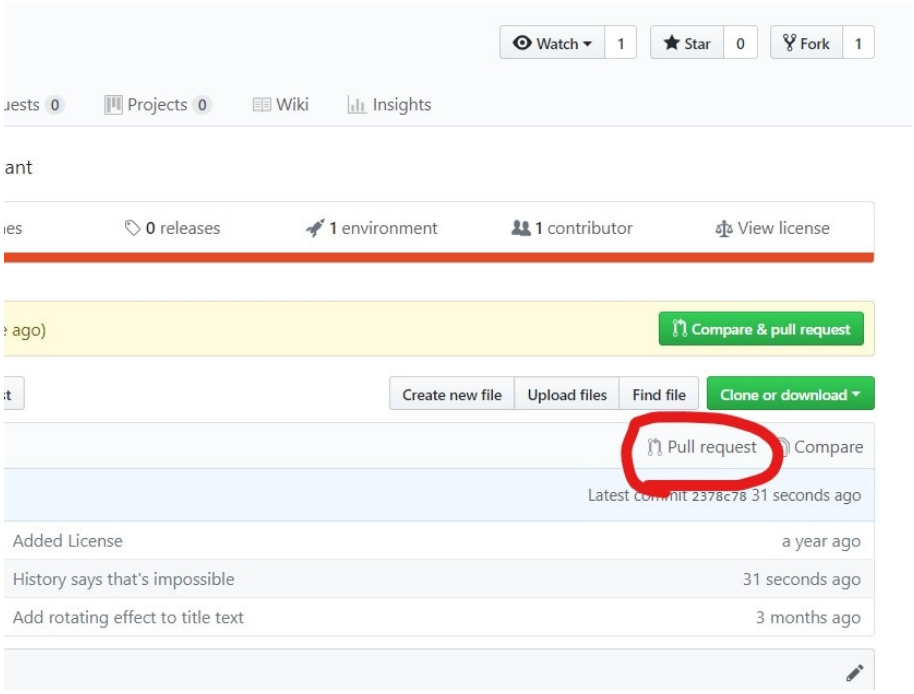

Figure 8: Click the "pull request" button to create the pull request

(b) Check the branches are correct (master on left, feature branch on right)

(c) create a helpful title and comment

(d) Review the code changes shown below

(e) Create the pull request

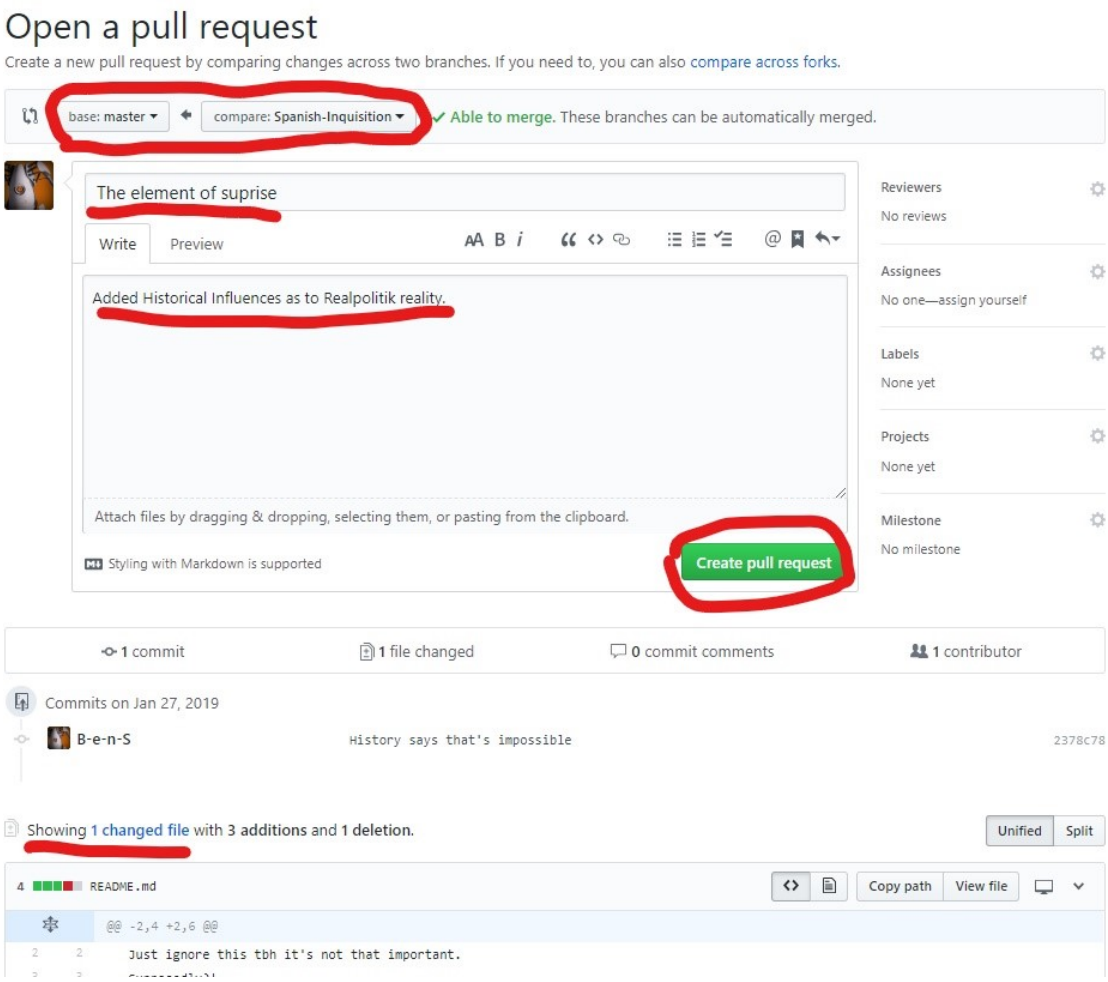

Figure 9: Do the things

5. Now, just get a programming leader to review your pull request!

3.3 Reviewing the pull request

Note that this subsection is only intended for those reviewing the pull request (i.e. doing the code review) - but it can still be useful to read it anyway!

1. Select, open and view the pull request by clicking “Pull Requests” (in the top bar, right below the repository name) >The pull request name:

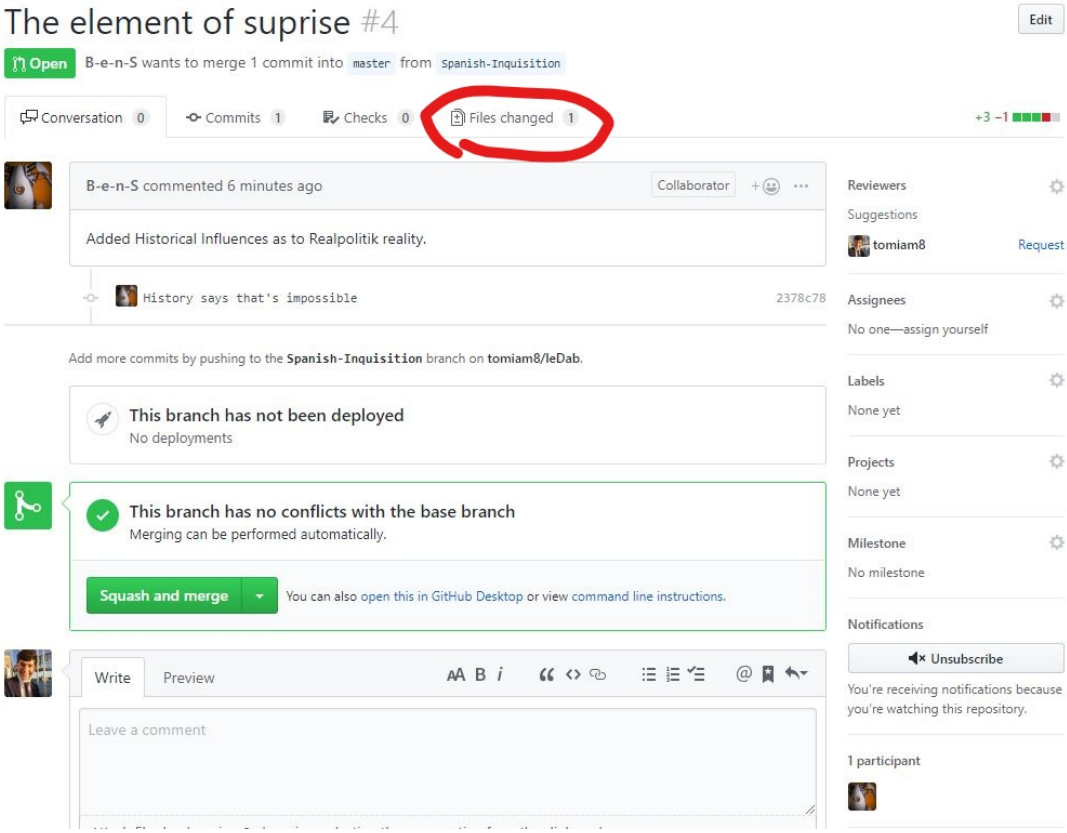

Figure 10: Yay do the things

2. Click "Files changed" to view the code updates in the pull request, and begin the review.

3. Click individual lines (near the line numbers) to add comments to individual lines of the file, and click "review changes" to make a general comment. You can change the "Diff settings" to see more of the file. For individual line comments you can make code suggestions for very simple changes, that can be easily accepted by the pull request proposer. For the general comment you can just make a comment, accept the pull request, or request code changes. For this example, we'll be demonstrating asking for additonal changes.

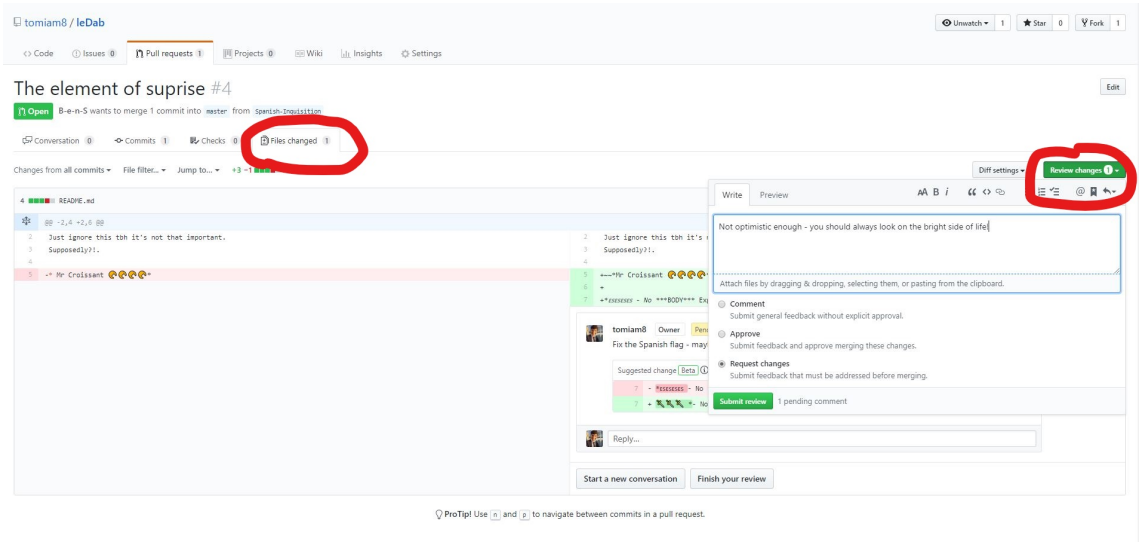

Figure 11: Note how the code has been reviewed

3.3.1 Making changes

This section details how to edit/update the pull request, eg after the code reviewer has requested changes

1. Open the pull request and view the comment

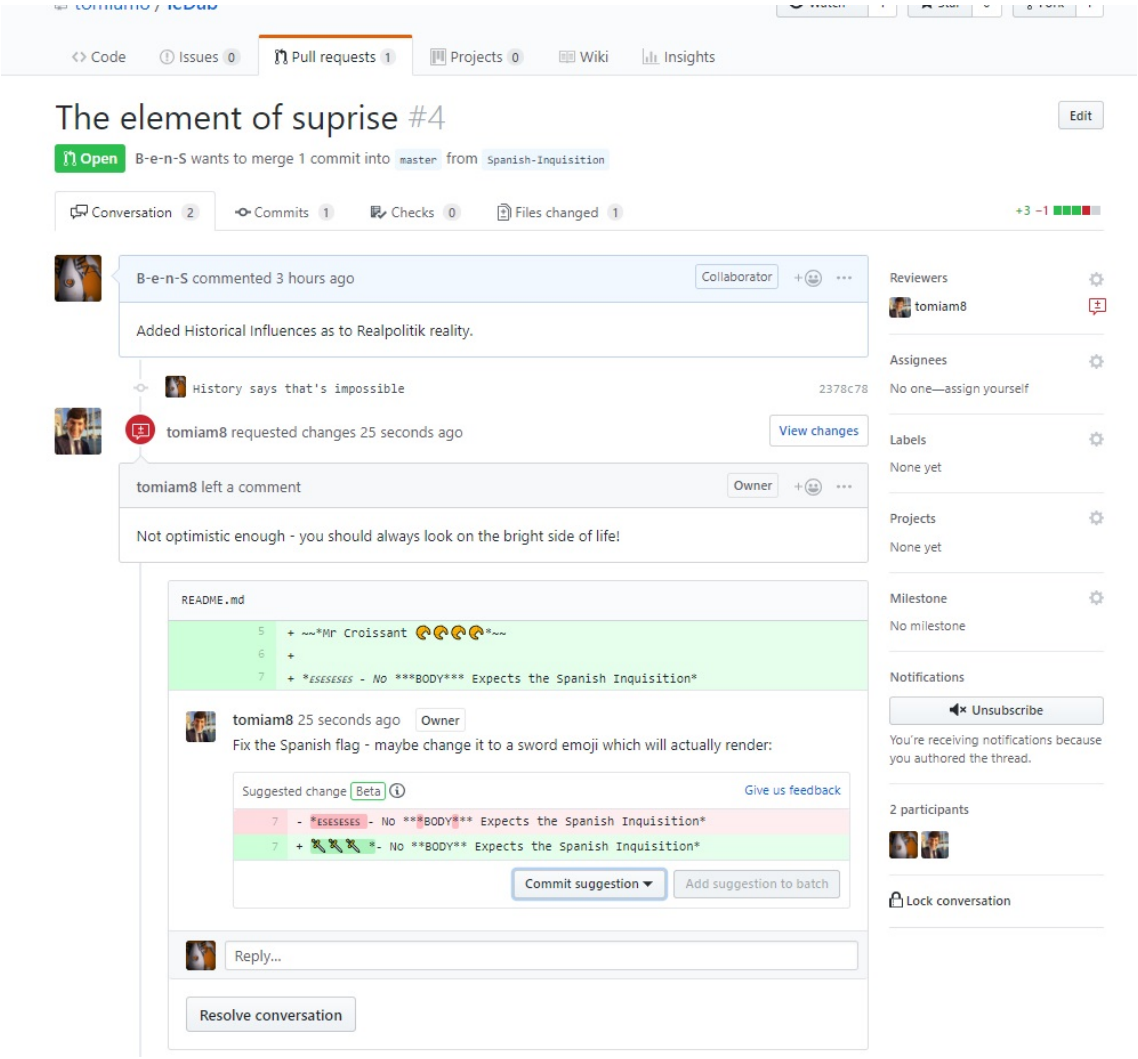

Figure 12: Look at all these lovely comments. Also, note how you can commit the code suggestion instantly!(You will need to re-pull if you do this)

2. Simply create a new commit to the feature branch, and push it to github. Notify the code-reviewer that you have met their tyrannical demands.

3.3.2 Approving the pull request

Again this is for the code reviewer!

1. Open up the code review on Github.com
2. View the changes as before (or, you can view the individual commit to see what changed) and then approve the changes, or create an additional request for changes as before.

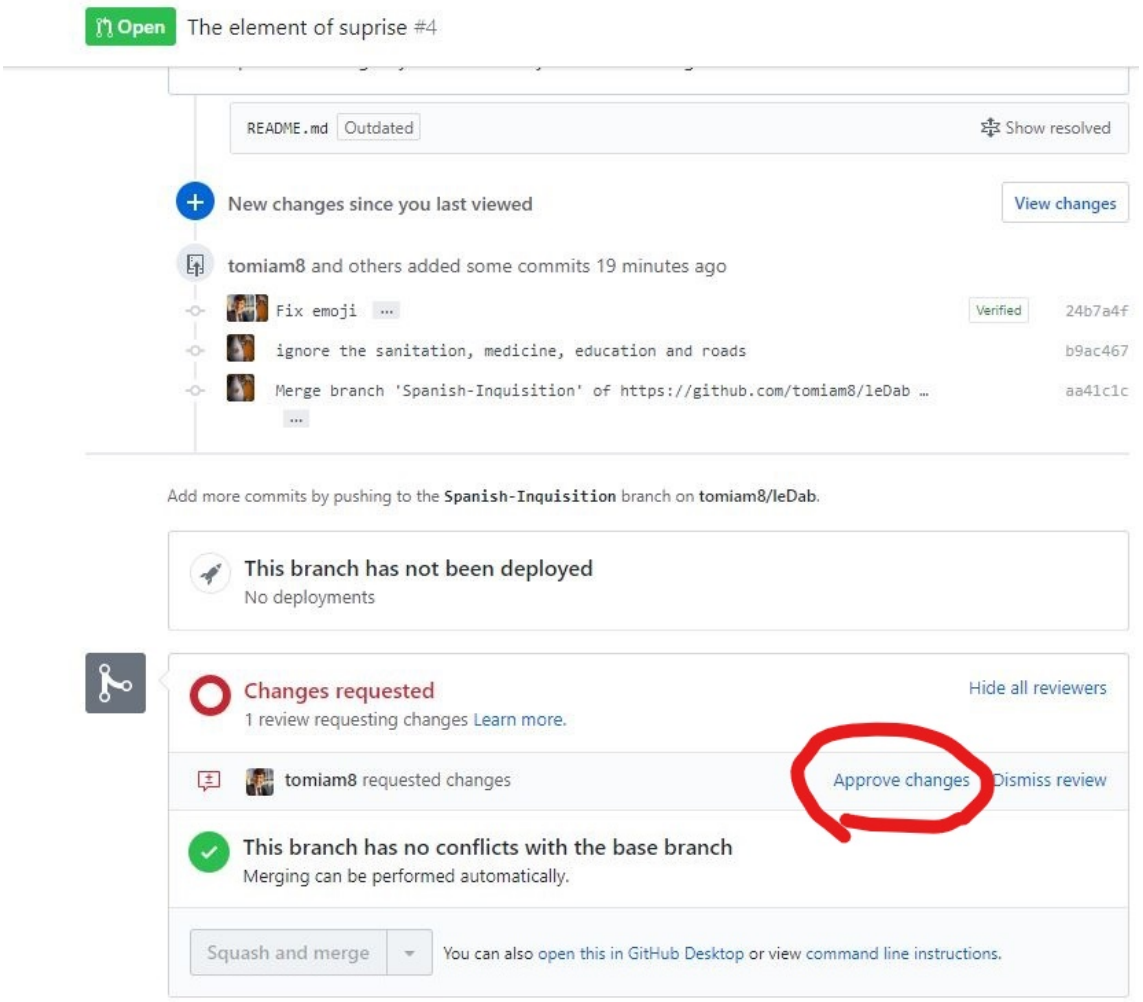

Figure 13: Click the approve changes button to approve the changes

3. Then click "merge" (the big green button) to merge the pull request. You can choose between creating a new "merge" commit, squashing the commits into a mega-commit or rebasing (append the commits to the end of master).

3.4 cleanup

1. As you can see, we have finished the merge and master has the new feature! Now however we must go ahead and delete the existing branch.

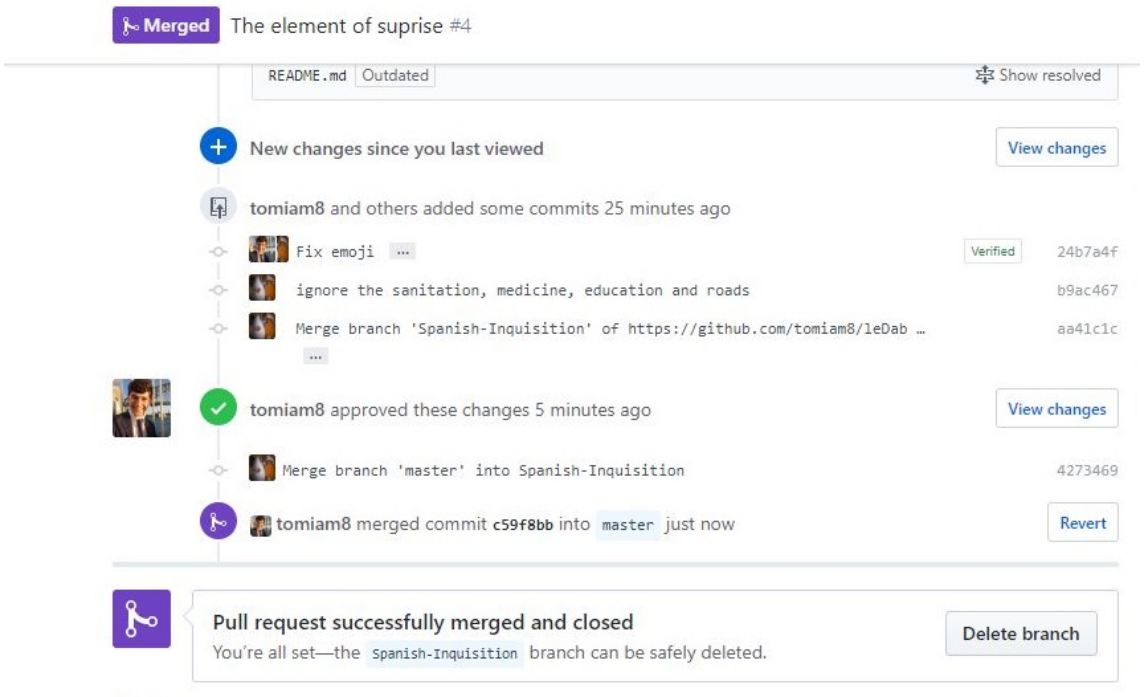

Figure 14: "I have done the deed. Didst thou not hear a noise?"

2. Click the "Delete branch" button on Github to delete the branch there. (You can also use "git push origin –delete your-branch" or on Github click "(number) branches" button (next to the number of commits), then click the bin icon.)

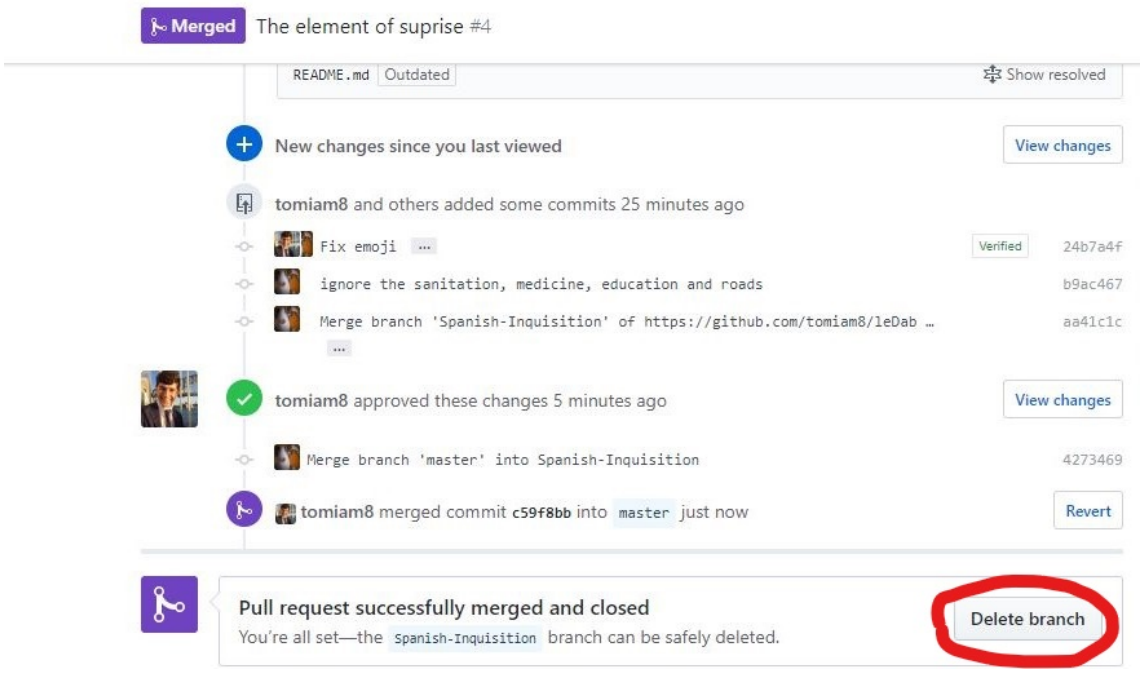

Figure 15: "Easily defeat Birnam forest with this one weird trick."

3. Now delete the branch on your local with "git branch –delete your-branch" or in the GUI go "Branch >Delete"

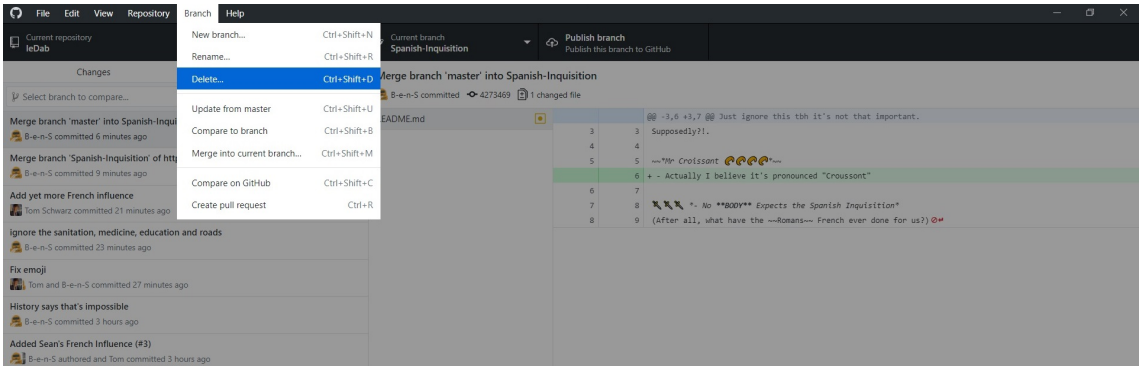

Figure 16: "Don't tell Greenpeace."

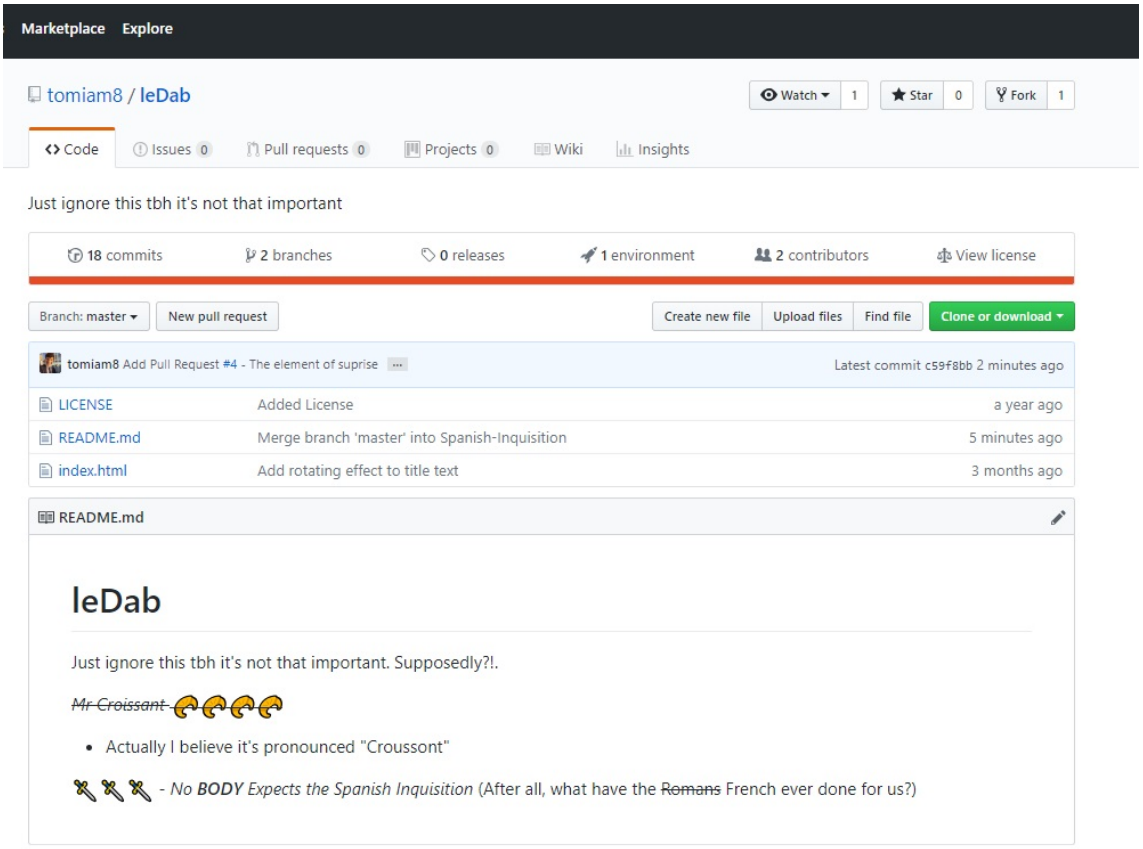

Figure 17: "Random monty python refrences counts as growing the (programming) python cult, right?"

4 license