CodeFirst:Girls Beginners Coding course – Front end Web development

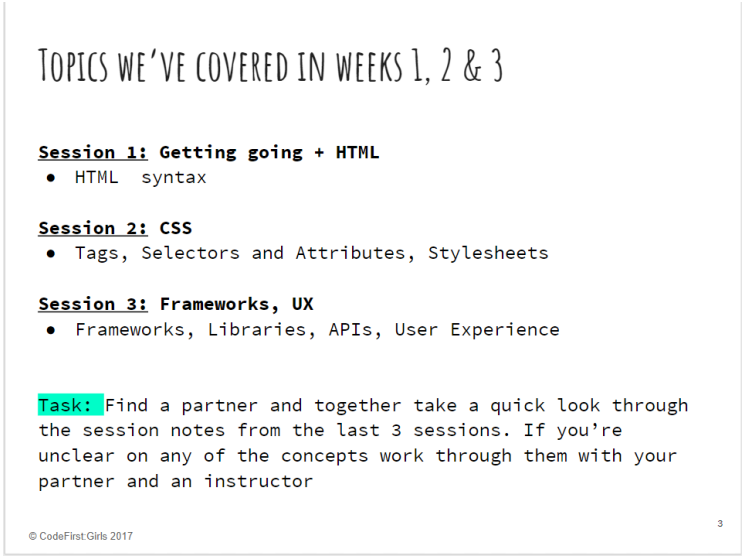
Week 4 - GitHub & Version control

What we'll cover this week:

1. Recap of what we've learnt so far
2. Version Control & Using GitHub
3. Create your group repository
4. Publishing on GitHub Pages
5. How Github Pages Works
6. Conflict scenario!
7. How to avoid and reduce merge conflicts
8. Homework: Create GitHub repository for the course project

Slide 2

Recap session: (20 mins)



Slide 3

Run through the topics covered in previous weeks (HTML, CSS and Frameworks / UX).

Task: find a partner and together take a quick look through the session notes from the last 3 sessions. If you're unclear on any of the concepts work through them with your partner and an instructor

Version Control & Using GitHub

Slide 4

So, what is Version Control and why is it important?

Have you ever worked in a group on a Powerpoint presentation or Word document? What is that process like? Let's not think about Google Slides or Google Docs for the moment.

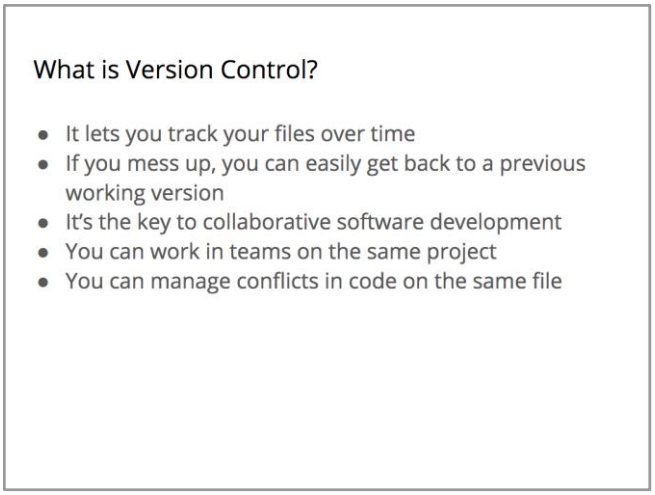
Let's start with two people - you and your partner are presenting on the controversial topic of the future of the UK's relationship with the European Union. You've got a structure for the presentation and started delegating the work; You'll work on the first section discussing the state of the relationship, and your partner will start working on the possible future scenarios.

You both go away and make the slides and handout notes separately and meet again in a week. But when you meet, there's a blocker that's completely irrelevant to your material:

You've decided to make your presentation with a black-background theme, in Calibri, and your partner has chosen a completely different colour scheme, used different fonts, and you've both made about 10-15 slides, organised in a different style from one another, and different types of handouts.

Yes, you could have waited for one person to start making the slides and emailed it over, or decided a "style" from the start, but that would have added an extra meeting and maybe a couple of days of waiting back and forth. Either way, you need to spend at least a couple of hours coordinating, or fiddling with the styling of the slides and handouts.

What if there were a way to automate the merging of your material and choose what to keep and what not to keep?



Slide 5

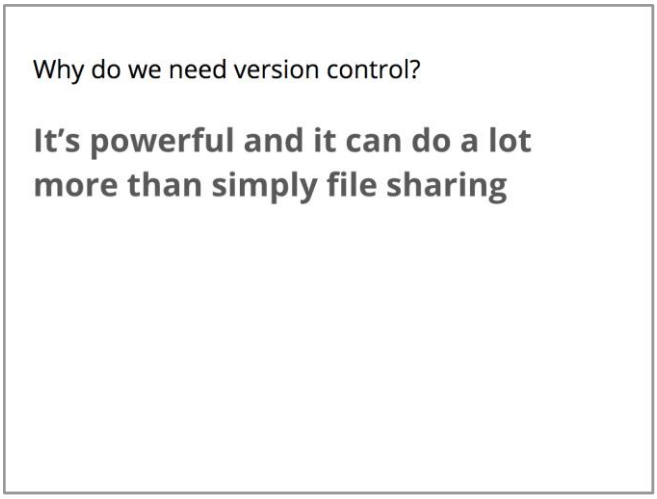
Version Control

The answer is Version Control, also known as Revision Control or Source Control. It lets you track your files over time. But why would you care about that?

Because if you mess something up along the way, you can easily get back to a previous working version.

Version control is the key to collaborative software development.

- You can work in teams on the same project easily and manage conflicts in code on the same files;
- You can work on different versions all at the same time;
- You can decide what you want to keep and what you want to bin at the end.

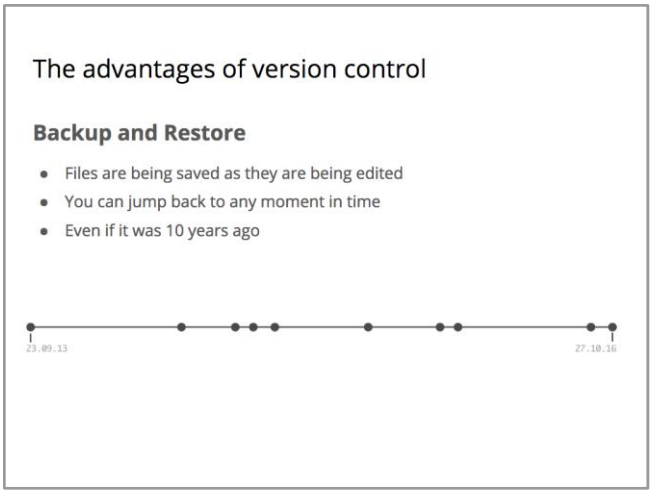


Slide 6

So why do we need a version control system (VCS)?

You may think that sharing a folder on a service like Dropbox or Google Drive might be enough when you are collaborating with other people. Sharing files and folders is quick and simple, but there are a whole bunch of things they can't do.

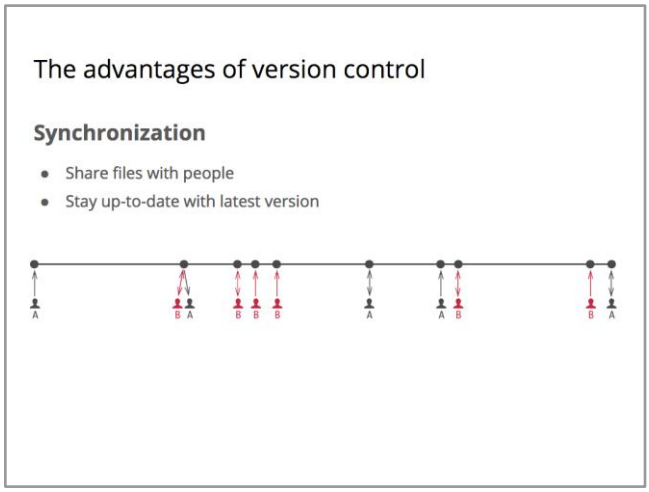
A version control system is much more powerful and can do a lot more than simply file sharing. In fact it is necessary if you want to do any of the following:



Slide 7

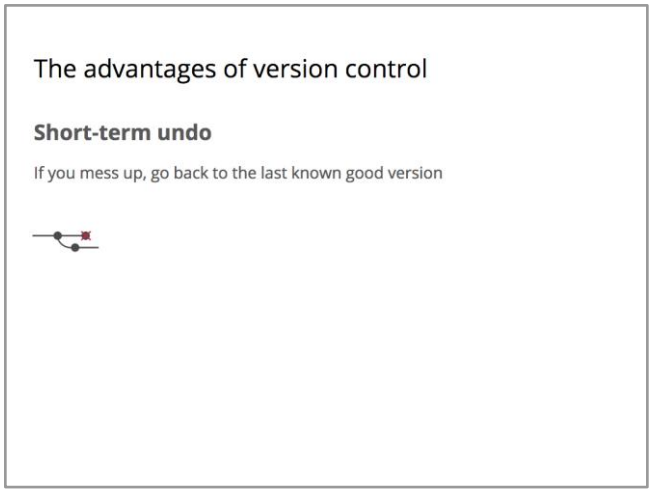
The advantages of version control

- **Backup and restore**: Files are being saved as they are being edited. This means you can jump back to any moment in time. Even if it was 10 years ago.



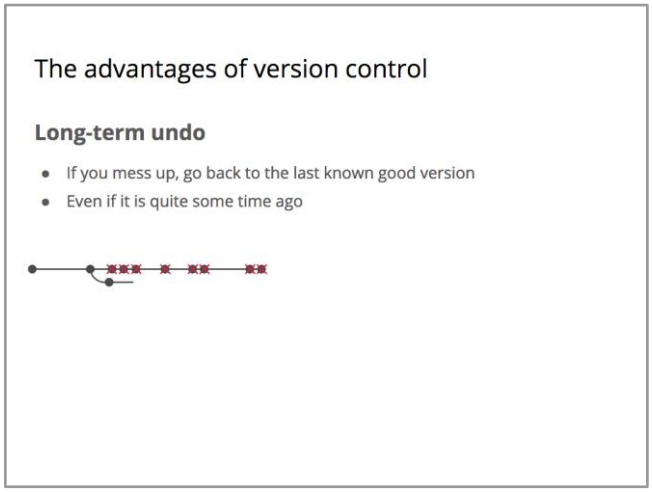
Slide 8

- **Synchronization**: It lets people share files and stay up-to-date with latest version.



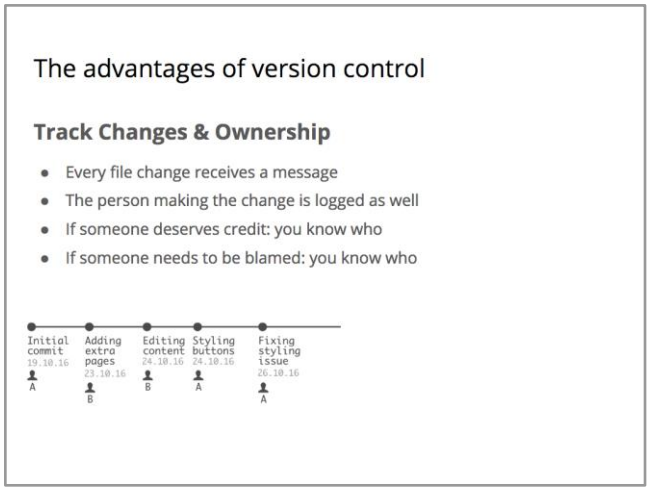
Slide 9

- **Short-term undo**: If you messed up a file you can throw away all the changes you made and go back to the last known good version in the database.



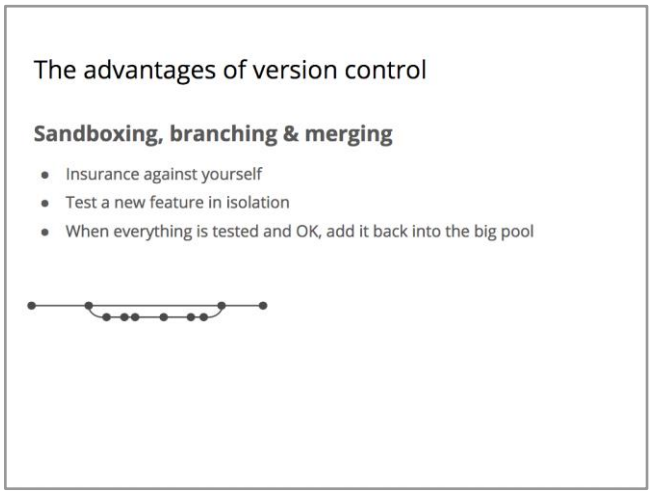
Slide 10

- **Long-term undo**: What if you messed up really badly? Imagine if a change you made a year ago has a bug you only discover today. You can jump back in time to see what changes were made then.



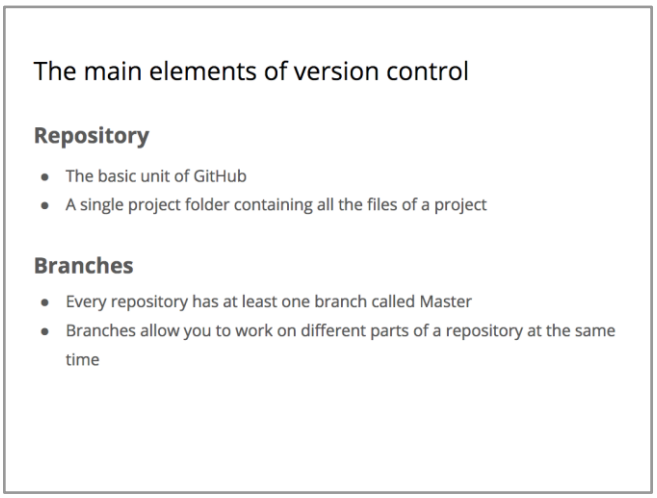
Slide 11

- **Track Changes**: A updates are being made to files you add a message explaining what changes you have made. These messages will make it easier to see how a file has evolved over time and why.
- **Track Ownership**: At the same time the messages are being added to changes, the name of the person making the change is added as well. If someone needs to be blamed for the mess, you know who it was. Or you could give someone credit too for a job well done.



Slide 12

- **Sandboxing**: This is a form of insurance against yourself. When you are making a big change you can do so in an isolated area. That way you can test your work and iron out any mistakes before adding it back into the big pool.
- **Branching and merging**: The same as sandboxing but on a larger scale. You can branch a copy of your code into a separate area to modify it in isolation. Later it can be merged back into the common work area.



Slide 13

The main elements of version control

The version control system we are going to learn about is Git. It's very powerful and popular. Hard line developers use Git on the command line, and that is where Git is the most powerful, but we'll go down a slightly easier route.

Please be aware that Git and GitHub are not the same thing. Git is the version control system and GitHub is a website where you can view coding projects that use Git.

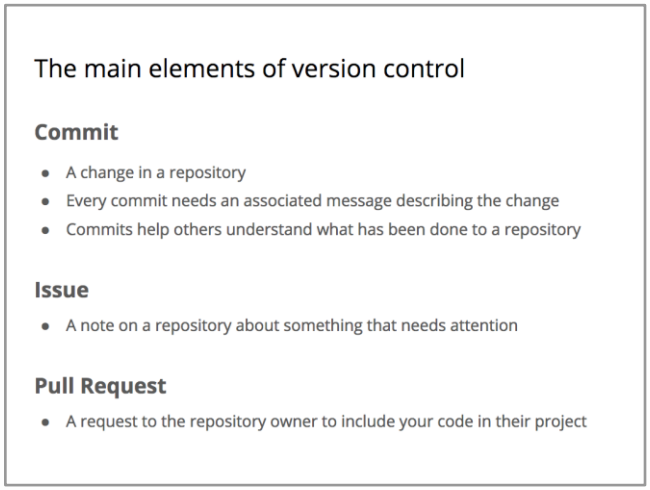
Before we can start using Git we need to understand what the main elements are of a version control system.

Repositories

- A repository is the basic unit of GitHub. Usually it is a single project or folder containing all the files needed for that project. That includes code files, image files, etc.
- A repository should also include a README file and a license. The README file needs to explain what the project is about and how you might want to use it, if it is something public. The license should outline what the rules and restrictions are in case someone wants to use the code.

Branches

- Branching is how you can work on different parts of a repository at one time.
- When you create a repository it will by default have one branch called Master.



Slide 14

Commits

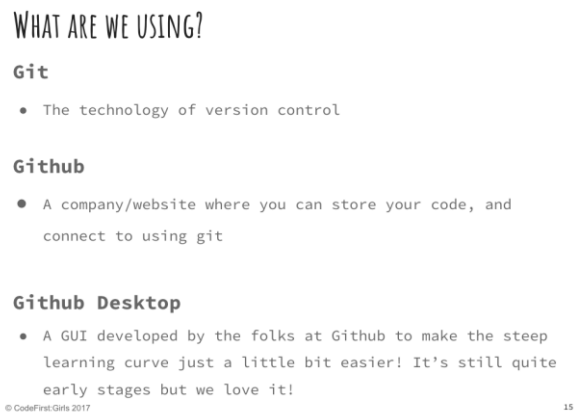
- Each time when you save a change it will be registered. On GitHub this process is called a commit.
- Every time you commit you need to write an associated message. This is a description of the changes you have made and why. It doesn't have to be a full essay, the trick is to keep them short, but descriptive.
- Commit messages will help you and others understand what has been done and why. All the commits of a project will read like the history of the project.

Issues

- An issue is a note on a repository about something that needs attention.
- It could be a note about a bug that needs fixing, a feature request, a question, or anything else.

Pull Requests

- Pull Requests are the heart of collaboration on GitHub. When you make a pull request you are asking the owner of the repository to include your contribution. In other words, you are asking them to merge your work into their branch.

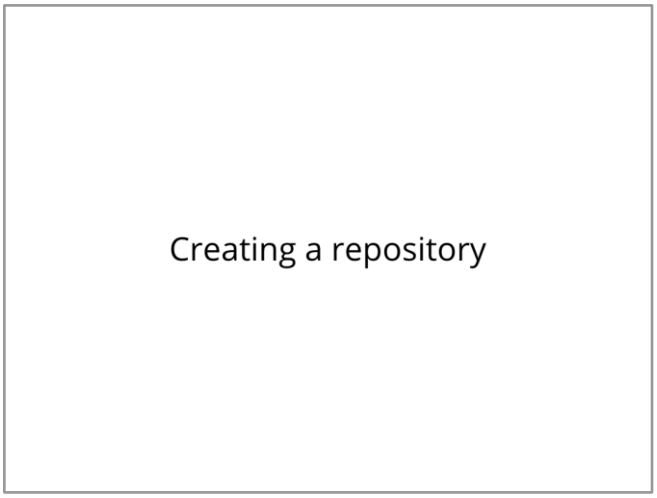


What are we using today?

Git - this is the technology of version control

Github - this is a website (and larger company) wherein you can store your code in their cloud online. You connect and control it using git protocols.

Github desktop - this is a GUI that has been developed by the folks at github to make the fairly steep learning curve a little bit easier.



Create your group repository

Time to roll up our sleeves and get ready for some tricky stuff. We are going to create a repository for your website project. And then we're going to commit some stuff and publish the repository to GitHub.

ACTION

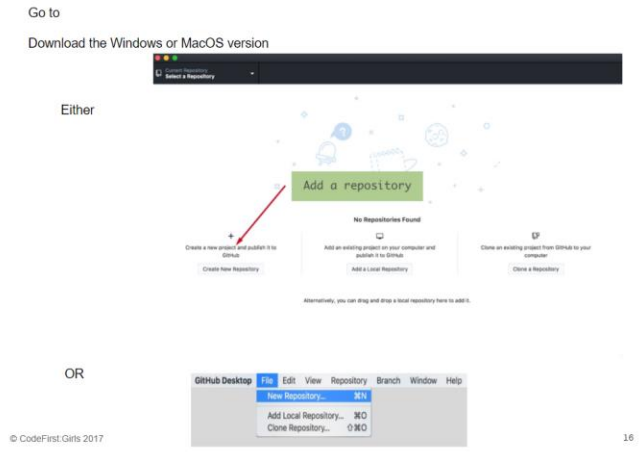
If any of the students have not yet confirmed their email address for their GitHub account then they need to do so now. As long as the email address is not confirmed any communication between the GitHub desktop app and GitHub online will run into problems.

IMPORTANT!

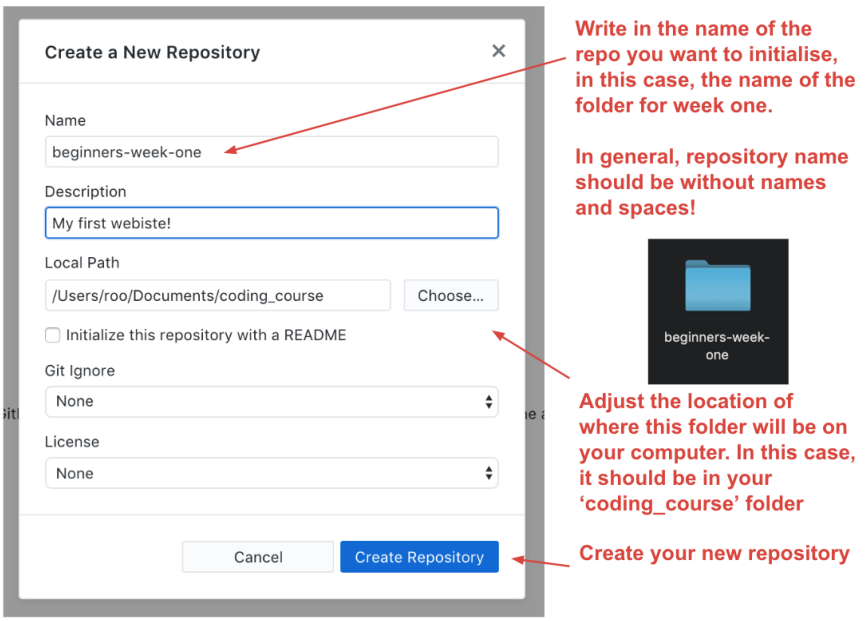
You can run this session either as a demo or codealong. Every student should use the project they started in week one so that they will all have a project published on GH pages by the end of the class. Every student must have got someway to achieving this before moving onto section two, collaboration.
Feel free to adjust this according to the level your class is at, however you must ensure that by the end of this session everyone has initialised, committed and published their website from week two, and started a new project for their group work.

Don't worry too much about the demo videos, they are included for reference for the students when working through the slides themselves.

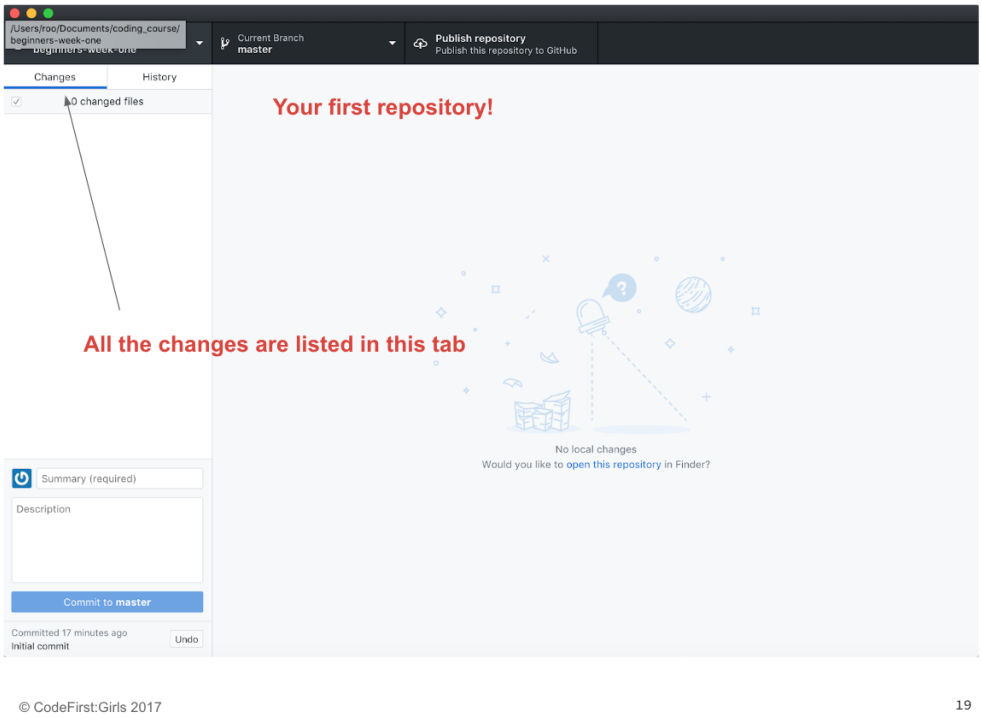
TASK: Creating a repository



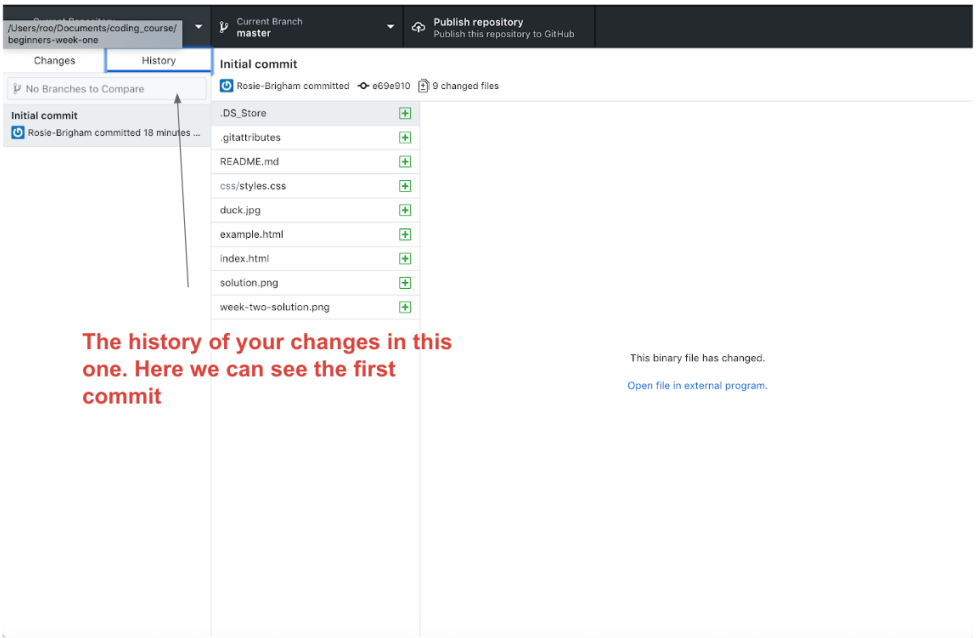
1. Open up the GitHub Desktop client on your computer
2. You should see something like this
3. Click on the **plus sign** in the top left hand corner of the screen



1. On the **Create** tab of the pop-up window type the name of your repository. In this case, because we are initialising a folder that is already there, we need to write in the folder name **exactly** as it is.
2. Check in the **Local Path** box that it has the correct path name for your week one project.
3. Click the **Create Repository** button.
4. The GitHub Desktop client will now create your folder at the location you wanted and, turn it into a git repository.



You now have a repository! You can look at all the changes you have made in the 'changes' tab



And you can see the history of your changes in the 'history' tab.

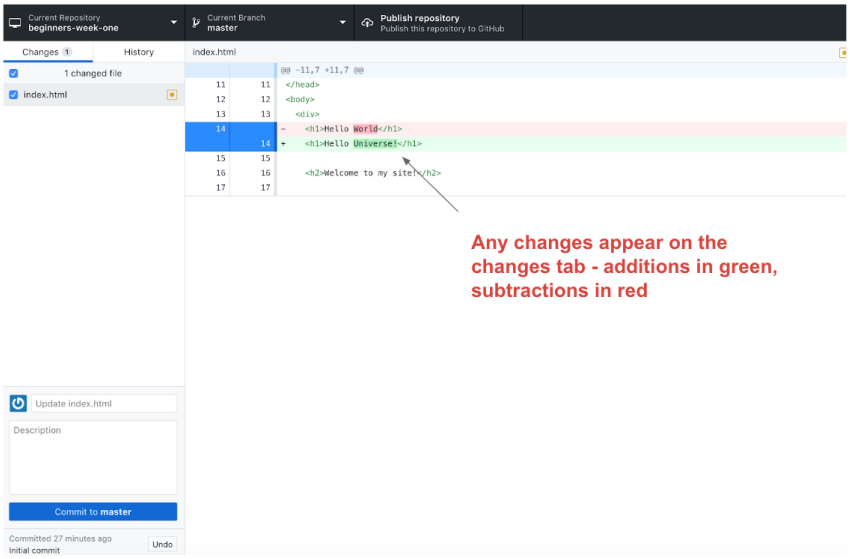
First Commit

Make a small change in your index.html file

index.html

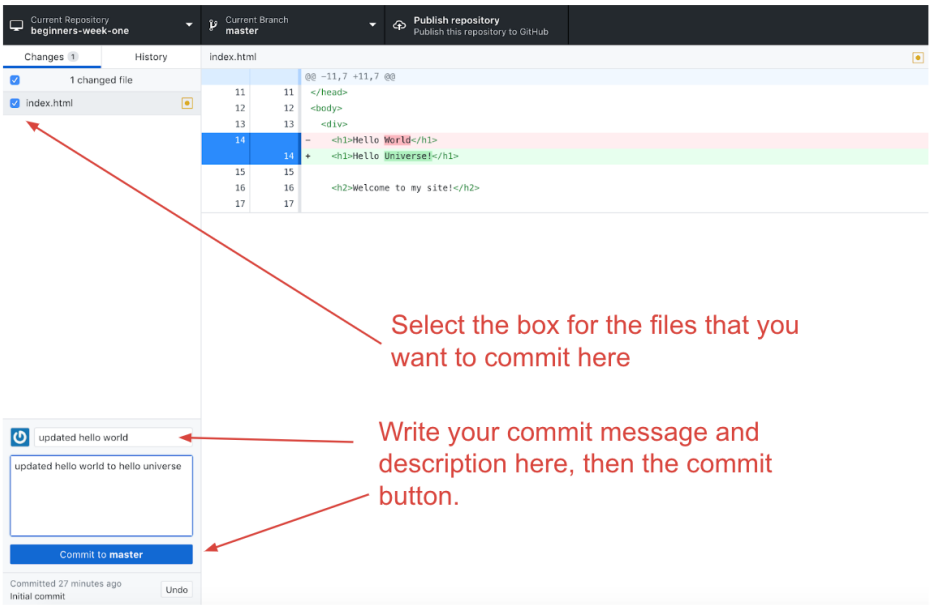
```
<!DOCTYPE html>
<html>
  <head>
    <title>First site</title>
     <link rel="stylesheet" type="text/css" href="css/styles.css">
  </head>
  <body>
     <h1>Hello Universe!</h1>
...
```

To make our first commit, we need to first make a small change to our project. Lets go over to the index.html file in atom and change ‘Hello World’ to ‘Hello Universe’.



24

Here we can see this change in the changes tab in desktop. Any additions to the file will be highlighted in green, and subtractions will be highlighted in red.

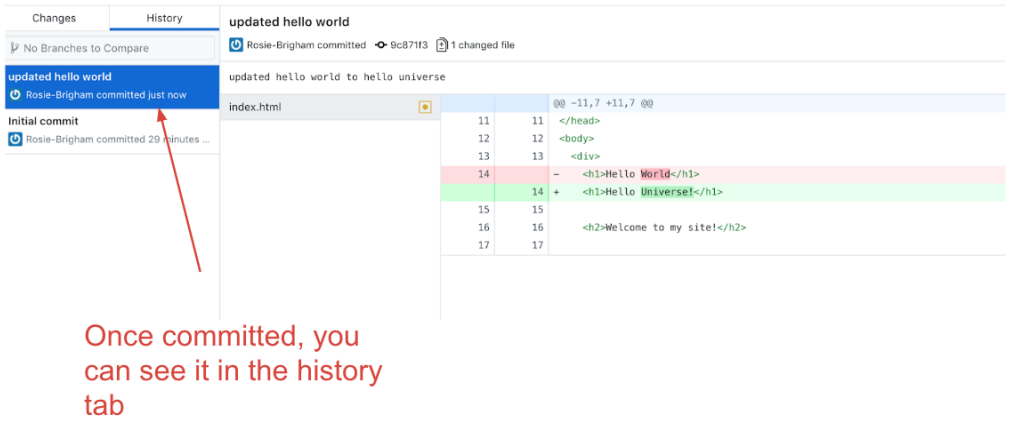


25

1. Tick the check box for the files that you want to commit (here we only have one line change in one file) but you can have multiple

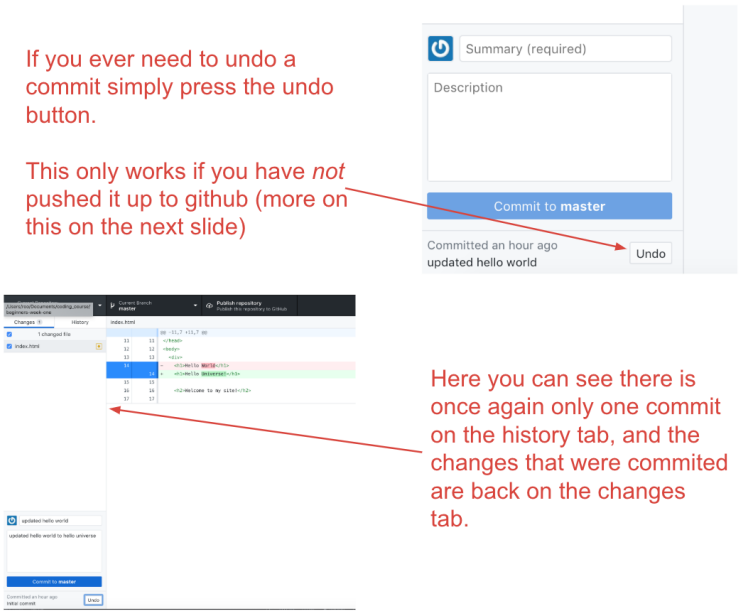
2. Then write a commit message and a description for the commit (if you want) and press the commit button

Make you commit messages descriptive, as when you look at the history of commits it will help you understand what you did. (‘added a thing’ is never helpful a month on when trying to find out where a bug may have been introduced!) Some developers also recommend to write them in the present tense. This makes it easier to read when you go through the history of a project later.



You will then be able to see the commit in the ‘history’ tab!

Undoing a commit



One of the benefits of using a version control system is that you can go back to a previous version of your project if you need to. This can be because of a bug that needs fixing, or maybe you made a mess of a file when trying something clever, or maybe the client you are working for decides that all the work you did in the past month is not what he wanted after all.

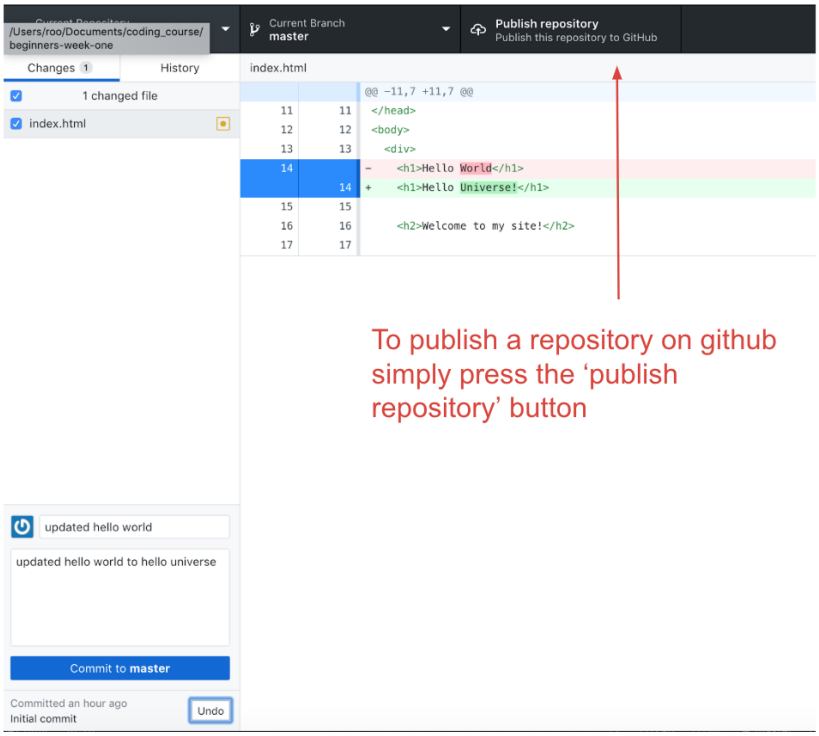
Whatever the reason, you'll be glad to be using version control when you are in a situation like that. There are two ways of reverting to an older version. Let's start with the easy way.

If you have just done a commit and realised you didn't want to do it, or maybe you didn't include all the files you needed to (or included the wrong ones) then undoing it is simple. Simply press the 'undo' button! When you click the Undo button it will put your repository back in the state it was just before you did the commit. Meaning that your changes will still be there, but they are listed as being uncommitted changes. Make whatever you need to do and then commit as usual.

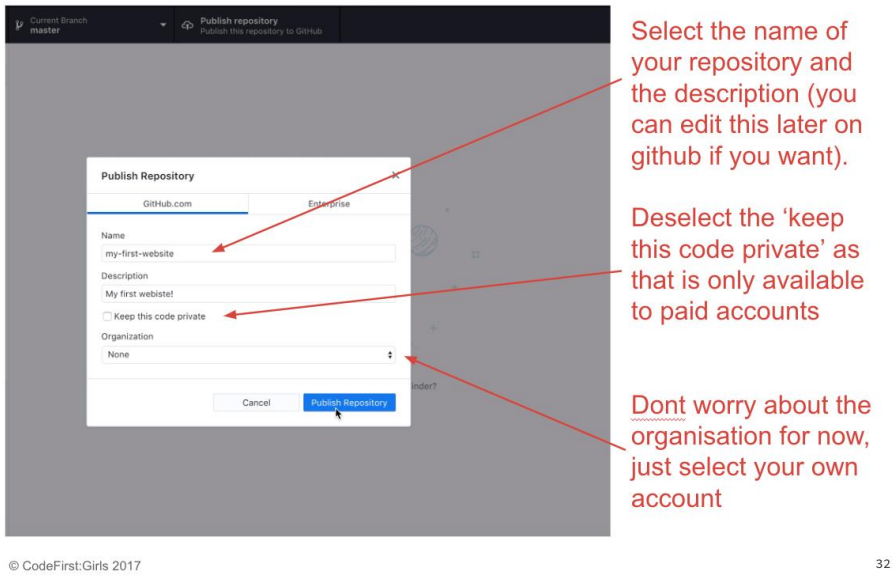
However, if you have already pushed the changes up to github 'undo' will get you in a muddle and you will need to 'revert' it instead. More on that later...

Publishing to github

What we have done so far is initialising a current project with git on your local machine. We need to publish it to our GitHub account. That's what we'll do next.

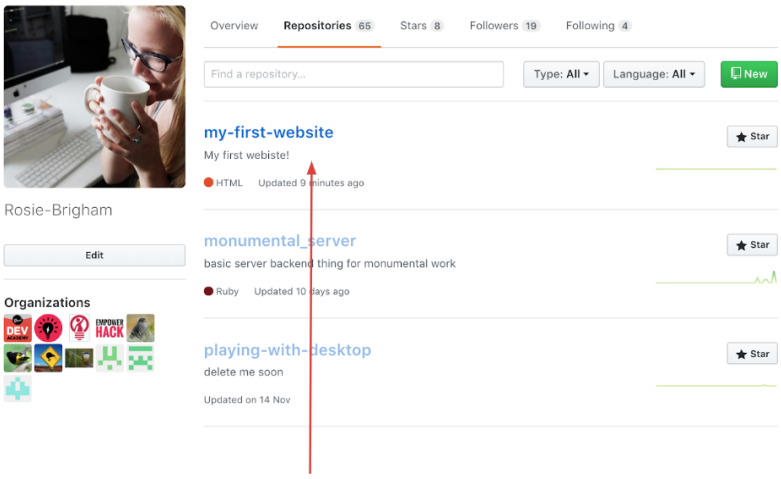


Simply click on the **Publish** button in the top right corner of the screen.



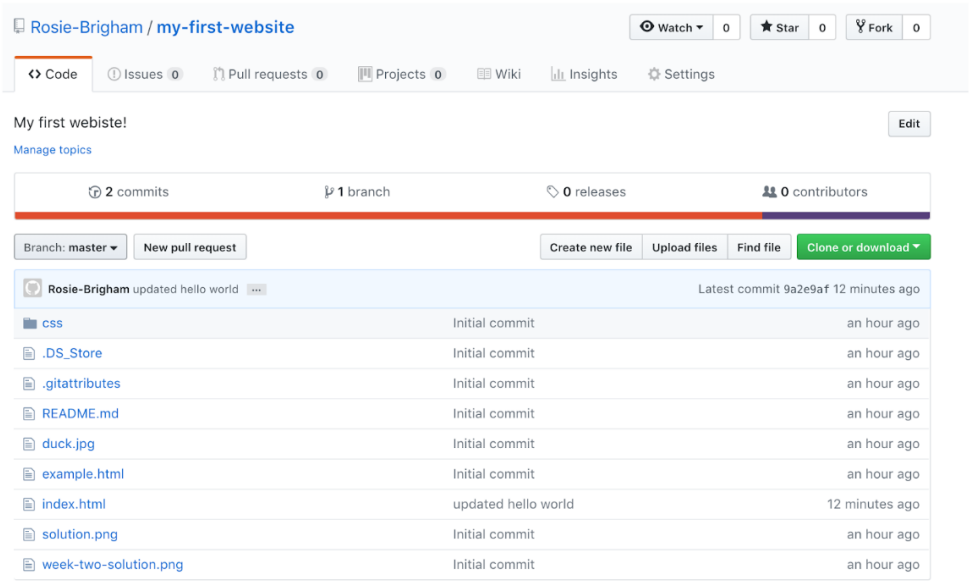
In the pop-up window you should see the name of your repository and your Github account name. If you want to you can write a description for your repository.

Click the **Publish Repository** button. You should now get a message saying it is publishing to GitHub.



Then head over to you github profile to see your new project!

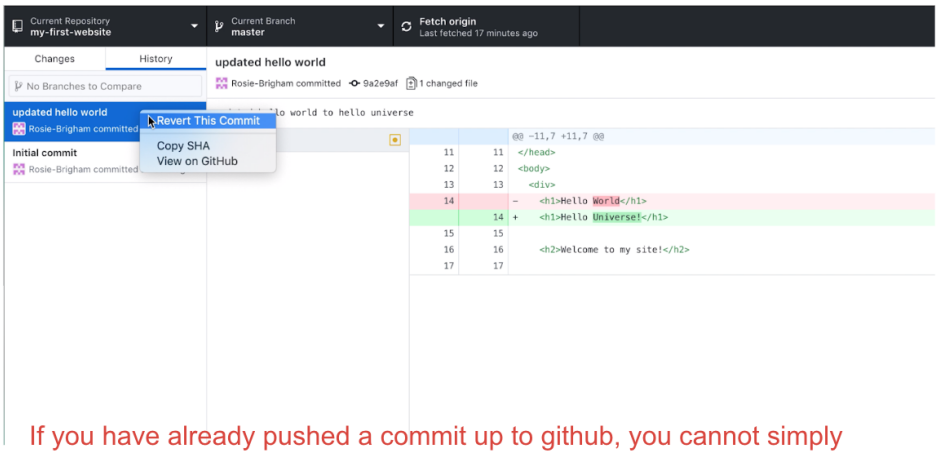
1. Go to your account on https://github.com
2. Click on the arrow next to your profile image in the top right corner and select Your profile from the drop down menu.
3. Then click on the Repositories tab and you should see your new repository at the top of the page.



All the files in your repository are listed here, together with the last commit message for each file

If you click on the repository name you will see the files in the repository. For each file the message of the last commit for the file will be shown too.

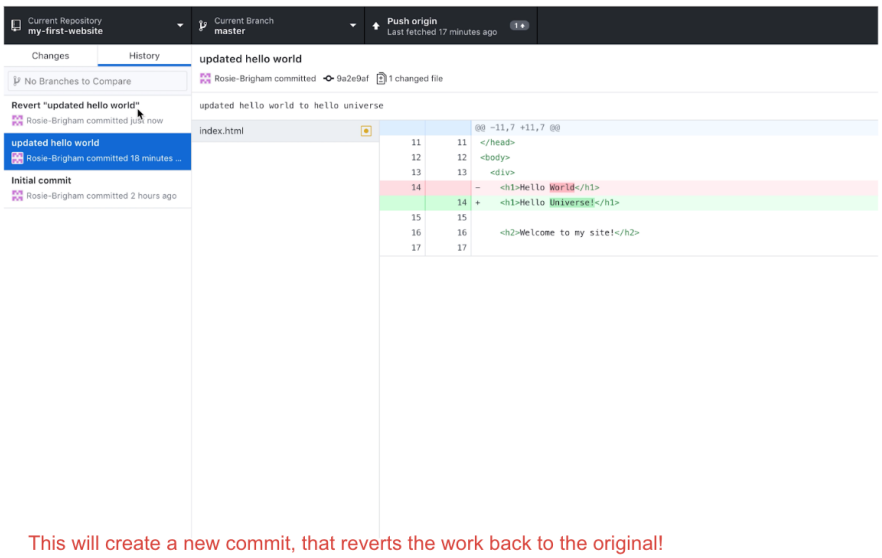
Reverting a commit



If you have already pushed a commit up to github, you cannot simply ‘undo’ it, instead you need to revert it. Simply right click on the commit and select revert

Use this, instead of the ‘undo’ function to revert a commit in the long ago past, to to undo something that you have already published and pushed up to github

When you have pushed up a commit you cannot simply ‘undo’ it, as it will mess up the history between your local and remote versions. Therefore, you will have to ‘revert’ it. This will create a new commit which reverses the work you did in the commit you want to undo. To do this, simply right click on the commit and select ‘revert’



This will create a new commit, that reverts the work back to the original!

As you can see, this creates a new ‘revert commit’, make sure you publish this by pressing the ‘push origin’ button.

Creating a branch

Up until now we have been using a version control system to commit our changes and to fix commits we should not have committed. But as we saw earlier, using a version control system is also a great way of adding new features in isolation to a project without jeopardising the current code.

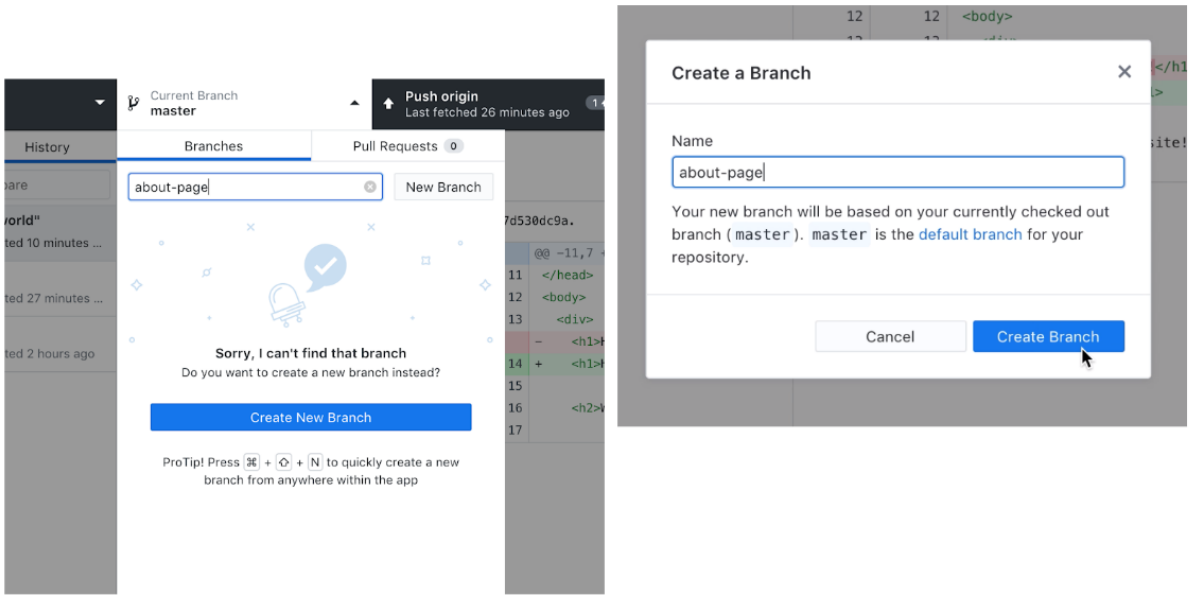
To do this you need to create a separate branch on your project.

Remember when you create a git repository it will also create your first branch with the name master. If you want to add a new feature to your project then you first create a different branch, do all your work, test it out thoroughly and then merge the feature branch back into the master branch.

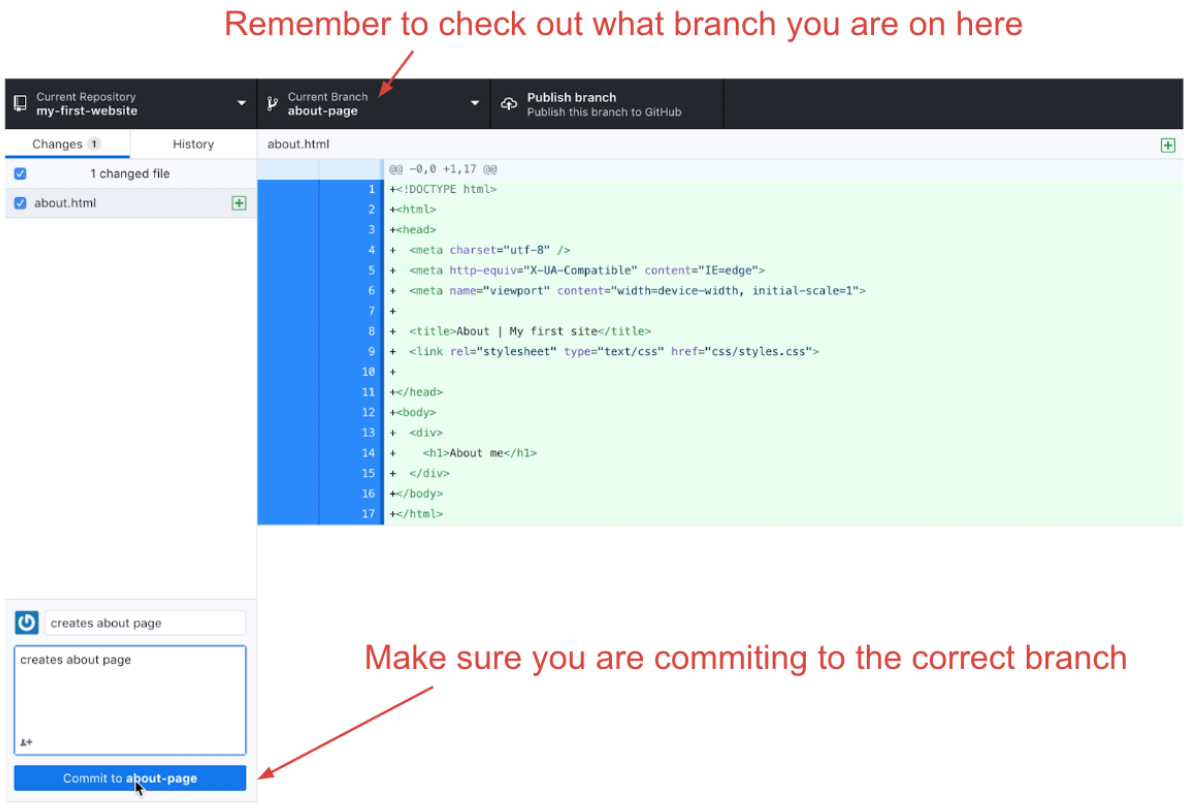
When you create a new branch, git will take a copy of the current branch you are in, usually the master branch. This new branch will sit alongside the main branch until you are ready to merge it back into the mainstream.

Let's create a branch on our repository.

Click on the branch tab to move onto a branch, or create (and then move onto) a new one

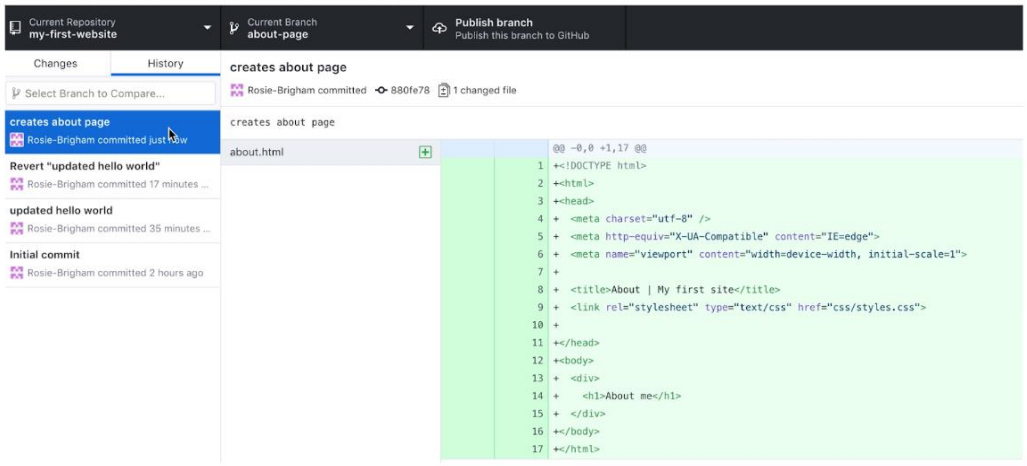


1. First remember to pull or push any changes up before you add a branch
2. Click on the 'branch' tab, you can type in a branch name to search for it. If there is no branch found, as in this case, you can simply create a new one.



Here we have added a new page, about.html.
You can check what branch you are in on the branch tab, and when commiting new changes you will see that the button tells you what branch you are committing to.

Let's commit this new page onto the about page branch.



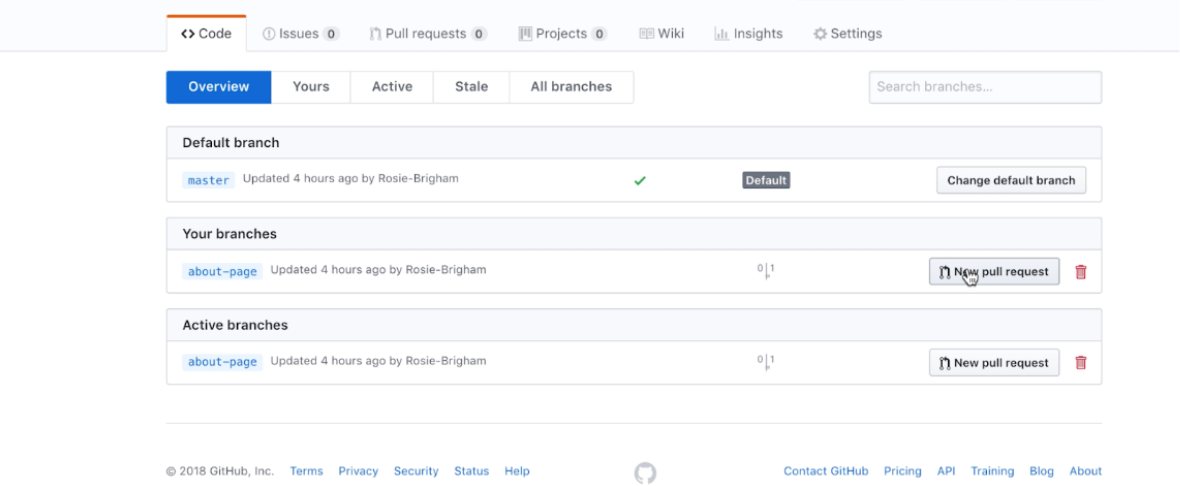
The history on this branch is now different to the master branch. Press the **publish** button to get the new branch, and your changes, up on github.

Here we can see the commit made onto the about page branch. The history on the about-page branch is now different to that on master. Let's push the **publish** button to push our changes, and the new branch, onto github.

But how do we get these changes back onto the master branch? We do this using **pull requests.**

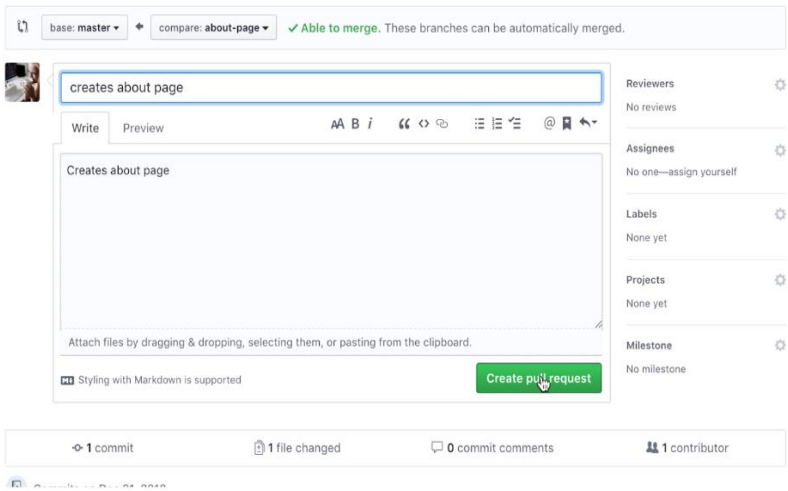
Pull requests

When you have pushed your new branch with your changes up to github, you can merge them into master using a pull request. On the ‘branches’ tab in your repo, select ‘new pull request’ on the branch you want to merge changes in from



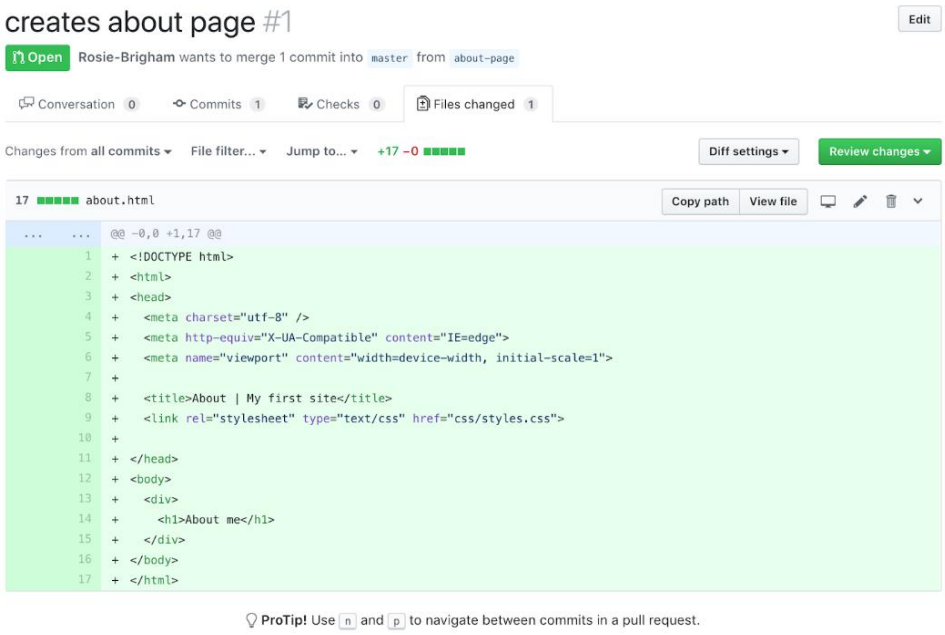
Go to your repo on github and navigate to the ‘branches’ tab.
Here you will be able to see a list of all the branches that you have pushed to github. If any of the branches histories are different to that of master, there will be a ‘new pull request’ button. Click on this for our about page to create a new pull request.

Add a title and description for your pull request to describe the changes you've made



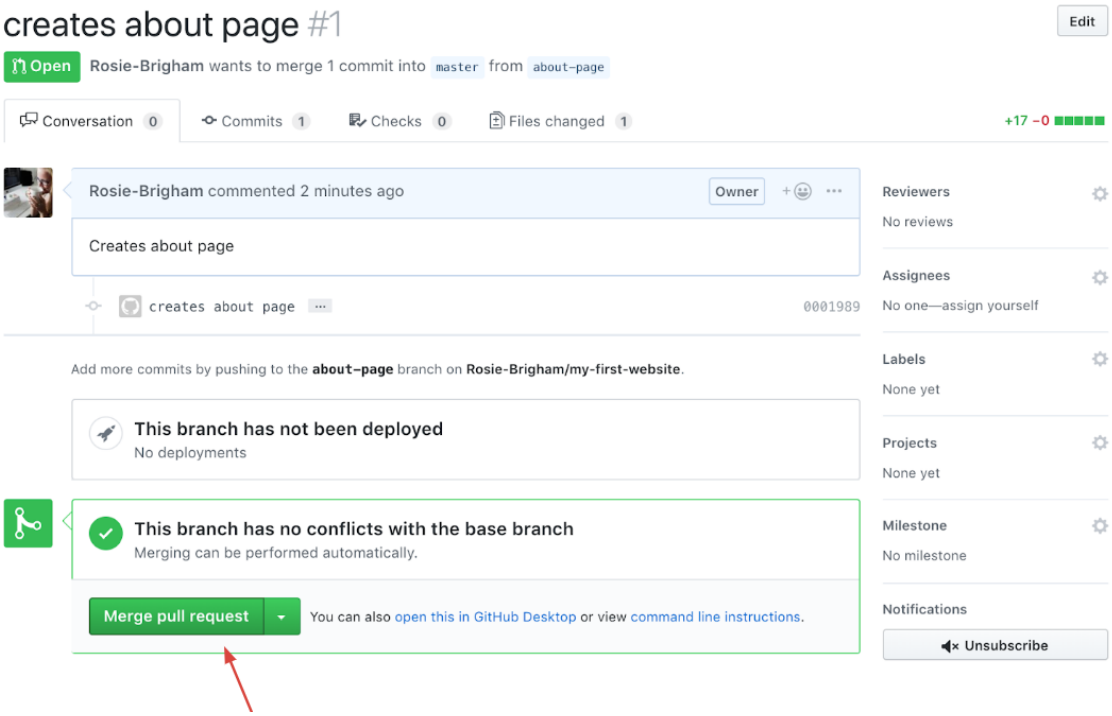
Here you will be able to review all the changes you have made, alongside adding a title and description to your pull request. For large complicated ones, these are important so that someone who is reviewing your code will know exactly what changes you are making and why.

In the pull request you can then review the changes you have made, and add comments to things that maybe need to change



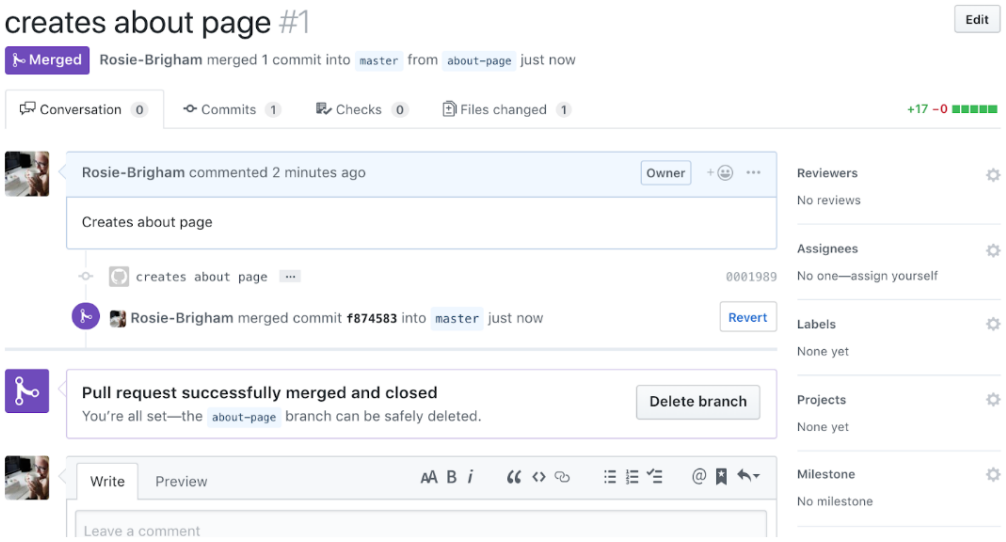
Once created, you, or a colleague, can review the changes you have made. At this point, if there are any things you think that needs to be changed before it's ready to be merged in, you can comment and request for changes.

If everyone looks good in the pull request, simply press the merge button and then confirm!



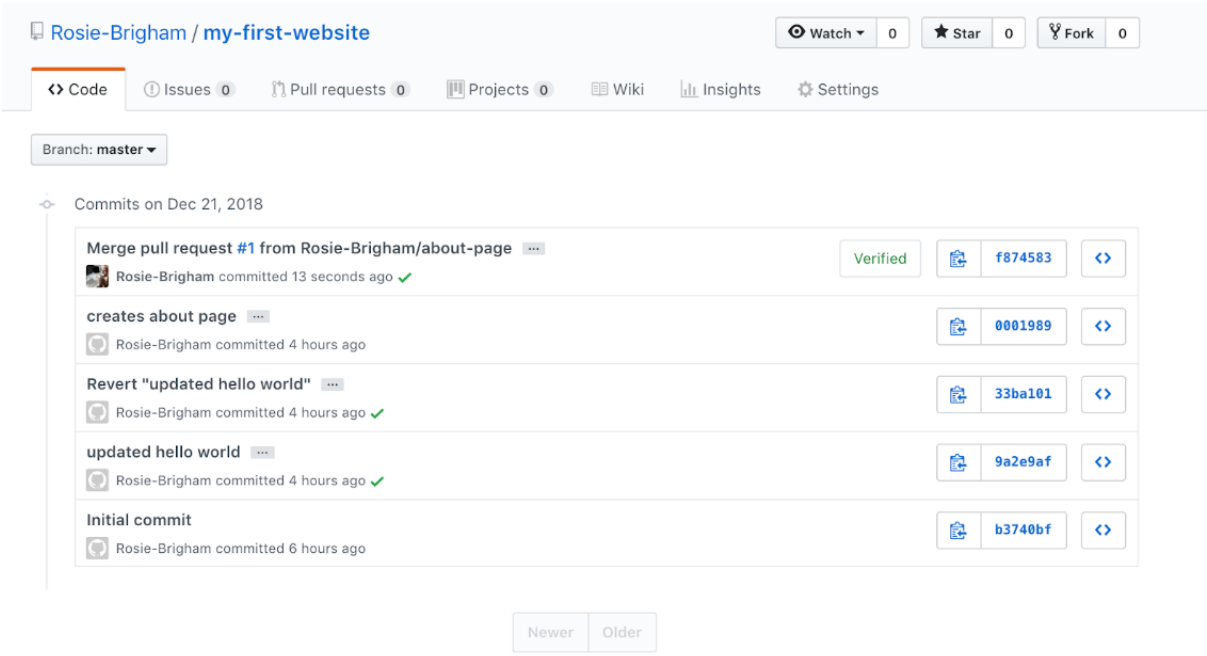
If everything looks rosy, simply press merge to merge your changes in!

All merged! Make sure you delete the branch after you have merged the pull request to keep your repo tidy



And vioala! All merged in. It's best practice to delete your branch to keep your repo nice and tidy.

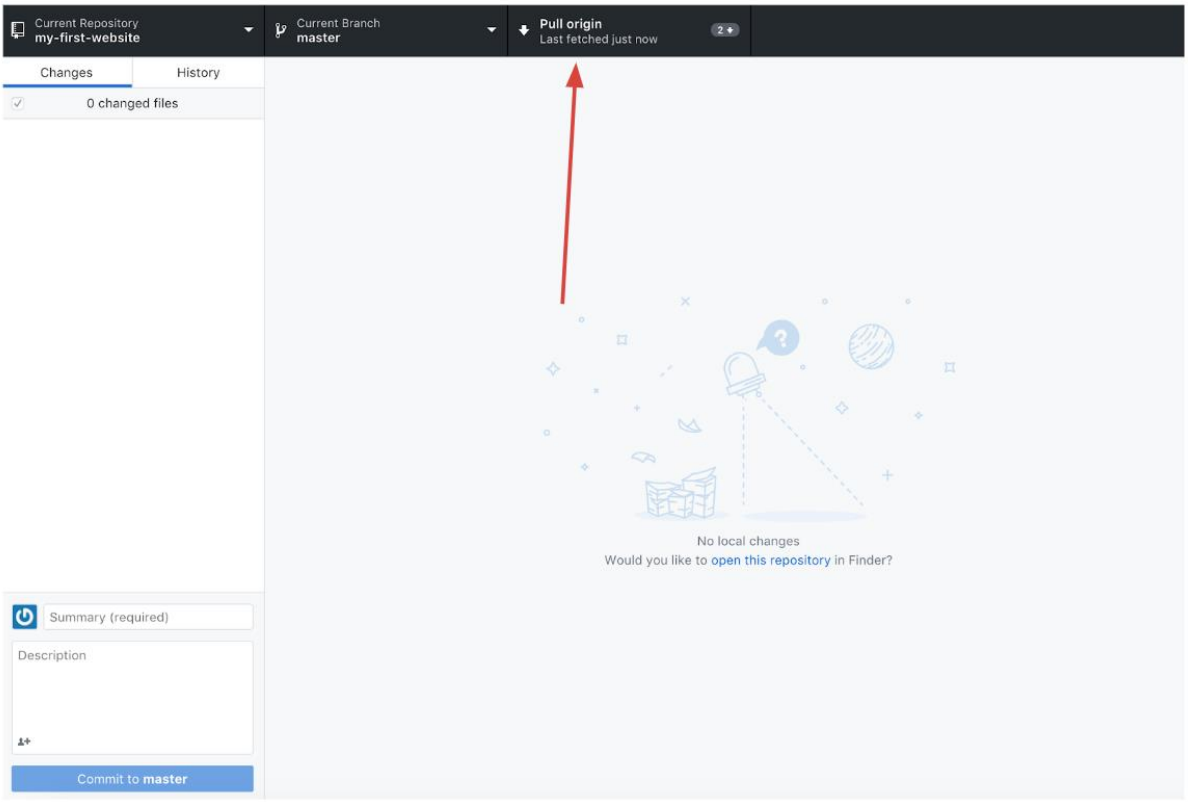
You can see the new commits from the merged branch, and the commit created by the merge, on the history of the master branch



Here we can see the history of the master branch, including the commit made on the about-page branch that has now been merged in, alongside a merge commit.

A little bit of good advice

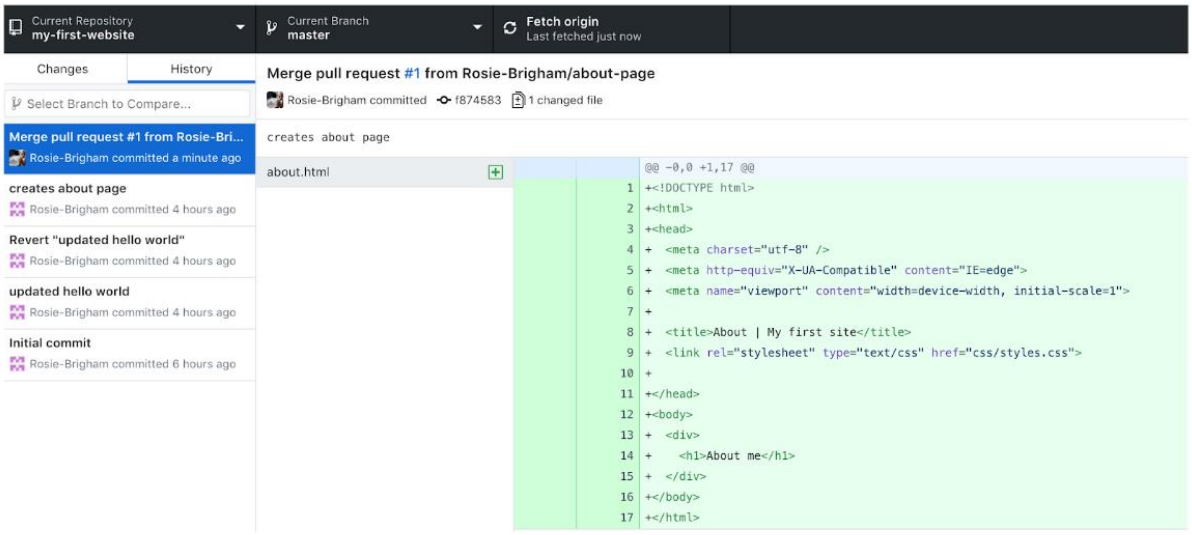
Whenever you have things on your remote repo that is not on your local one (such as merged branches) it is crucial that you pull them down. You can do this by pressing the 'pull origin' button.



It is really easy to make a right mess of things when working with repositories. In a moment we'll do this deliberately so that you can learn how to resolve conflicts. But the best way of working with git is to avoid conflicts all together.

So the first rule of git is as follows:

Always pull down new changes



Now you have the commit created on github by merging the about branch into master on your local machine

Now we can see the merge commit, created on your remote repository, pulled down onto your local machine.

Publishing to github pages

Publishing on GitHub Pages

- Free web hosting for git repositories on GitHub
- Make sure `index.html` is all lowercase
- Your website url will be `[your-github-username].github.io/[repository-name]`

Publishing on GitHub Pages

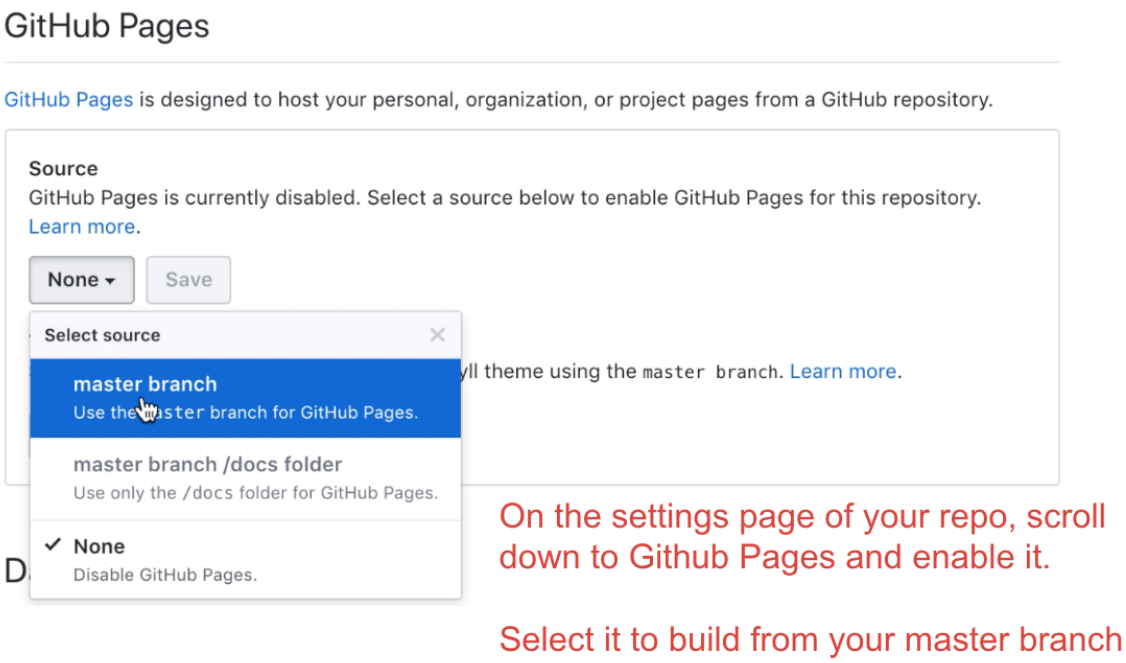
Git and GitHub isn't only about version control. Another great feature that GitHub has is free hosting for your website projects. The hosting can only be used for repositories stored on GitHub accounts, and is only suitable for static websites where there is no requirement for a database. Despite that, it's a good way of having a website online at no extra cost.

How Github Pages works

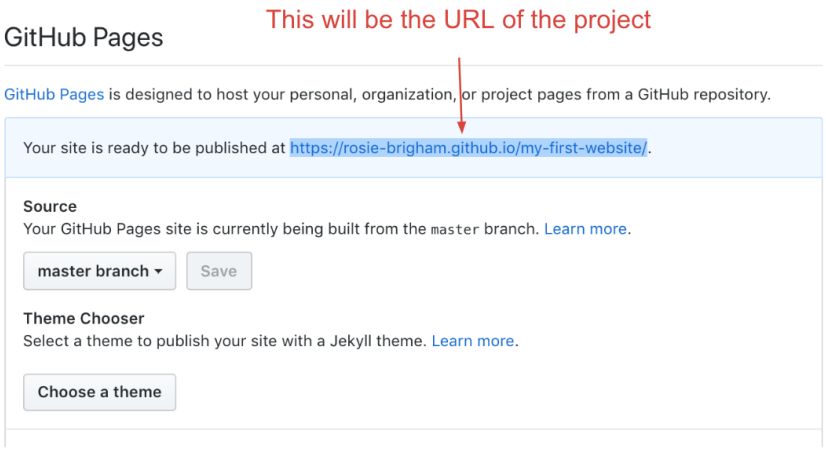
In order to publish your website through GitHub pages you will need to create a branch in your repository called gh-pages. GitHub will look for this branch and use the contents of this repository to create a website from .

Any code you want to show on your website will need to be added to this branch. If it isn't on this branch you will not see it on your online website. In other words, if you for example added a page to your website on the master branch, and you haven't merged your master branch into your gh-pages branch, then you online website will not have this extra page.

Once you have published your website to GitHub Pages your site will show up at **[your-github-username].github.io/[repository-name]**



On the settings page of your repo on github, select the ‘settings’ tab and scroll down to the ‘github pages’ section and select ’enable’. Select the master branch to be the branch that it is built from.



After you have done that, and if there are no bugs in your code, it will give you the url that your site will be published on. **Note:** it will not automatically deploy the site, but will do after you push up your next commit. So you will need to create a new commit in order to see your site on the given URL.

Part 2 - Collaborating

Note - by this point, every student should have published their project on github, if they have not, allow some time for them to go through the slides at their own pace and do that before we move onto collaborating

Collaborating

So far you have initialised your first site with git, made couple of commits on two different branches, pushed those commits to github and even published your site on github pages!

Now lets step things up a notch, and use github to collaborate with the team projects you started last week

You can use a version control system on your own to track the changes you make to your projects. And in that case you will be the only person contributing to the repository. But version control is at its most powerful when a project is being worked on by more than one person. And that's what we are going to do now. We are going to add a collaborator to your repository.

Find the people in your team in order to create your group project repo on github. Only one of you needs to do this and this person will become the owner of the repository. The other person will be the collaborator.

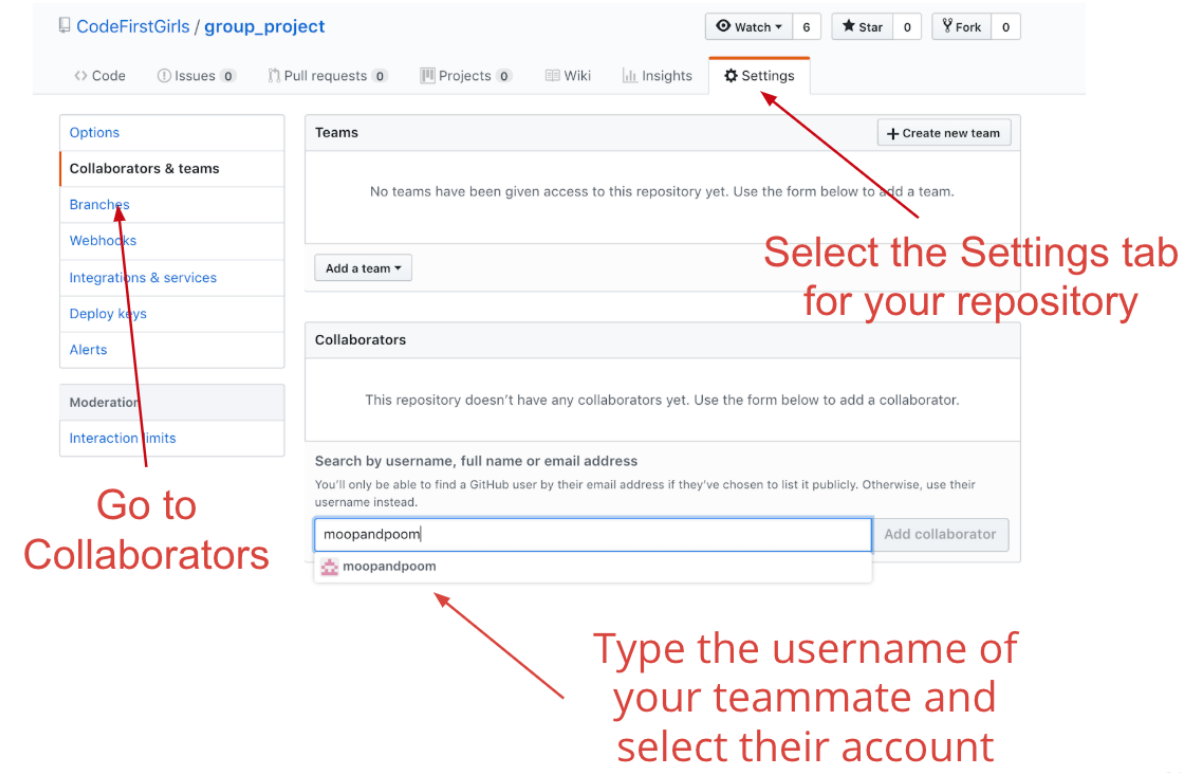
Task:

1. Find your teammates and sit with them for the rest of the session. Choose one person from your team to lead the first section.
2. If you have already started coding your project, initialise this in a new repository on github desktop.
3. If you haven't started, create a new folder in your coding_course folder called 'group_project' and create a file called 'index.html'. Initialise this as a new repository on github desktop following the instructions earlier
4. Commit your work and publish it to github

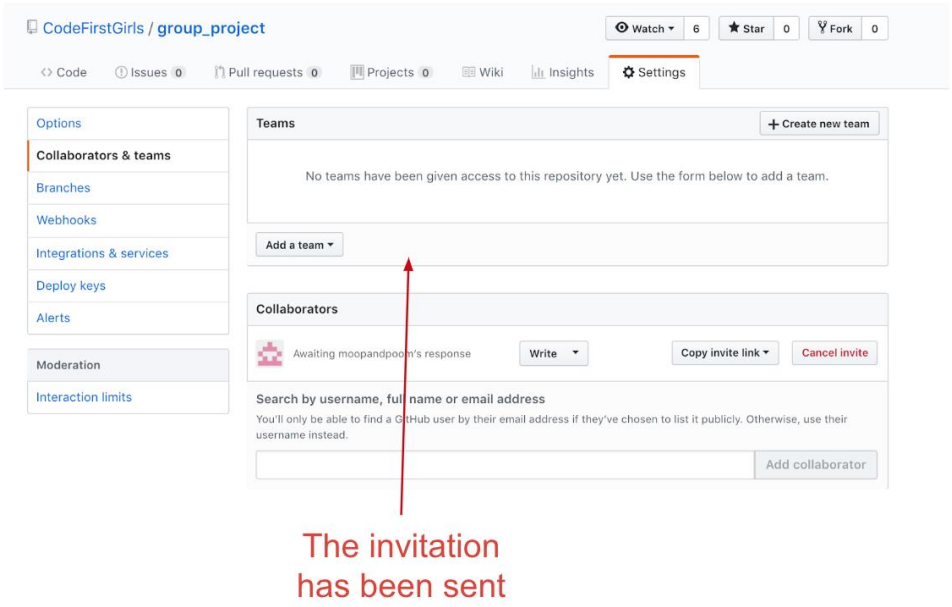
Adding collaborators

- You should now have your group project pushed up to github (don’t worry if it’s only an empty file for now, you can add to it later on)
- We are now going to add your other team-mates as collaborators

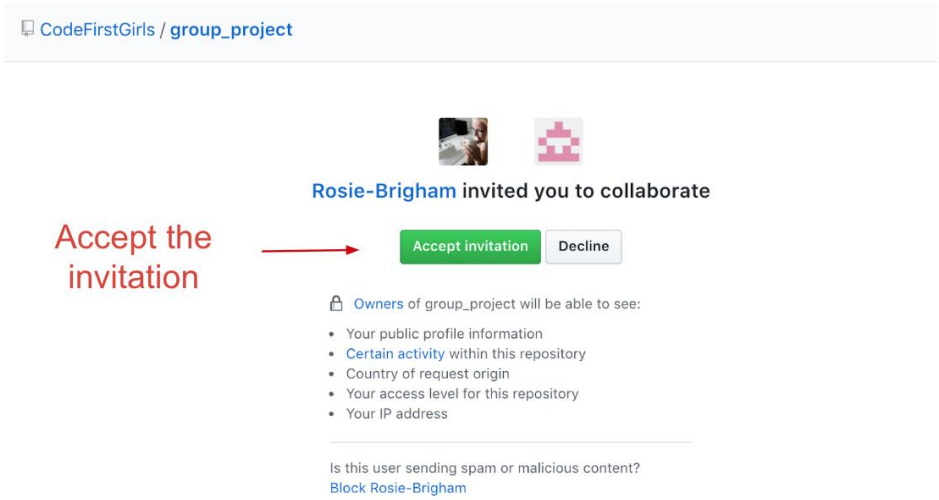
Everyone should now have a group project on someone's laptop, and pushed up to github.
Now we are going to add everyone else in the group as collaborators so that you can all add to the project



1. In the 'settings' tab of the new project on github, select the 'collaborators and teams' button and add each team mate.
2. GitHub might ask you to confirm your password at this point.
3. You will need to search for them by email or github username. Once added, this will send them an invitation to their email account that they will then need to accept.



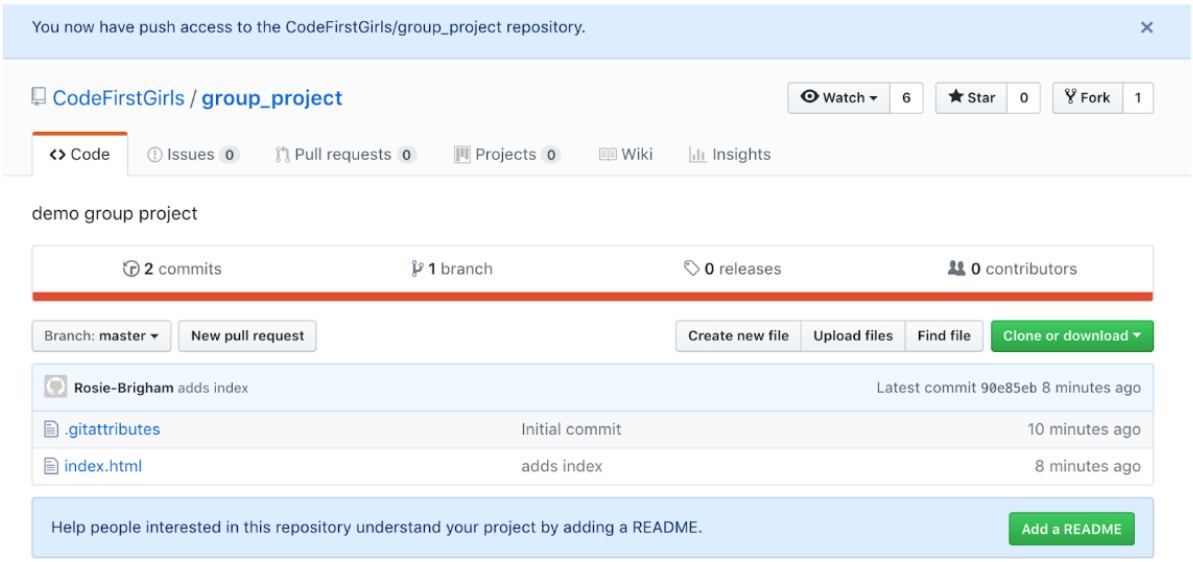
1. Your Collaborators page will now show you your teammates account and telling you that it is awaiting a response.



2. Meanwhile GitHub will have sent an email to your teammate asking them to view the invitation.
3. If you are the collaborator teammate, click the **View invitation** button. This will open up your GitHub account and you will see a page asking you to accept the invitation. Click the **Accept invitation** button.
4. You will now be taken to the repository page. Beware that even though you are now a collaborator on this project, the repository does not appear on your

GitHub account online. Repositories remain attached to the account of the creator.

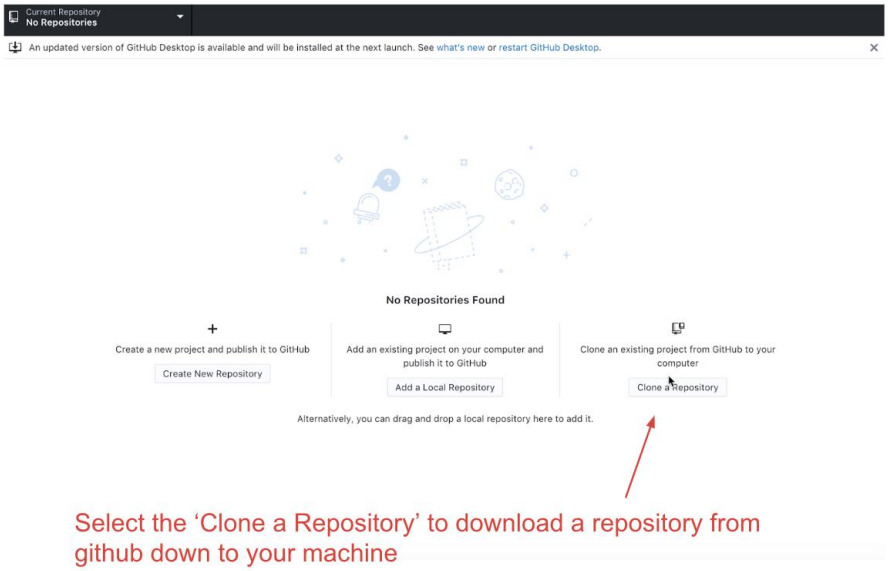
You now have push access!



Once you have accepted, you will now have push access to the repo! You can now clone it down onto your desktop.

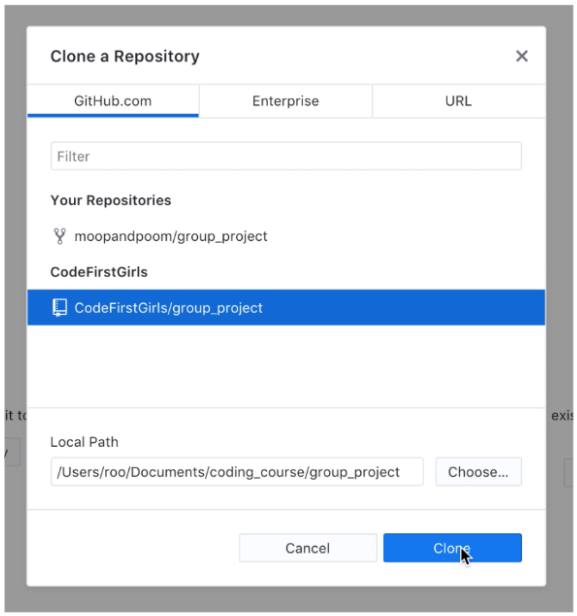
TASK: make sure everyone in the group is added as a collaborator

Cloning a repository



Now that everyone is a collaborator, you will now all have to clone the project onto your machine. This is slightly different to how we did it the first time as now we are simply downloading a current project onto your machine, instead of initializing a project that is already on your machine with git.

Select the 'clone a repository' option and select the repository you want to download



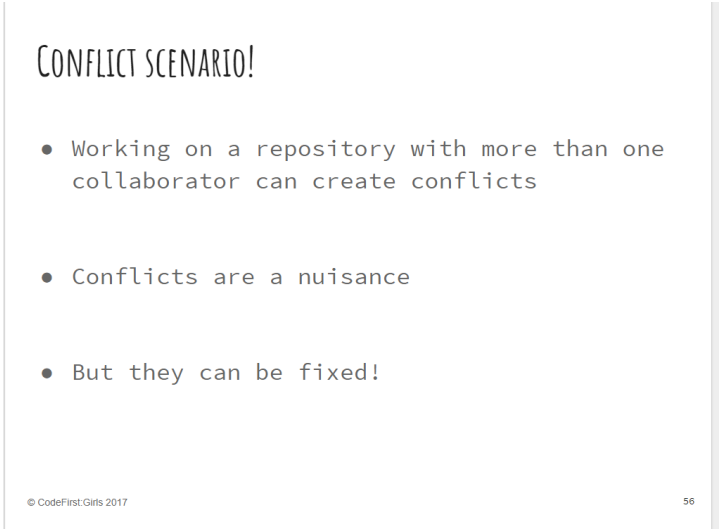
Select the repository you want to download

Make sure you put it in your coding_course folder. Once the cloning is complete you will now see your newly cloned repository in your GitHub Desktop client. As a collaborator you can now make some changes in this repository.

TASKS

Take it in turns, working together, to do the following:

1. **Check out a new branch**
2. **Make a small change, commit it and push it up to github**
3. **Open a pull request on github**
4. **Get someone to merge in the pull request**
5. **Pull down the changes onto everyone's version**



Conflict scenario!

When you are working on repositories with more than one collaborator it is likely that at some point a conflict will arise. And I don't mean conflicts in terms of a disagreement between collaborators on an aspect of the project. But conflicts where one collaborator is working on an older version of the repository that has already been updated.

Remember the first rule of git? Always pull down changes first will avoid most conflicts, but sometimes they still slip through the net. The most likely time when conflicts will happen is when collaborators are working on the same repository at exactly the same time.

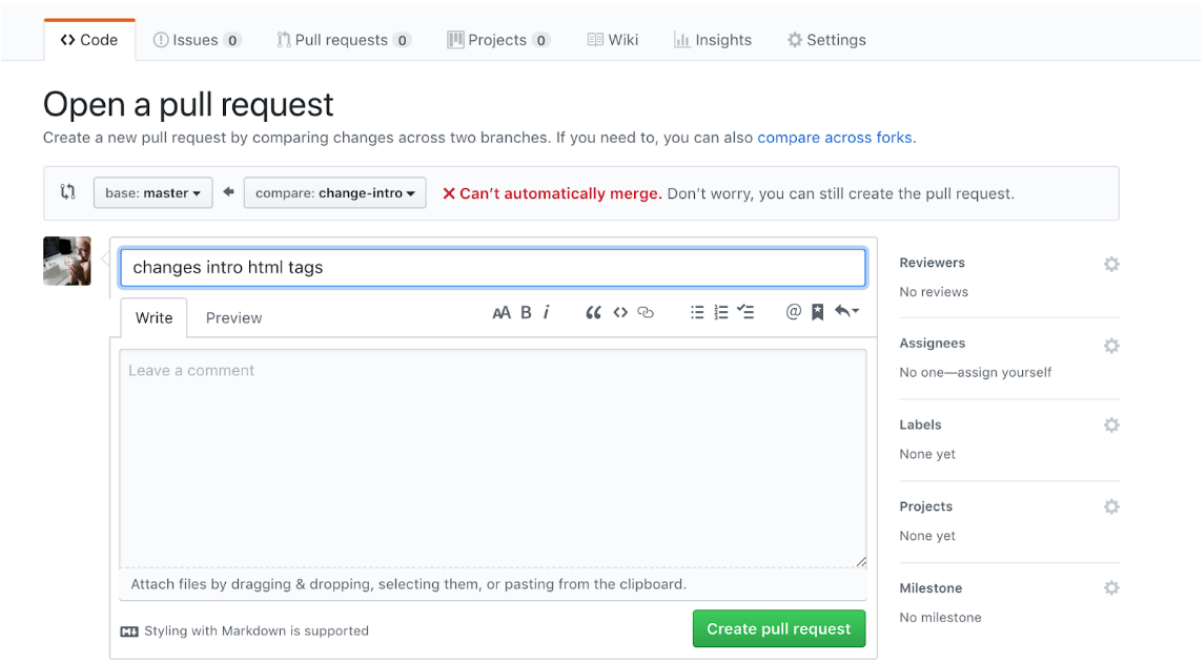
Let's set up a conflict scenario so that we can learn how to resolve it.

Conflict scenario!

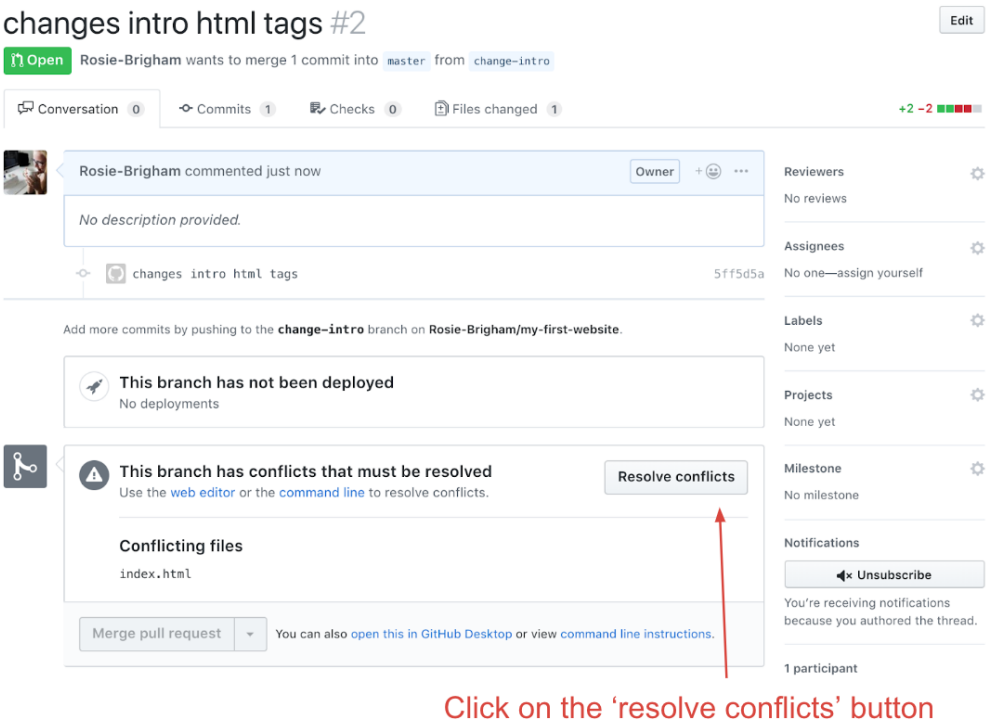
- In the following example, line 14 of index.html was changed in the **change-intro**, and in master *after* that branch was created. Github can't work out which one should have precedence so you have to tell it manually

In this scenario, a branch was created called 'change-intro' in which we have changed our intro paragraph. However, after the branch was created, that same line was changed on the master branch. As both branches have been changed on the same line, github cannot work out which one should have precedence, so it requires you to tell it manually. We can do this by opening a pull request.

You can still open a pull request with a merge conflict, but you have to manually fix the conflicting changes



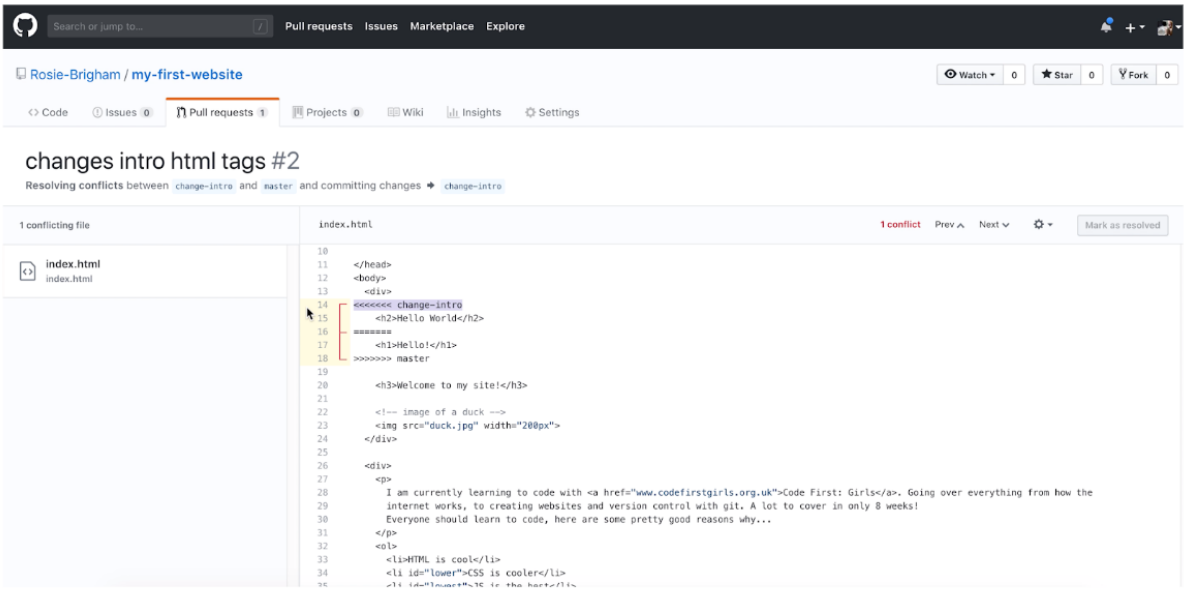
Here is the pull request to merge the 'change-intro' branch into the master. As you can see, we can still open the pull request even though it cannot be automatically merged.



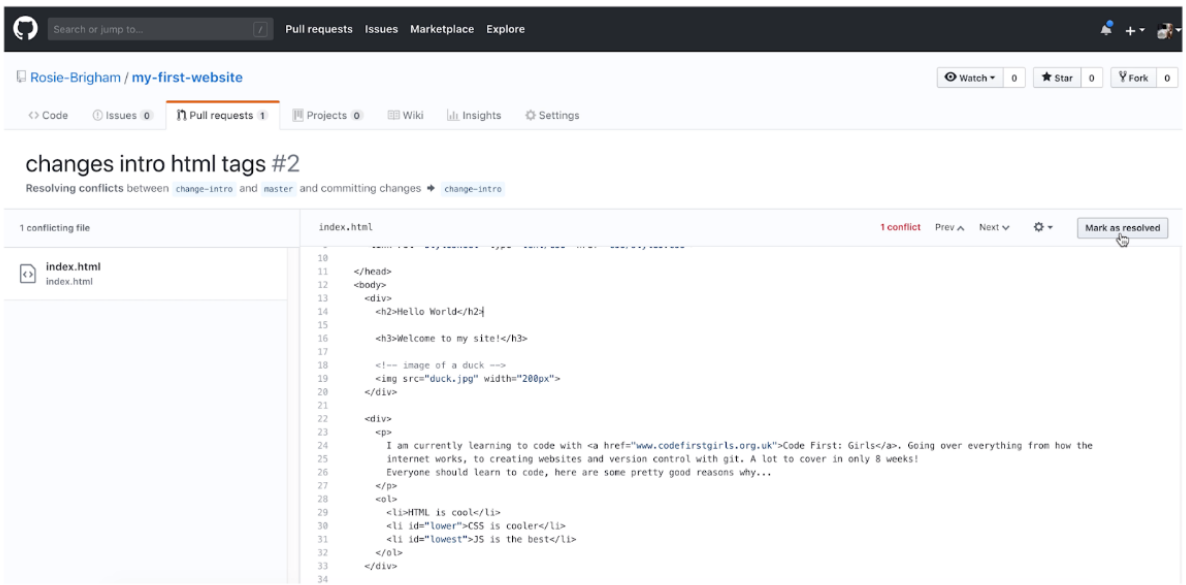
Click on the 'resolve conflicts' button

Handily, github gives us the option to 'resolve conflicts' manually and opens a web text editor. This is what we will do here.

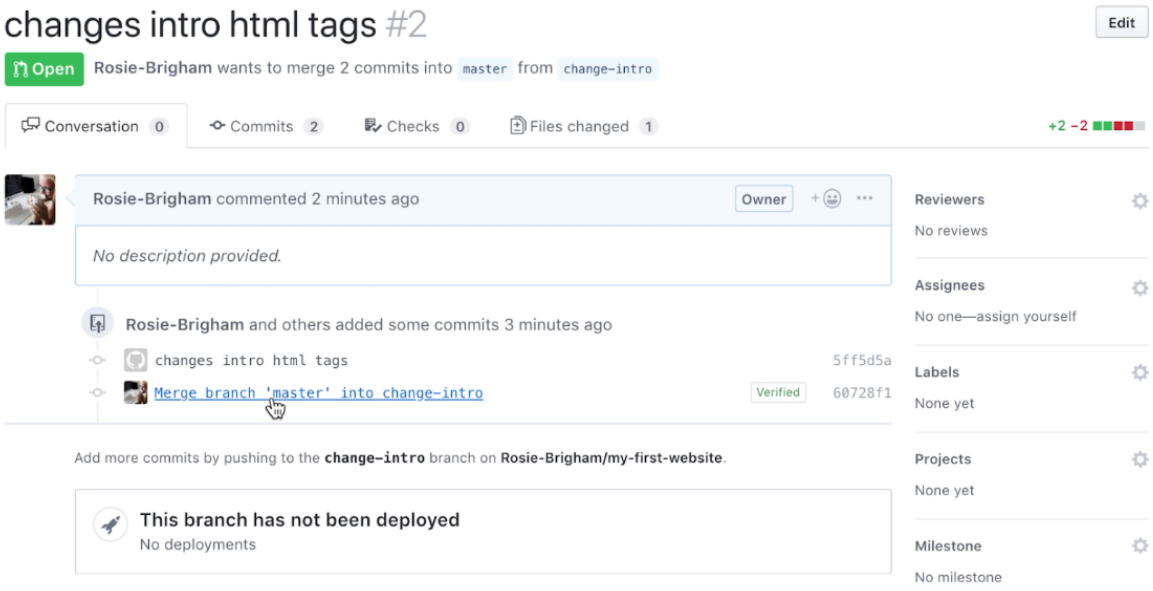
This opens the web editor on github, simply delete the changes that you do not want and select ‘mark as resolved’



Simply delete the version of the line, or lines, that you do not want. Make sure you also delete the chevrons as well, otherwise they will appear in your code.

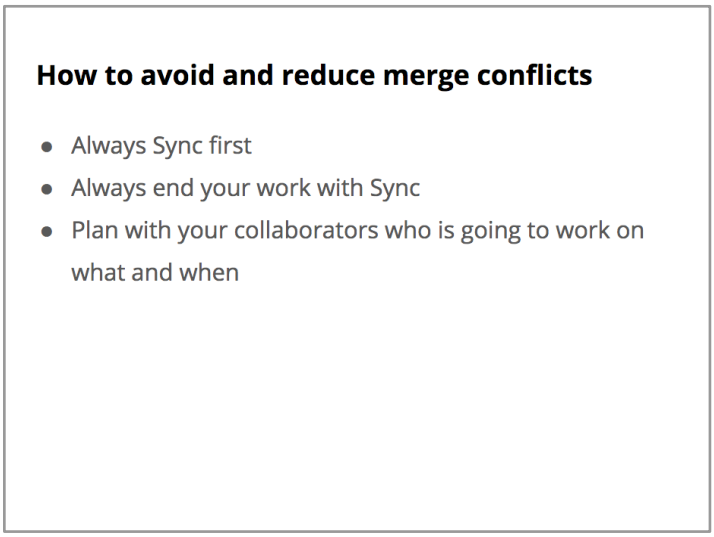


When you’re done, hit the ‘mark as resolved’ button.



This adds a new merge commit - and the conflict is resolved!

This creates a new merge commit, and the conflict has been resolved!
Make sure you then pull this down onto your remote machine.



How to avoid and reduce merge conflicts

Merge conflicts are not fun to deal with, but a few things can be done to avoid them.

1. Always sync first. Anytime you want to make a change to a shared repository make sure you sync first. This should also be the last thing you do so that you don't run into sync issues at a later date.
2. It's worth planning with your fellow collaborators which sections who is going to work on and when. This will avoid two people working on the same file at the same time.

Homework

1. Make sure both your group project and personal site is published on github, and your personal site is published on GH pages
2. Work on your group projects, using github to collaborate!

Homework for this week:

1. Make sure both your first website and group project is on github, and your first site is published on github pages

2. Work on your group projects, using git to collaborate!