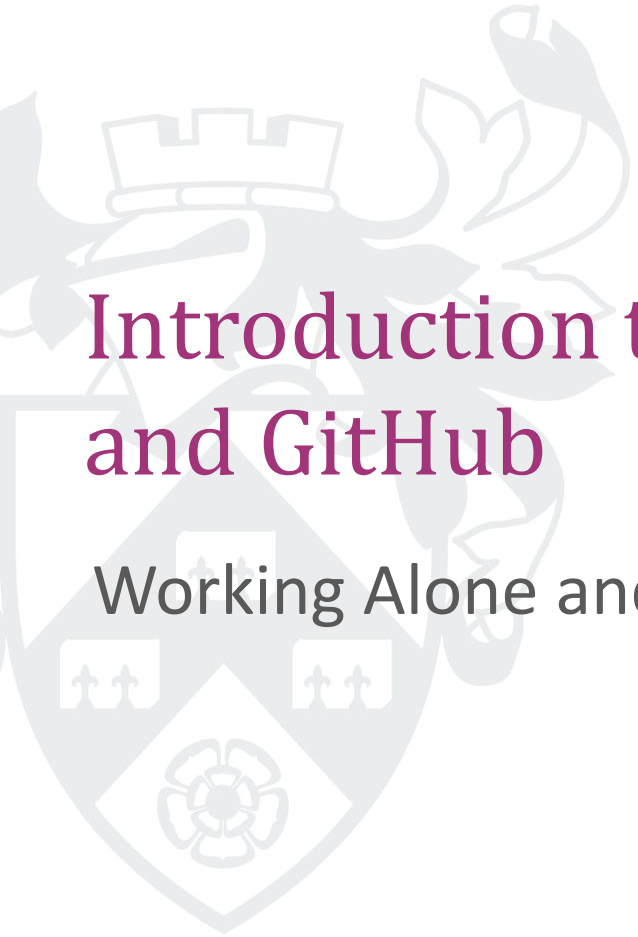


UNIVERSITY of York

Introduction to Version Control with Git and GitHub

Working Alone and Working Collaboratively

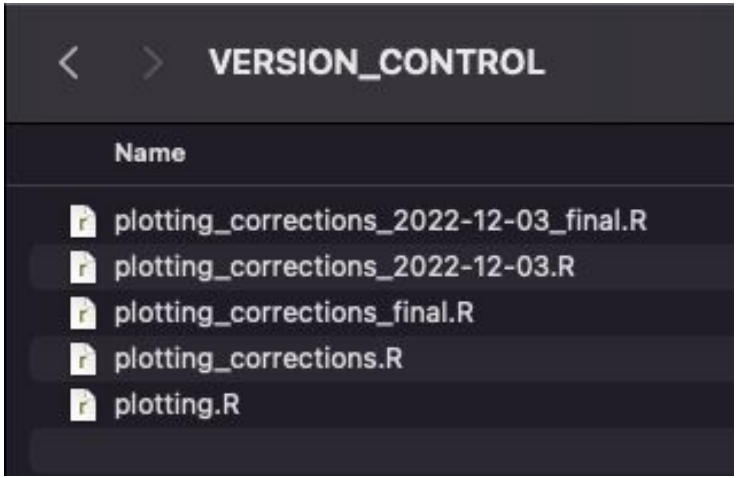
Goals for the session:

- Basic understanding of what version control, git, and GitHub are
- Knowledge of a few important git-related terms
- The ability to create a repository hosted on GitHub
- The ability to make commits in a repository
- The ability to clone a repository
- The ability to revert to a previous commit in a repository
- The ability to work on branches in a Git repository
- The ability to create and resolve a Pull Request on GitHub

Part 1: Introductory Information

What is version control?

- Any means by which you track changes to something:



We've all done it!

Is this effective version control?

What is Version Control?

- Structured way of keeping track of changes to files
- Implemented differently in different ‘version control systems’ (VCS)
- Typically share the concept of a history, the changes which make up the history, and the ability to move freely back and forth within the history
- The most popular VCS is *git*, which we will be focusing on today

Why Version Control?

- Keeps a labelled history of all changes made to a project, and lets you rewind them if something goes wrong
- Lets you experiment with changes to a project whilst keeping stable versions of the project untouched
- Provides lots of convenience features for working with other people on a project
- Can act as a backup of your projects, especially when using a hosted service like GitHub

What is git?

- A specific version control system, created by the guy who created the Linux kernel (the myth goes that he named it by his personal reputation)
- A ‘distributed’ version control system - many people can have copies of the stuff under version control and they don’t need to be in sync
- One of the most important programming tools you can benefit from becoming comfortable with

What is GitHub?

- Importantly, a website for hosting, organising, and collaborating on projects that are under version control
- Provides a consistent way to store and share version-controlled code (and other projects)
- Can act as a portfolio of sorts
- Has a whole host of other features (project management, continuous integration tools…)
- See the WACL GitHub organisation for an example

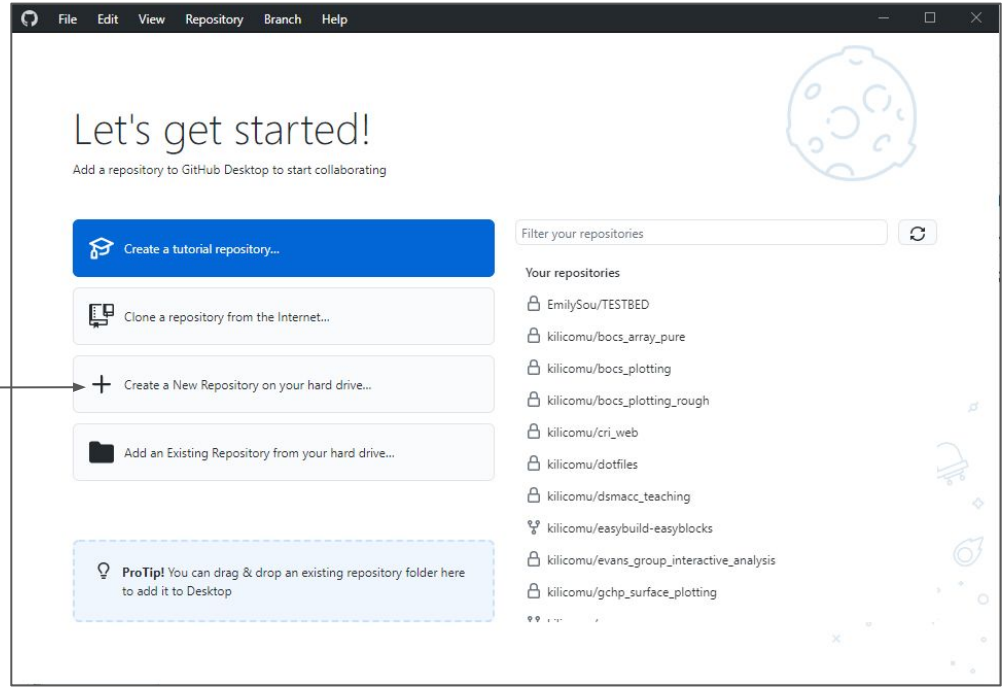
Git Glossary: Basics

- **Repository**: the collection of files that you want to version control
- **History**: the timeline of changes that have been made to your files
- **Clone**: the act of copying a repository
- **Commit**: a packaged and labelled change to some files in your repository
- **Push / pull**: the act of synchronising your history with another copy of your repository

Part 2: Working Alone

Creating Your First Repository - GUI

Click this!



Creating Your First Repository - CLI


```
$ git config --global init.defaultBranch main
```

(It is not important to understand this first command - you will only ever need to run it once, and if you've never used git before you will likely never notice what it has changed)

```
$ mkdir my_first_repository

$ cd my_first_repository

$ git init
```

On success:

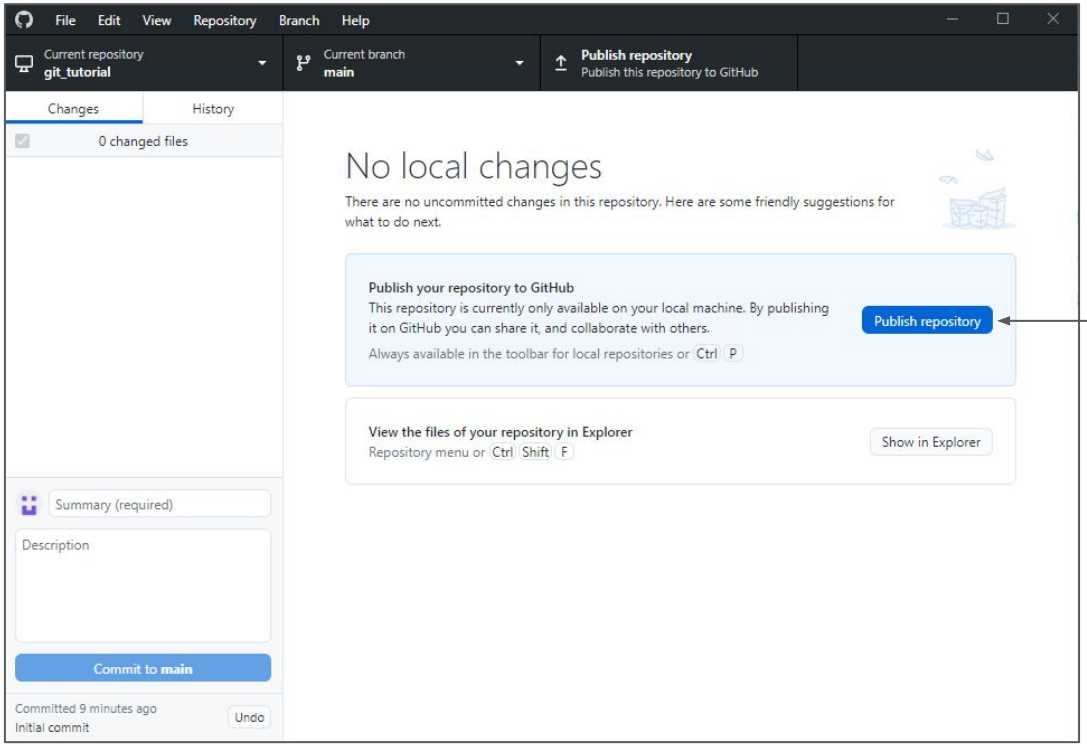
```
Initialized empty Git repository in
/Users/klcm500/practical_version_control/my_first_repository/.git/
```

Creating Your First Repository

- **README.md:** A file describing the project you have under version control. Typically written in Markdown and is nicely rendered on the GitHub page for your repository
- **.gitignore:** A file describing the things in your repository that you **don't** want to track
- **License:** a document describing the rights and permissions you grant to others who may wish to use your software

1. .gitignore file generator: https://www.toptal.com/developers/gitignore/
2. Software licence picker: https://choosealicense.com/

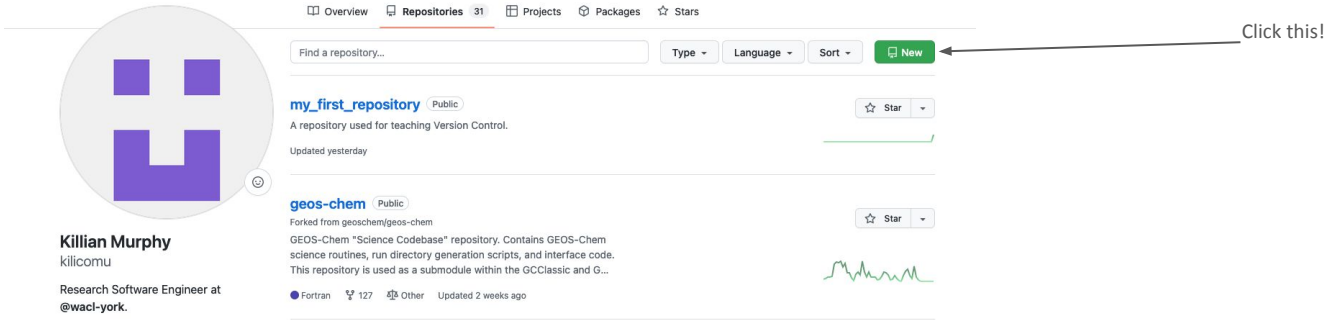
Creating Your First Repository - GUI



Now click this!

Creating Your First Repository - CLI

- Navigate to https://github.com/your-username
- Select the ‘Repositories’ tab
- Select the green ‘New’ icon




```
$ git remote add origin https://github.com/YOUR_GITHUB_USERNAME/my_first_repository
```

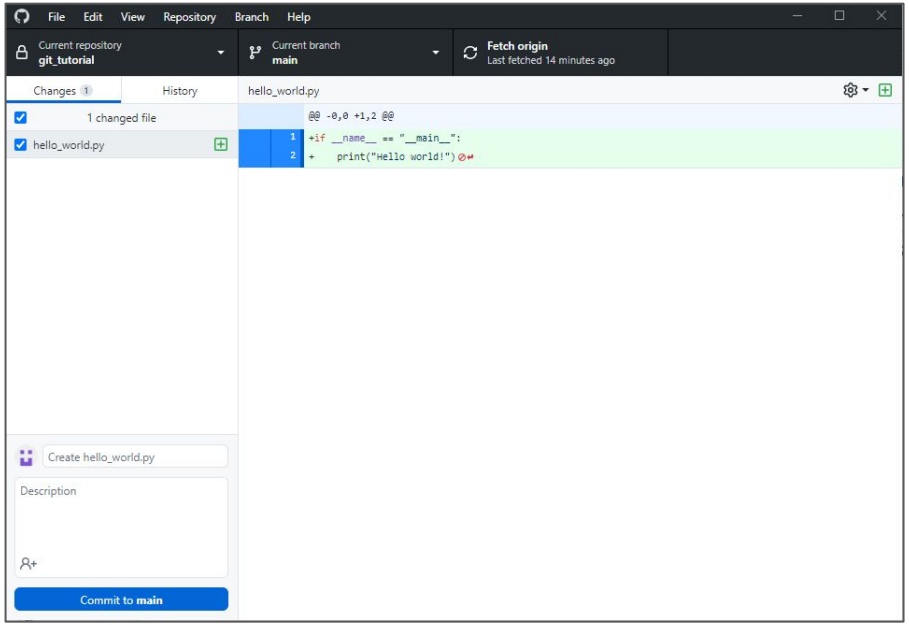
^ Creates a reference in your copy of the repository to an ‘origin’ repository on GitHub

Creating Your First Repository

- **Description**: a one-line description of the repository contents
- **Privacy**: the visibility of your repository - just you or anybody?
- **Organization**: the ownership of your repository - you or some organization (e.g. university-of-york) that you are a member of?

Making Your First Commit - GUI

- Add a file to the directory that you used when creating the repository (make sure it's not an empty file!)
- Take a look at what happens in the GitHub desktop window



Making Your First Commit - CLI

- First, run `git status` and note the output
- Then, create a file in the repository using your favourite text editor / development tool (`README.md` is a good idea!)
  - If you're stuck for something to put in there, how about the name of your favourite sandwich filling?
- Then, run `git status` again and note the different output
- Git is aware that you have added a file

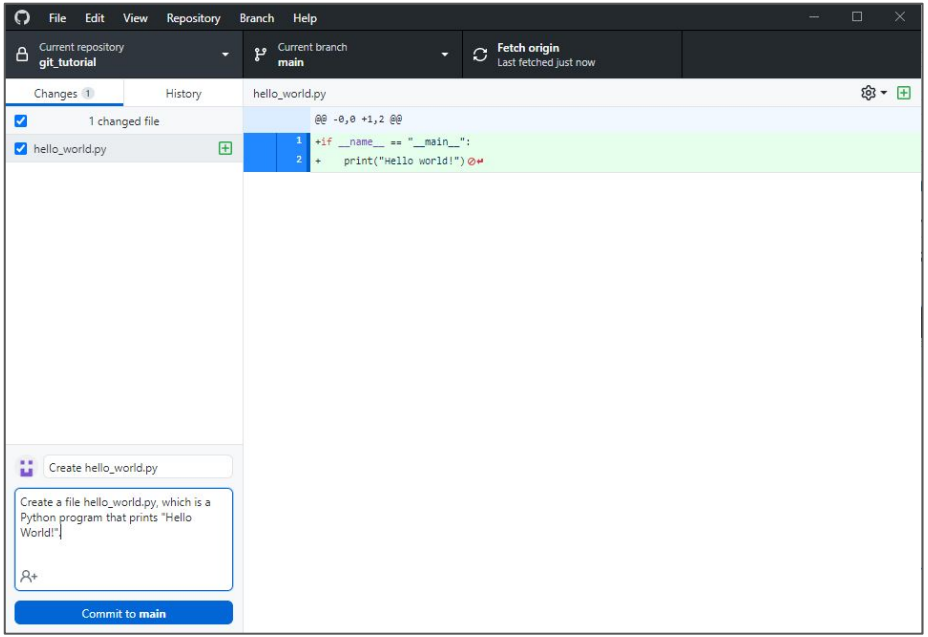
Making Your First Commit - GUI

- Add a commit label and a commit message to the box
- A nice way to think about commit labels is that they should concisely complete the following sentence:

  “This commit will…” e.g.

  “This commit will create hello_world.py”

- The commit message can contain as little or as much as appropriate to help you and others understand what the change consists of (be kind to Future You!)
- How frequently should you commit? What should be in a commit?



Making Your First Commit - CLI

- Need to set up your name and email address
- `git config --global user.name "YOUR NAME"`
- `git config --globaluser.email "your@email.address"`
- This will make your name and email address appear correctly on any commits you make to repositories
- Need to link your local repository with the repository you created on GitHub:

```
$ git remote add origin https://github.com/YOUR_GITHUB_USERNAME/my_first_repository
```

^ Creates a reference in your copy of the repository to an 'origin' repository on GitHub

Making Your First Commit - CLI

```
$ git add FILENAME
```

^ Tells git that you'd like to add this file to the commit

OR

```
$ git add -A
```

^ Tells git that you'd like to add all changed files to the commit

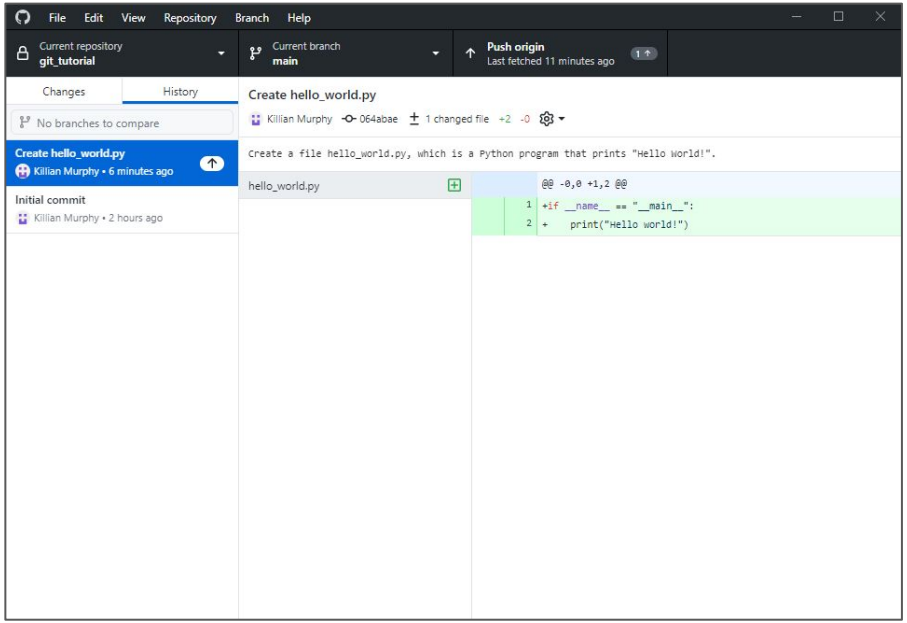
THEN

```
$ git commit
```

^ Opens up a text editor for you to add a label and message to the commit

Making Your First Commit - GUI

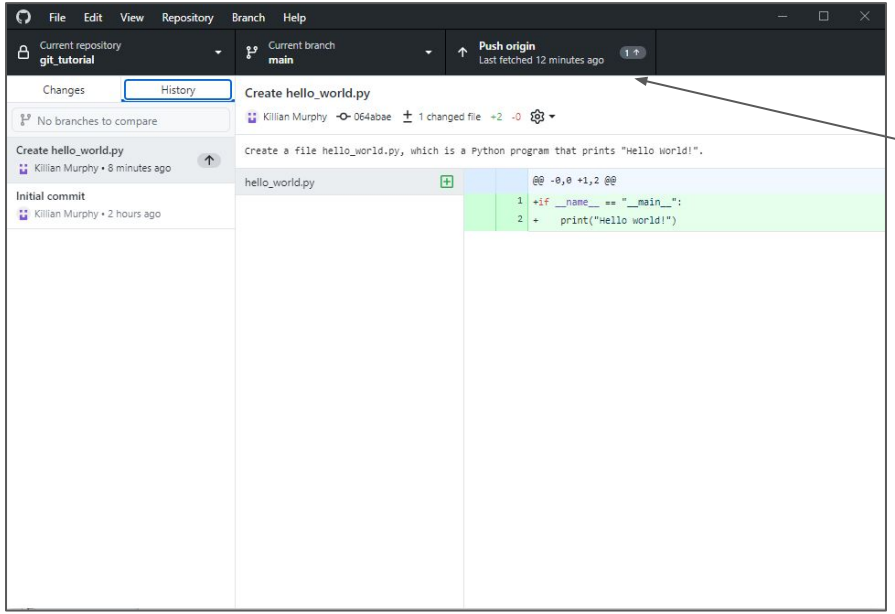
- Switch over to the “History” tab
- You should be able to see your recent commit and its details
- Right now, this exists **only on your computer**



Making Your First Commit - CLI

- Run `git log` to see the repository history
- You should see a single commit
- It will have your name and address in the ‘Author’ field
- It will have the date and time of creation in the ‘Date’ field
- Your commit label and message will be present
- This is how you communicate high-level changes in the repository to other people / yourself!

Pushing Your First Commit - GUI



Click here now!

Pushing Your First Commit - CLI


```
$ git push --set-upstream origin main
```

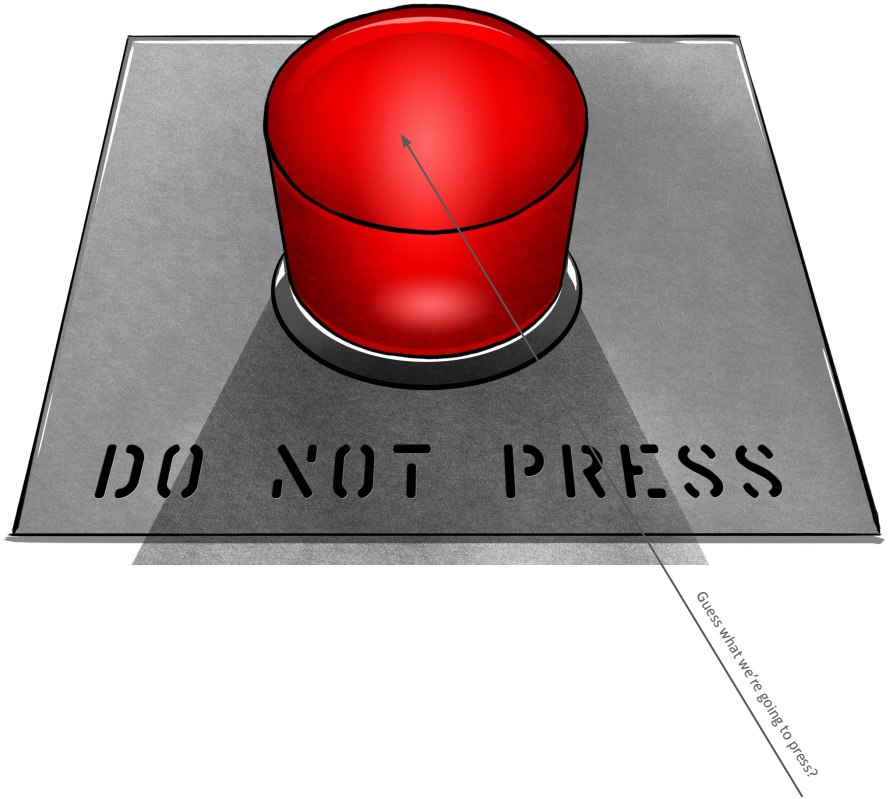
^ Only need to do it this way the first time we push to a new branch. `--set-upstream` links your branch `main` with a branch `main` in GitHub. Normally, we just need:

```
$ git push
```

Pushing Your First Commit

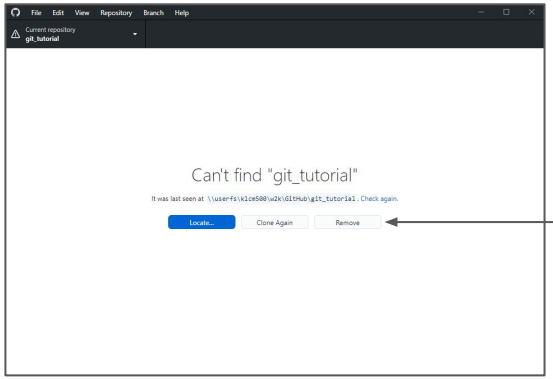
- Your repository is now on GitHub:
  https://github.com/your-username/your-repo-name

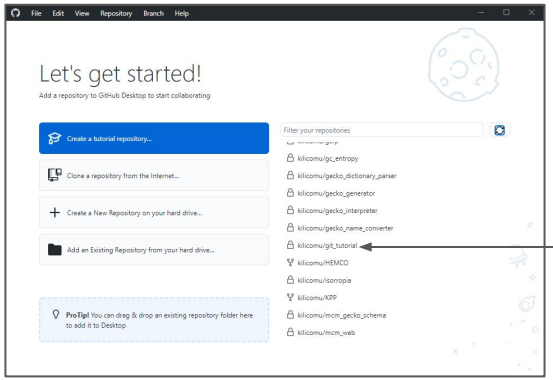
- This means it is:
  - Backed up!
  - Easily shareable!
  - Easily accessible from other machines!
- **Now delete the repository from your computer using the file explorer / finder / rm command**



Cloning Your Repository - GUI

- Getting your code and all of its history back is as easy as a few button clicks
- Once it has finished cloning, we should be back to where we were before deleting
- You can clone other GitHub users' public repositories too!





Cloning Your Repository - CLI


```
$ git clone https://github.com/your-username/your-repo-name
```

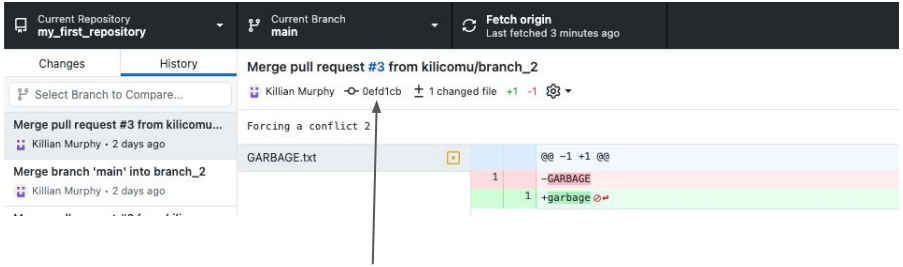
- If the repository is private, you will need to enter your GitHub username and password
- You can set up passwordless access using SSH keys, see the GitHub documentation for more details

Making More Changes

- Go ahead and make some more commits to your repository:
  - **Add a README** (everybody do this and have a look what happens on GitHub!) if you haven't already
  - Add files
  - Add contents to files that are already there
  - Delete things
- Make sure to push commits when you've made them!
- Think and chat about the way you like to work and how that might map onto version control history:
  - Do you work in big chunks and then sign off for the day?
  - Do you often switch tasks and have trouble figuring out where you were when you left off?
  - Do you program first and plan later? Or the other way around?!
- Ask any questions you'd like!

Commit Hashes

- Each commit in the history has a unique identifier associated with it, the commit hash
- This is a long string that looks something like this: `0efd1cb9e37318404b76de7c99e26fbef16ef3a3`
- You only ever need the first 7 characters of the ID!
- It is used in any Git operation that needs to refer to a specific commit



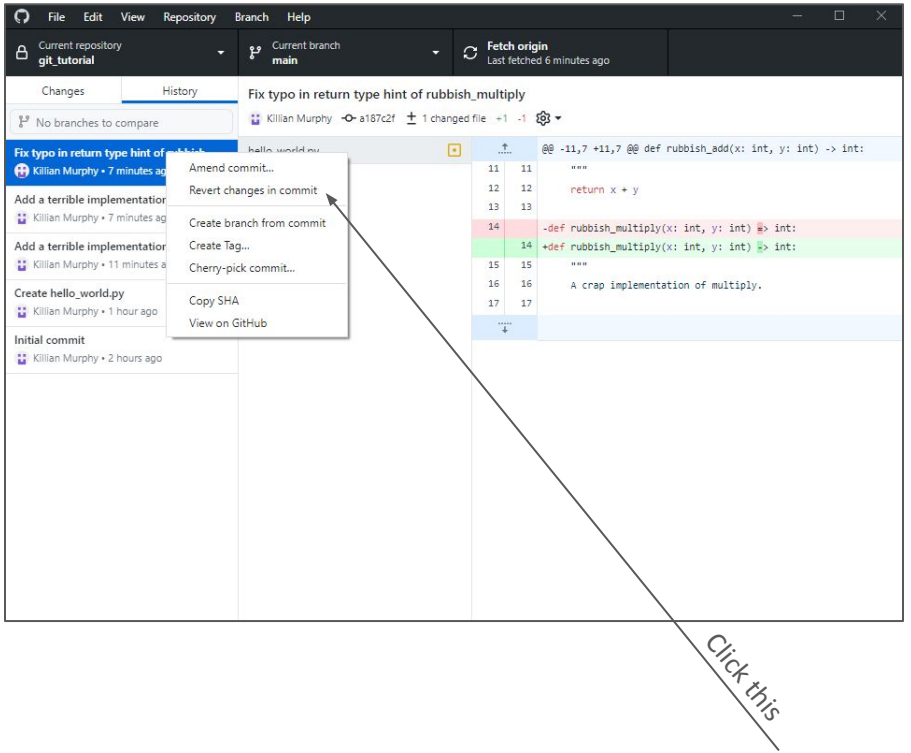
It's here!

```
klcm500@matscrn my_first_repository % git log
commit 0efd1cb9e37318404b76de7c99e26fbef16ef3a3 (HEAD -> main, origin/main, origin/HEAD)
Merge: c267c36 77caf66
Author: Killian Murphy <killian.murphy@york.ac.uk>
Date:   Mon Jan 23 16:43:31 2023 +0000
```

It's here!

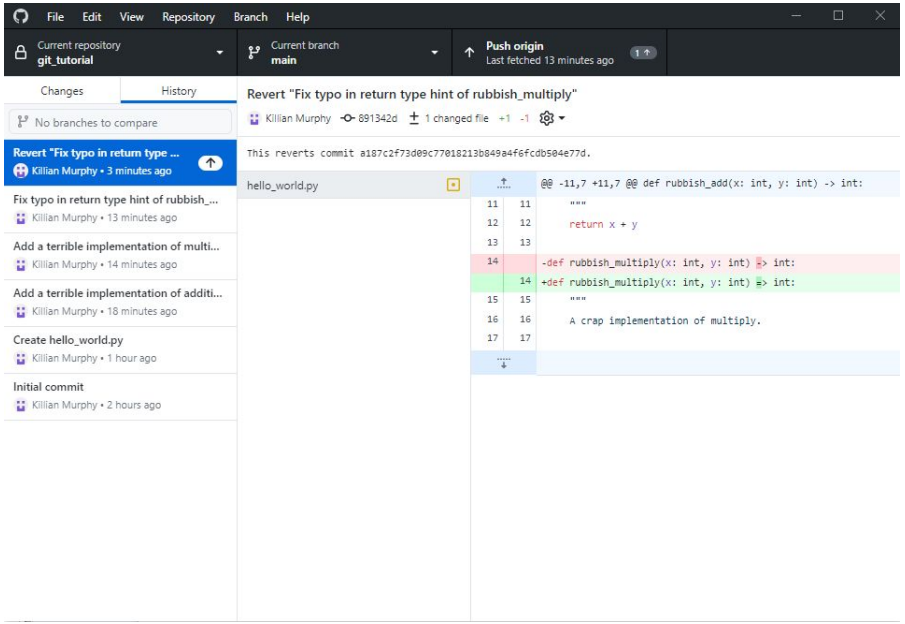
Reverting A Commit - GUI

- GitHub desktop can’t revert many commits at once
- We can easily revert commits one-by-one!
- Switch to the “History” tab and right-click the latest commit



Reverting A Commit - GUI

- We now have a new commit that reverts the previously made changes
- We can play around with this and see if it works
- If we are happy with the reversion, we can push to reflect the reversion on GitHub
- With GitHub Desktop, we need to do this for each commit we want to revert, in reverse order
- With the command line, reverting multiple commits at once is possible - not covering that today



Reverting A Commit - CLI

- First need to identify the ID of the commit we want to revert - the last commit we made. Take a look at `git log` output:

  ```
  commit f7cfe07b8d7691241ebca90c19b187da142350c1 (HEAD -> main)
  ```

  We need the first 7 characters of this ID string!

- Then we run:

  ```
  $ git revert f7cfe07
  ```

  Since we are making a new commit which reverts the previous commit, we are prompted for a label and a message. This is a good opportunity to document *why* we are reverting a commit

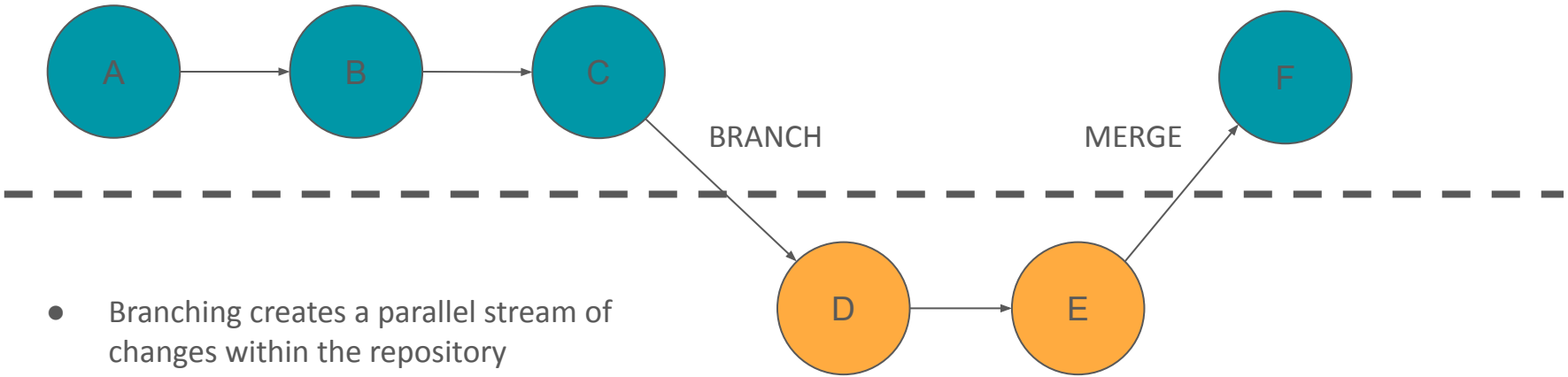
- If we are happy with the reversion, we can push the changes as with any other commit

Summary

- **Create** a repository, or **clone** an existing repository
- Make changes in the repository and **commit** them, adding a **label** and a **message**
- **Push** the changes to GitHub to synchronize
- **Revert** commits if you find you have broken something!

Part 3: Working Collaboratively

Branches - Overview



- Branching creates a parallel stream of changes within the repository
- Changes on one branch aren't reflected on another branch, unless you want them to be!

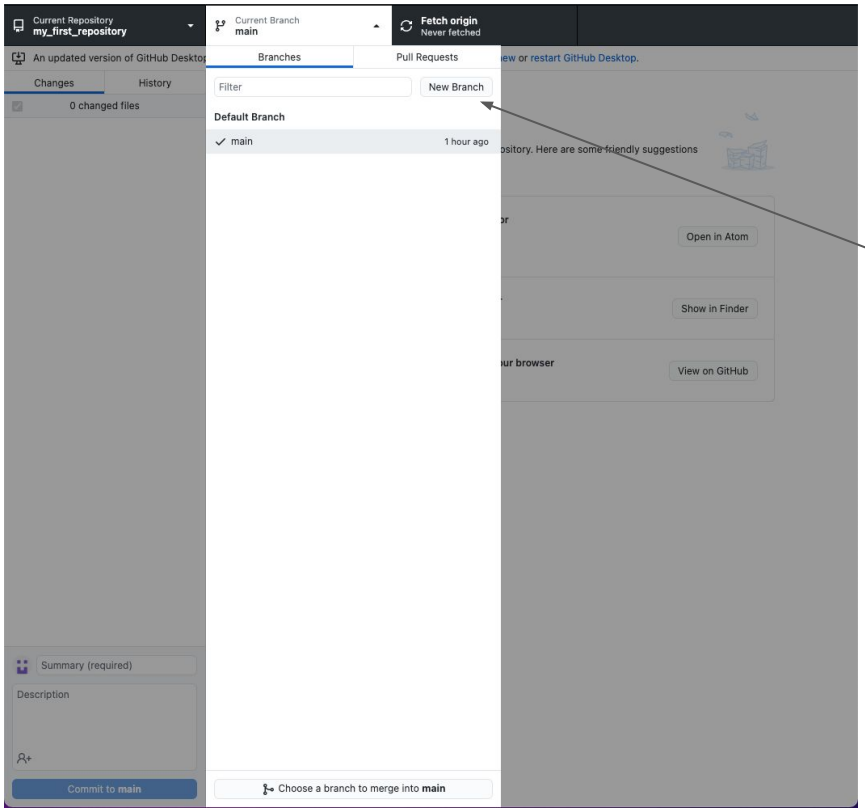
Branches - Overview

- Branches are great for working on new things once you've got something working - make changes on a branch knowing that they won't affect changes on the 'main' branch
- Branches are great for working together - multiple people can work on different changes without interfering with each other
- At some point you will want to bring all of these changes back into one place - this is called a 'merge'

Git Glossary: Branching

- **Branch**: a set of changes in your repository, being tracked in parallel to your ‘main’ branch
- **Merge**: the act of bringing changes from one branch onto another
- **Conflict**: a situation where competing changes have been made to some part of your repository
- **Pull Request**: a way of reviewing, labelling, and discussing a merge before it takes place

Creating A Branch - GUI



Click this and give the branch a name!

Creating A Branch - CLI

```
$ git branch my_first_branch
```

^ Create a new branch in your local repository called ‘my_first_branch’

```
$ git switch my_first_branch
```

^ Switch to the newly created branch

Publishing Branches

- The branch currently only resides in our local repository
- We can publish the branch to GitHub in the same way we published the ‘main’ branch at the start of the tutorial:
  - GUI users can click ‘Publish branch’
  - CLI users can run `git push --set-upstream origin BRANCH_NAME` where BRANCH_NAME is the name you chose for the new branch
- Now other people could see your branches when they clone your repository

Working On Branches

- Go ahead and make at least one new commit to your branch
- Make sure to add a label and commit message to the branch
- Once you have made at least one new commit, switch back to your main branch. What do you notice?
- Have a look in your file explorer / finder / directory listing in your terminal in between switching branches

Working On Branches

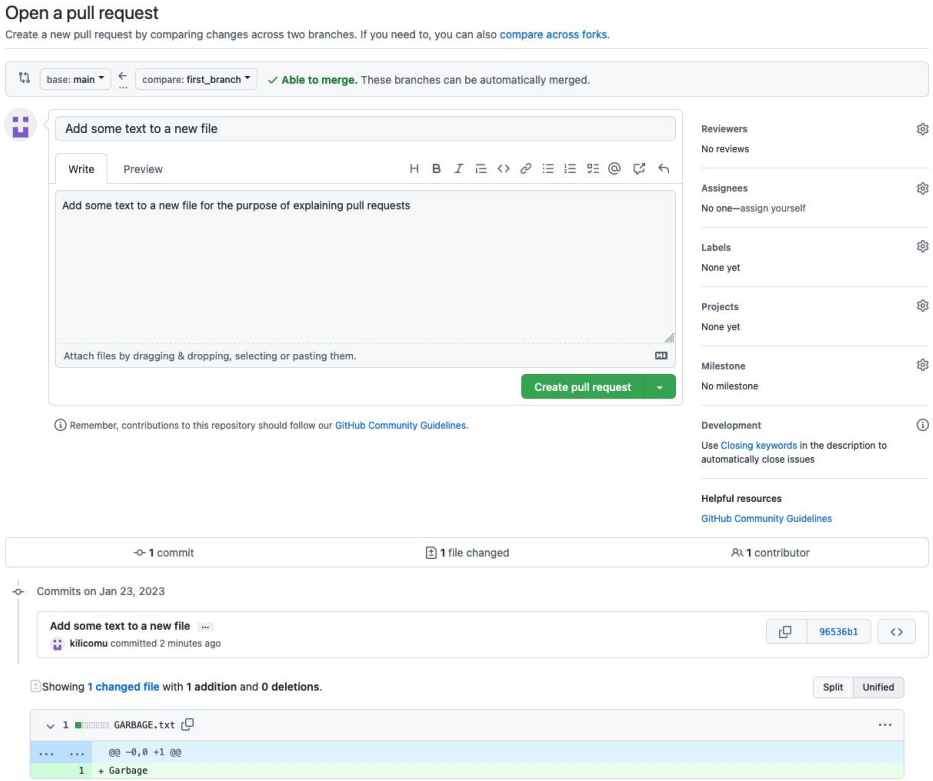
- Git is keeping track of changes to these branches separately
- Files that only exist on one branch will not be visible to you if you have switched away from that branch
- If we now want our main branch to reflect changes made on our new branch, we need to merge the new branch onto the main branch
- Before we do this, we can review the changes using a Pull Request

Creating A Pull Request

- Navigate to https://github.com/your-username/your-repo-name
- Select the ‘Pull requests’ tab
- Select ‘New pull request’
- We need to select two branches to be compared for a merge - we want to set ‘main’ as our ‘base’ branch (the branch onto which changes will be merged) and our new branch as the ‘compare’ branch, the branch from which changes will be taken
- GitHub will show us some information about the changes that we are looking to merge
- With these branches correctly selected, select ‘Create pull request’

Creating A Pull Request

- Now we can add a title and formatted description of the changes to make it easy for us and others to understand the changes to be merged
- PRs are a good opportunity for you to describe the changes to yourself, review them, and make sure you are happy with them, before merging them with your main branch
- If there are multiple people working on your repository this is a great time for them to have a look at your changes and add any comments they might have!
- Let's add a nice description and click 'Create pull request', then see if we can get somebody else to review your changes



Granting Access To Your Repo

- Navigate to your repository ‘Settings’ (the gear icon on the repository web page)
- Select ‘Collaborators’ from the menu
- Partner up with the person sat next to you and enter their GitHub username into the search box, selecting them from the results
- Check your email for an invitation to collaborate on the repository and follow the instructions to become a collaborator

Experiment With Pull Requests

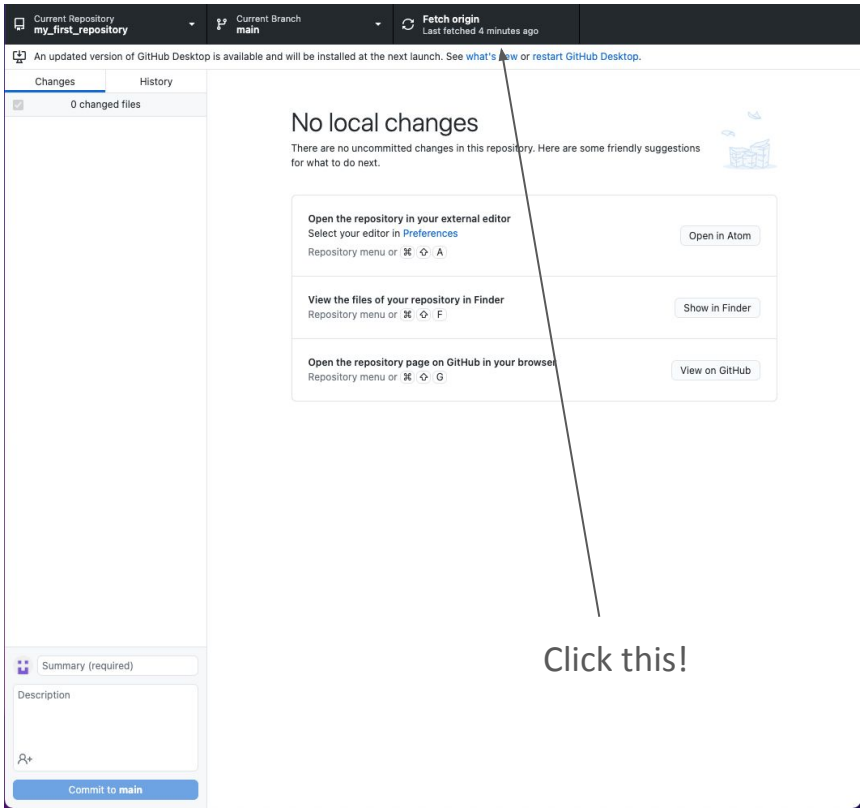
- Try leaving some comments on each other's Pull Requests
- Try adding an assignee to your Pull Request:
  - This is an indicator that somebody is responsible for resolving the Pull Request - requesting more changes, rejecting it, accepting it etc.
- Try adding some labels to your Pull Request:
  - These let you filter Pull Requests based on the labels associated with them

Merging A Pull Request

- Once you are done experimenting with your Pull Requests, select ‘Merge pull request’:
  - This takes the changes from the ‘compare’ branch - the branch you created earlier - and merges them onto the ‘base’ branch - the main branch of your repository
  - In this case, the changes should be able to be merged automatically
  - After merging, you can select ‘Delete branch’ - we are finished merging the changes and in this case don’t need to keep it!
- We now need to make sure our local repositories have kept up with the changes that have happened on GitHub…

Keeping Repositories Up To Date - GUI

- We have made some changes in GitHub that we need to reflect in our local repository!
- In the GitHub Desktop app, we can use the ‘Fetch origin’ button to get the latest changes from GitHub
- If we switch back to the main branch after doing so, we will see a Merge commit in the history
- We can also delete our branch that has been merged, since we are finished with it



Keeping Repositories Up To Date - CLI


```
$ git switch main
```

^ Switch back to our main branch, if we aren't already on it

```
$ git pull
```

^ Get the changes from GitHub and reflect them in our local repository

```
$ git branch -d my_branch
```

^ Delete the local copy of 'my_branch', as we have finished with it

Introducing Conflicts

- Conflicts happen when something in a file has been changed in more than one place
- Git doesn't know what you want the file to contain, so you have to help it!
- One way this can happen is when multiple people work on e.g. the same bit of code on their own branches, and you want to merge those branches back together

Introducing Conflicts

- With the person next to you, decide on one of your repositories to use for this exercise (you can work alone if you prefer)
- Make sure both of you have a copy of the chosen repository (go back and look up how to clone a repository if you can't remember how!)
- Each of you create a new branch from the main branch
- Both of you make changes to same part of a file in the repository, commit the changes, and push them

Introducing Conflicts

- We now have two branches to merge onto the main branch!
- Set up a Pull Request for each of these branches
- You should be able to merge the Pull Request for one of these branches without conflicts
- After merging the first one, the second Pull Request should now tell you that there are conflicts to be resolved
- We have to tell Git what we want the conflicting file to look like in order to continue

Resolving Conflicts

- If you select ‘Resolve conflicts’ on the Pull Request, GitHub will show you something like the above
- This is Git’s conflict syntax - anything above the line of ‘=’ characters is what that section of the file looks like in the conflicting branch
- Anything below the line of ‘=’ characters is what that section of the file looks like on the ‘base’ branch
- We can choose how we want to resolve the conflict by removing everything we don’t want to keep, including the Git conflict markers
- Once you are done, select ‘Mark as resolved’ - you can now ‘Commit merge’ to resolve the conflicts and continue

Summary

- Create a **branch** in a repository, allowing separate streams of changes to be tracked
- **Switch** between branches in a repository if you need to work on multiple streams of changes simultaneously
- Create **pull requests** when you want to merge changes from a branch into another branch
- **Merge** a pull request when you and collaborators are happy with the changes
- **Pull** changes from GitHub to synchronise your local repository
- **Resolve conflicts** when there are overlapping changes to some part of your repository

Additional Resources

- https://swcarpentry.github.io/git-novice/
- https://chryswoods.com/introducing_git/
- https://docs.github.com/en/get-started/quickstart/hello-world