Introduction to Version Control with Git and GitHub

Working Alone and Working Collaboratively

Killian Murphy, Peter Hill
University of York

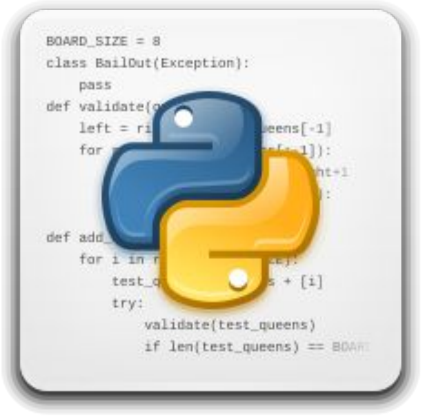
Goals for the session:

- Basic understanding of what version control, git, and GitHub are
- Knowledge of a few important git-related terms
- Know how to:
  - create a version controlled project
  - make changes under version control
  - undo changes to a project
  - make a local copy of a project
  - work with others on a version controlled project

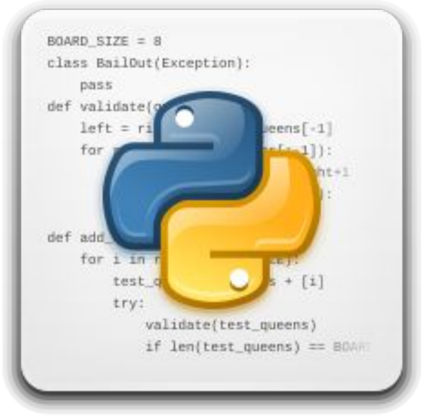
Part 1: Introductory Information

What is Version Control?

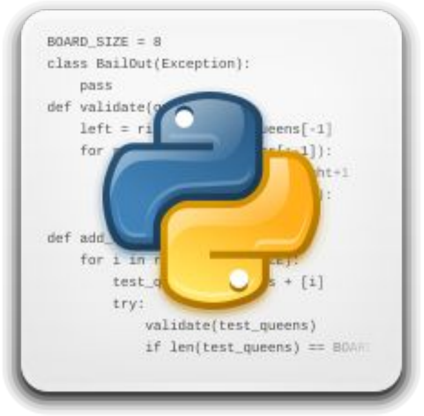
Any means by which you track changes to something:



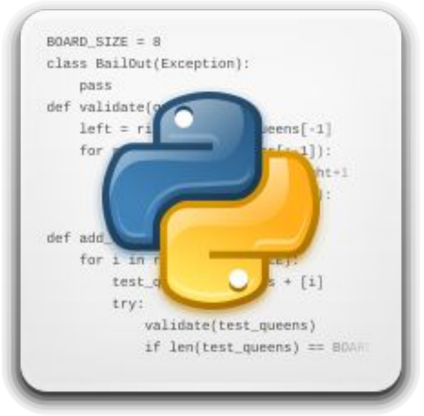
work.py



work_final.py



work_final_2.py



work_final_2
_really_final.py

We've all done it!

Is this effective version control?

What is Version Control?

- Structured way of keeping track of changes to files
- Implemented 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
- Many tools (like IDEs) have git integrations

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 all be in sync
- One of the most important programming tools you can benefit from becoming comfortable with

What is GitHub?

- A service for hosting, organising, and collaborating on projects that are under version control
    - Other git services are available
- Provides a consistent way to store and share version-controlled projects
- Can act as a portfolio of sorts
- Has a whole host of other features (project management, continuous integration, etc.)
- See the PlasmaFAIR GitHub organisation for an example

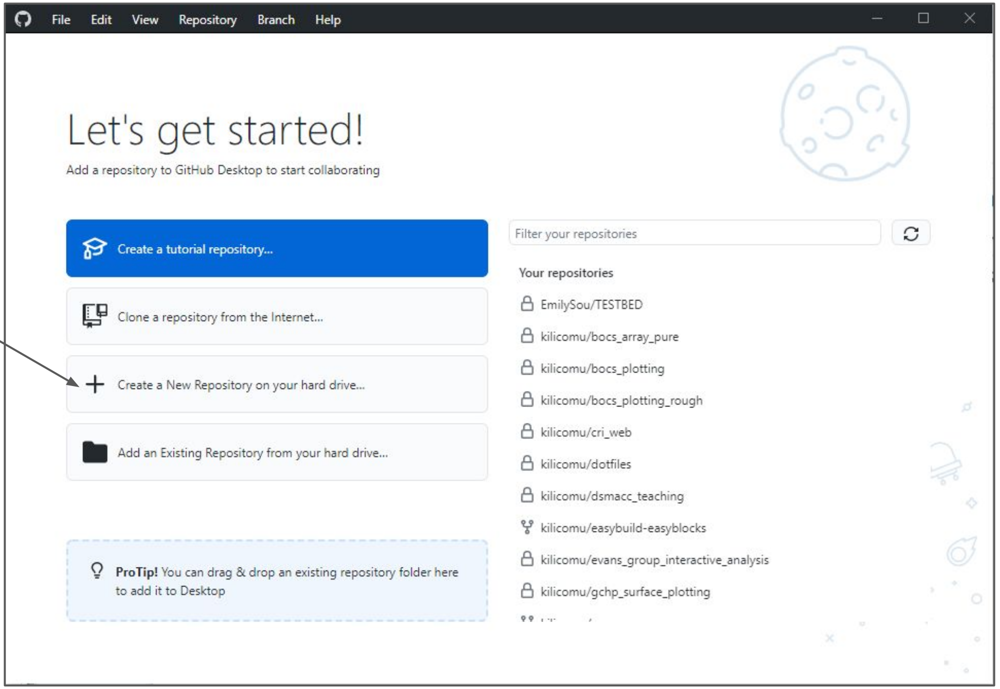
Git Glossary: Basics

- **Repository (repo)**: a project under version control
- **History**: timeline of changes to your repo
- **Clone**: make a local copy of a repo, including its history
- **Commit**: a labelled set of specific change to files in your repo
- **Push / pull**: synchronise history with another copy of your repo
- **Working tree**: the current state of your project and files on disk
- **Track**: keep under version control

Part 2: Working Alone

Creating Your First Repository - GUI

Click this!


File
Edit
View
Repository
Branch
Help
Create a tutorial repository...
Clone a repository from the Internet...
Create a New Repository on your hard drive...
Add an Existing Repository from your hard drive...
Filter your repositories
Your repositories

Let's get started!

Add a repository to GitHub Desktop to start collaborating

ProTip! You can drag & drop an existing repository folder here to add it to Desktop

EmilySou/TESTBED
kilicomu/bocs_array_pure
kilicomu/bocs_plotting
kilicomu/bocs_plotting_rough
kilicomu/cri_web
kilicomu/dotfiles
kilicomu/dsmacc_teaching
kilicomu/easybuild-easyblocks
kilicomu/evans_group_interactive_analysis
kilicomu/gchp_surface_plotting

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 - CLI

```
git status
```

- This is going to be our "default" command — if you're ever not sure what to do, run `git status`, it often has useful info!

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
  - In British English this is correctly spelled "licence" which is a noun — "license" is the verb, c.f. "advice/advise"

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

Creating Your First Repository

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

Adding first file

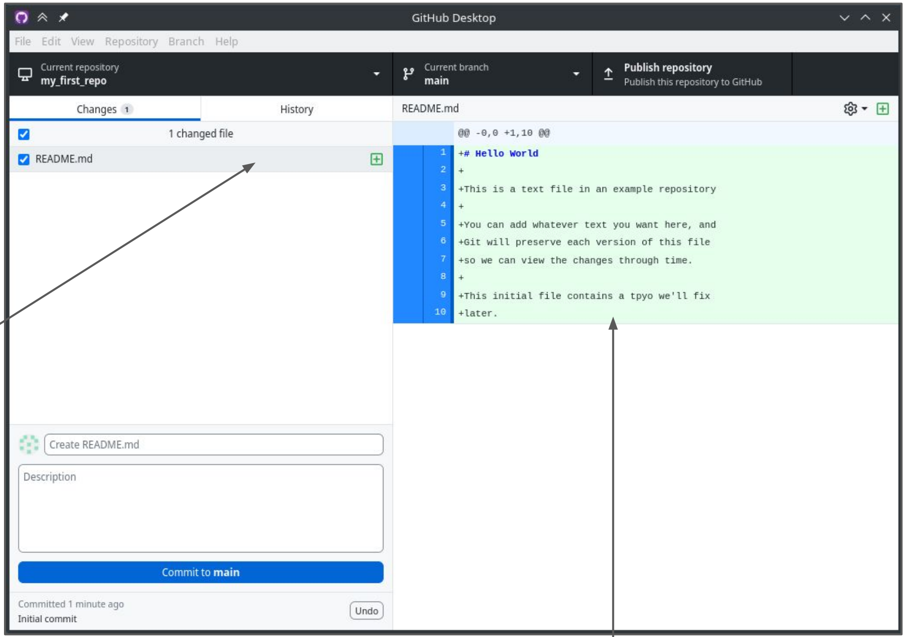
- Use your favourite text editor to create **README.md**

```
# Hello World

This is a text file in an example repository

You can add whatever text you want here, and
Git will preserve each version of this file
so we can view the changes through time.

This initial file contains a tpyo we'll fix
later.
```

Making Your First Commit - GUI

- Take a look at what happens in the GitHub desktop window

File status (added in this case)



```
@@ -0,0 +1,10 @@
+# Hello World
+
+This is a text file in an example repository
+
+You can add whatever text you want here, and
+Git will preserve each version of this file
+so we can view the changes through time.
+
+This initial file contains a tpyo we'll fix
+later.
```

Our changes

Making Your First Commit - CLI

- Run `git status` again and note the different output
- Git is aware that you have added a file

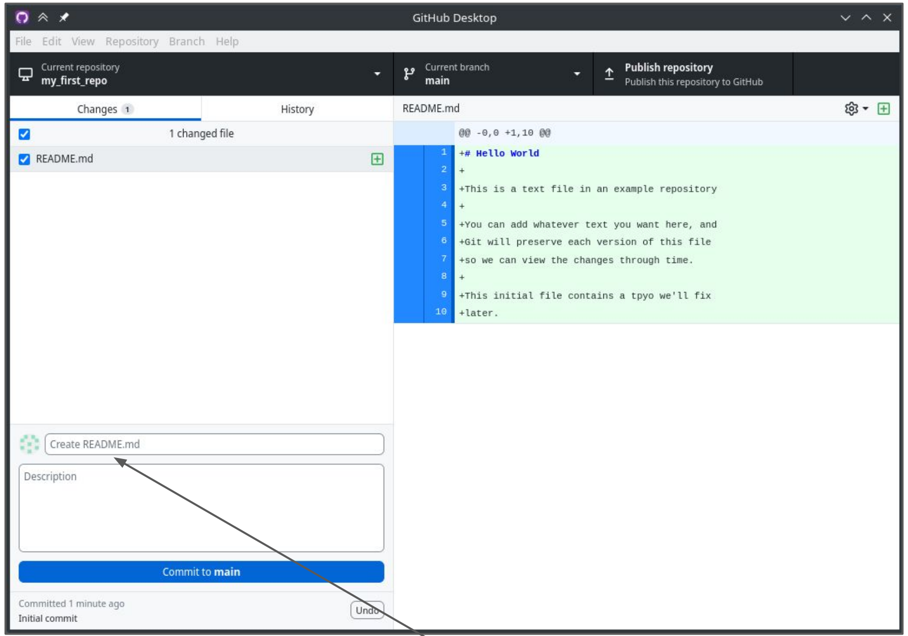
Making Your First Commit - GUI

- 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 README.md"

- 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*!)
- Best practice is to keep first line short, like an email subject line, and optionally a longer explanation below, separated by a blank line
- How frequently should you commit? What should be in a commit?



```
README.md
@@ -0,0 +1,10 @@
+# Hello World
+
+This is a text file in an example repository
+
+You can add whatever text you want here, and
+Git will preserve each version of this file
+so we can view the changes through time.
+
+This initial file contains a tpyo we'll fix
+later.
```

Create README.md

Description

Committed 1 minute ago
Initial commit

oh, look what the default message is for a new file! clever machine

Making Your First Commit - CLI

- Need to set up your name and email address
  - `git config --global user.name "YOUR NAME"`
  - `git config --global user.email "your@email.address"`
- This will make your name and email address appear correctly on any commits you make to repositories
- Only need to do this once on a system!
- It's also a good idea to set the text editor to whatever your favourite is:
  - `git config --global core.editor "nano"`
  - Nano is a pretty sensible default if you don't have a favourite
    - Although Emacs is much better if you want an editor that does everything and you enjoy tinkering!
  - Press F1 in nano for help

Making Your First Commit - CLI

```
git add README.md
```

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

THEN

```
git commit
```

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

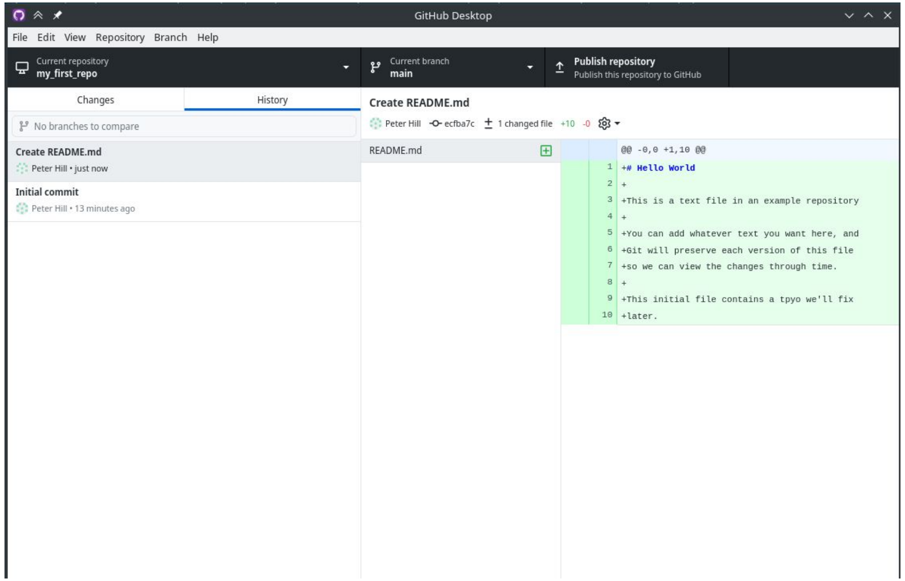
OR

```
git commit -m "<message>"
```

^ To immediately write a commit message (without a longer description)

Making Your First Commit - GUI

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



Create README.md
Peter Hill • just now

Initial commit
Peter Hill • 13 minutes ago

Create README.md
Peter Hill ecfba7c 1 changed file +10 -0

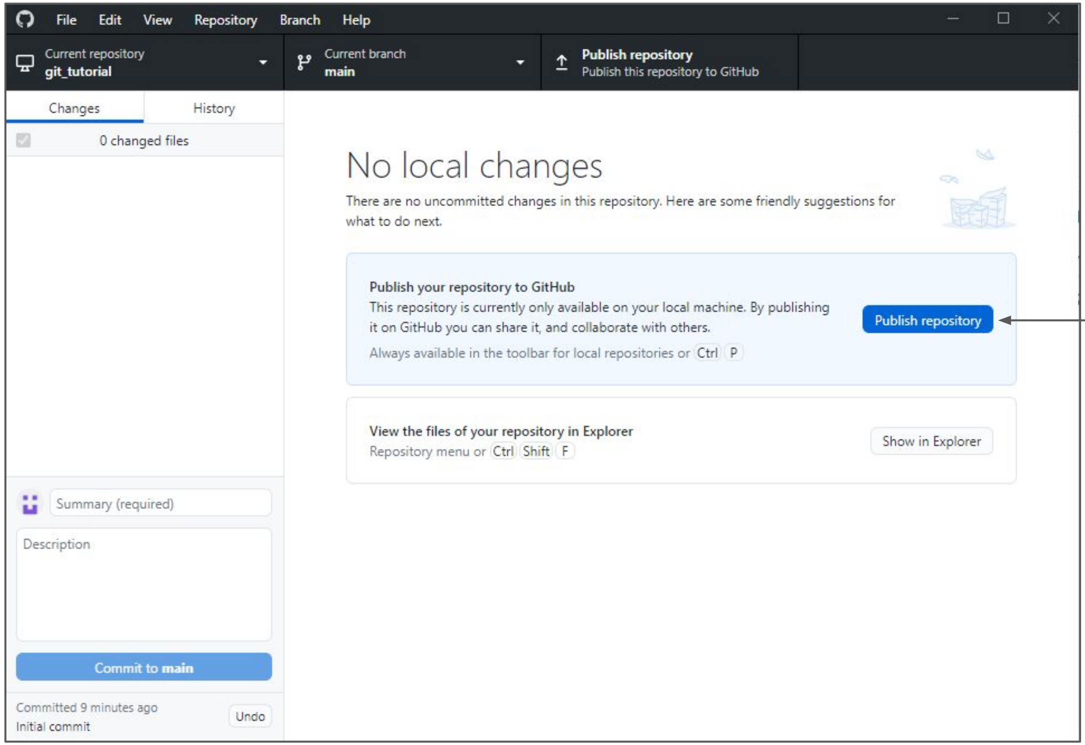
README.md

```
@@ -0,0 +1,10 @@
+# Hello World
+
+This is a text file in an example repository
+
+You can add whatever text you want here, and
+Git will preserve each version of this file
+so we can view the changes through time.
+
+This initial file contains a tpyo we'll fix
+later.
```

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!

Making a repo on Github - GUI



No local changes

There are no uncommitted changes in this repository. Here are some friendly suggestions for what to do next.

Publish your repository to GitHub
This repository is currently only available on your local machine. By publishing it on GitHub you can share it, and collaborate with others.
Always available in the toolbar for local repositories or Ctrl P

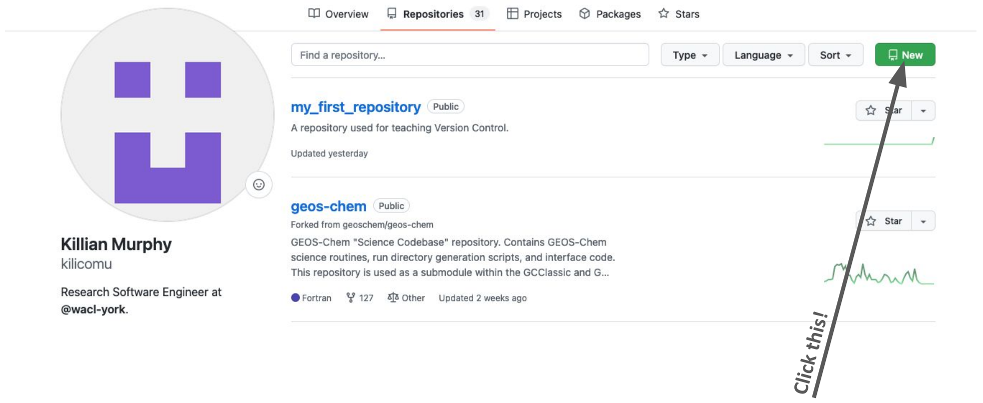
View the files of your repository in Explorer
Repository menu or Ctrl Shift F

Committed 9 minutes ago
Initial commit

Now click this!

Making a repo on Github - CLI

- Navigate to https://github.com/your-username
- Select the ‘Repositories’ tab
- Select the green ‘New’ icon
- *Don't* add a **`README`** or **`.gitignore`**
- Make it **public** rather than **private**, will simplify things later!
- From the front page of your new repo, press the green **Code** button and copy the URL
- Paste it into the following command:



Overview Repositories 31 Projects Packages Stars

Find a repository... Type Language Sort New

Killian Murphy
kilicomu
Research Software Engineer at @wacl-york.

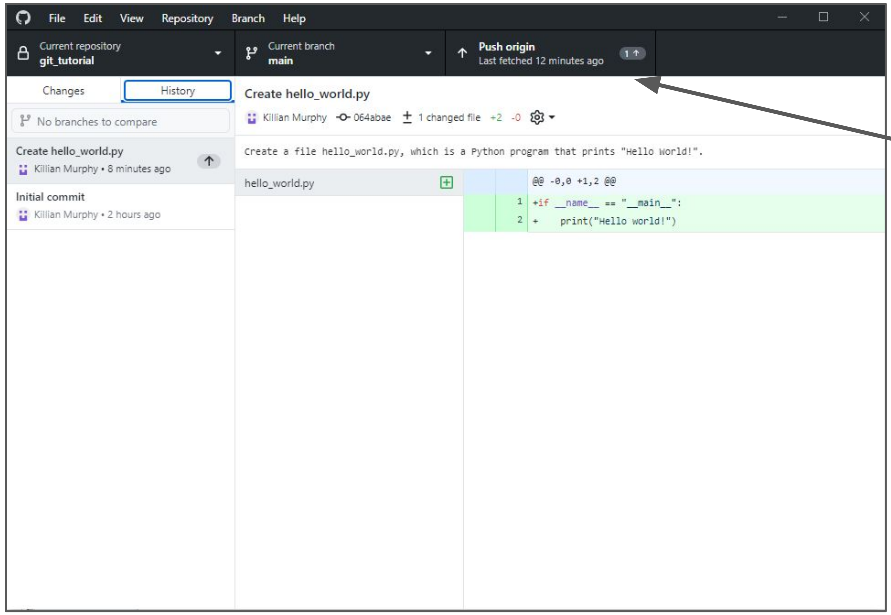
my_first_repository Public
A repository used for teaching Version Control.
Updated yesterday

geos-chem Public
Forked from geoschem/geos-chem
GEOS-Chem "Science Codebase" repository. Contains GEOS-Chem science routines, run directory generation scripts, and interface code. This repository is used as a submodule within the GCClassic and G...
Fortran 127 Other Updated 2 weeks ago

```
git remote add origin <YOUR REPO URL>
```

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

Pushing Your First Commit - GUI



```
Create a file hello_world.py, which is a Python program that prints "Hello World!".
```

```
@@ -0,0 +1,2 @@
+if __name__ == "__main__":
+    print("Hello World!")
```

Click here now!

Pushing Your First Commit - CLI

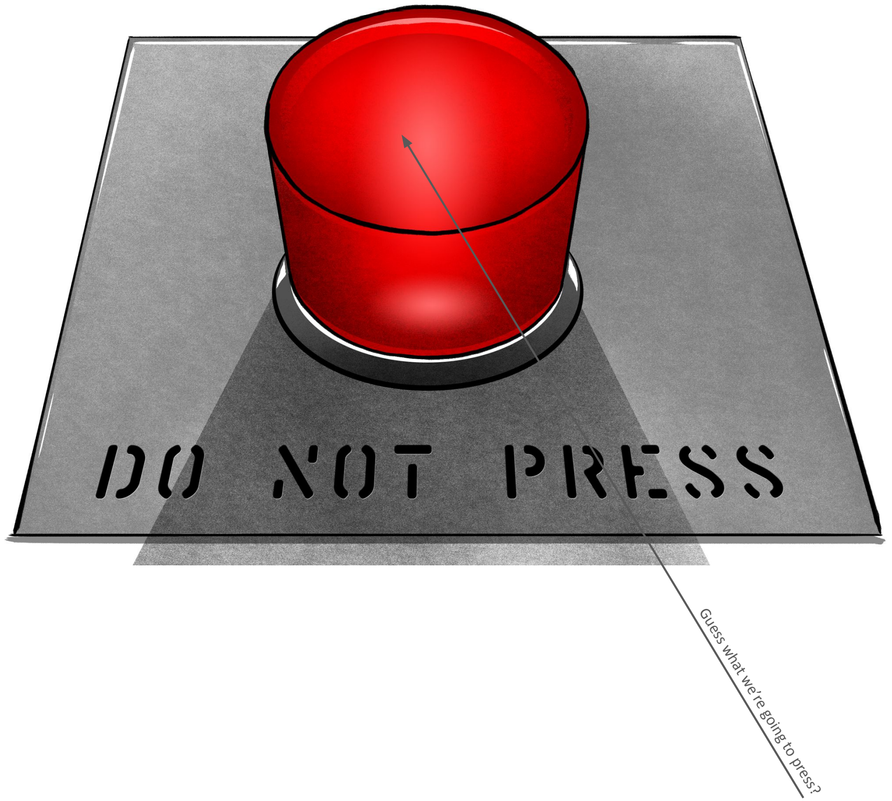
```
git push --set-upstream origin main
```

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

```
git push
```

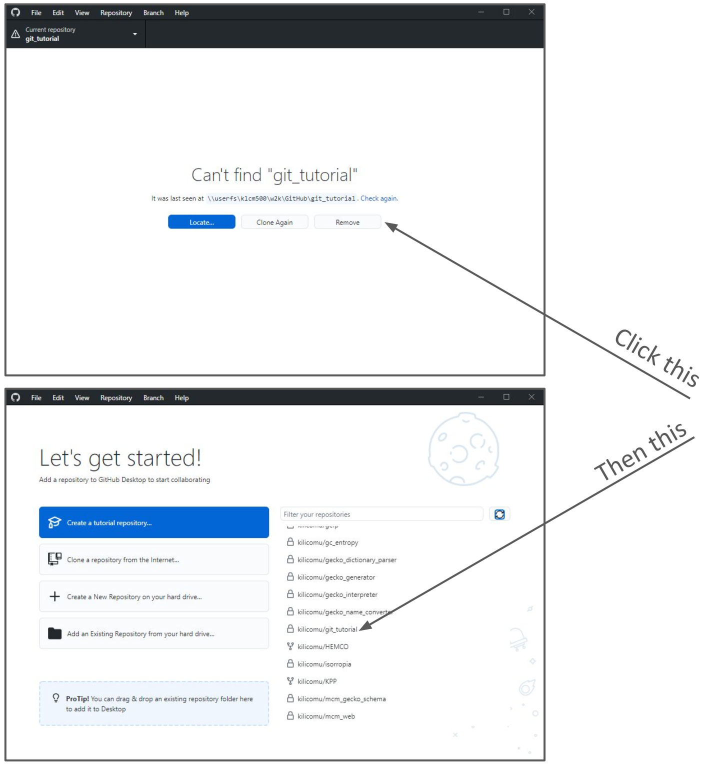
Pushing Your First Commit

- Your repository is now on GitHub: https://github.com/<username>/<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!


File
Edit
View
Repository
Branch
Help
Current repository
git_tutorial
Locate...
Clone Again
Remove
Click this
Then this
Create a tutorial repository...
Clone a repository from the Internet...
Create a New Repository on your hard drive...
Add an Existing Repository from your hard drive...
Filter your repositories

Can't find "git_tutorial"

It was last seen at `\\userfs\klcm500\w2k\GitHub\git_tutorial`. Check again.

Let's get started!

Add a repository to GitHub Desktop to start collaborating

ProTip! You can drag & drop an existing repository folder here to add it to Desktop

kilicomu/gc_entropy
kilicomu/gecko_dictionary_parser
kilicomu/gecko_generator
kilicomu/gecko_interpreter
kilicomu/gecko_name_converter
kilicomu/git_tutorial
kilicomu/HEMCO
kilicomu/isorropia
kilicomu/KPP
kilicomu/mcm_gecko_schema
kilicomu/mcm_web

Cloning Your Repository - CLI

```
git clone https://github.com/<username>/<repo-name>
```

- If the repository is private, you will need to set up a personal access token before you can clone the repository
- You can also set up passwordless authentication using SSH keys, see the GitHub documentation for more details
    - This is a Good Idea!

Making Changes

- Let's fix up the deliberate mistake in our README

```
# Hello World

This is a text file in an example repository

You can add whatever text you want here, and
Git will preserve each version of this file
so we can view the changes through time.

This initial file contains a typo we'll fix
later.
```

- Add the file and commit it
    - What would a good commit message be for this change? Discuss

Viewing diffs - CLI

```
$ git status
On branch main
Changes not staged for commit:
 (use "git add <file>..." to update what will be committed)
 (use "git restore <file>..." to discard changes in working directory)
       modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")

$ git diff
diff --git a/README.md b/README.md
index b277f80..ea21a73 100644
--- a/README.md
+++ b/README.md
@@ -6,5 +6,5 @@ You can add whatever text you want here, and
Git will preserve each version of this file
so we can view the changes through time.

-This initial file contains a tpyo we'll fix
+This initial file contains a typo we'll fix
later.
```

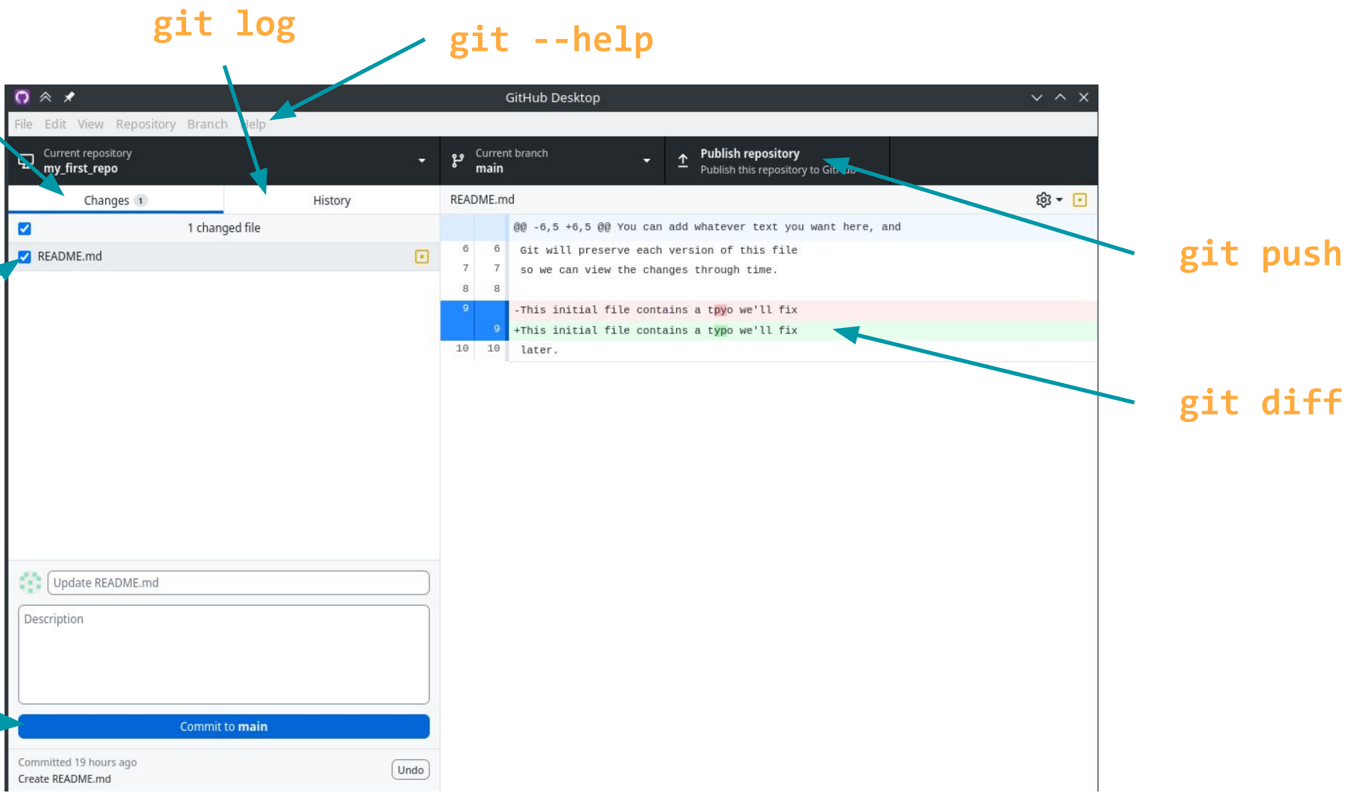
This header is because diff is a more general tool for comparing text files

This line shows the line number and context (e.g. function name)

This line has been removed

This line has been added

Git commands in the GUI



```
@@ -6,5 +6,5 @@ You can add whatever text you want here, and
 Git will preserve each version of this file
 so we can view the changes through time.

-This initial file contains a tpyo we'll fix
+This initial file contains a typo we'll fix
 later.
```

Making More Changes

- Go ahead and make some more commits to your repository:
  - On the command line, don't forget the two-step dance: add, then commit
- Make sure to push commits when you've made them!
  - You don't have to push after every single commit, as long as you remember to push at some point
- 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!

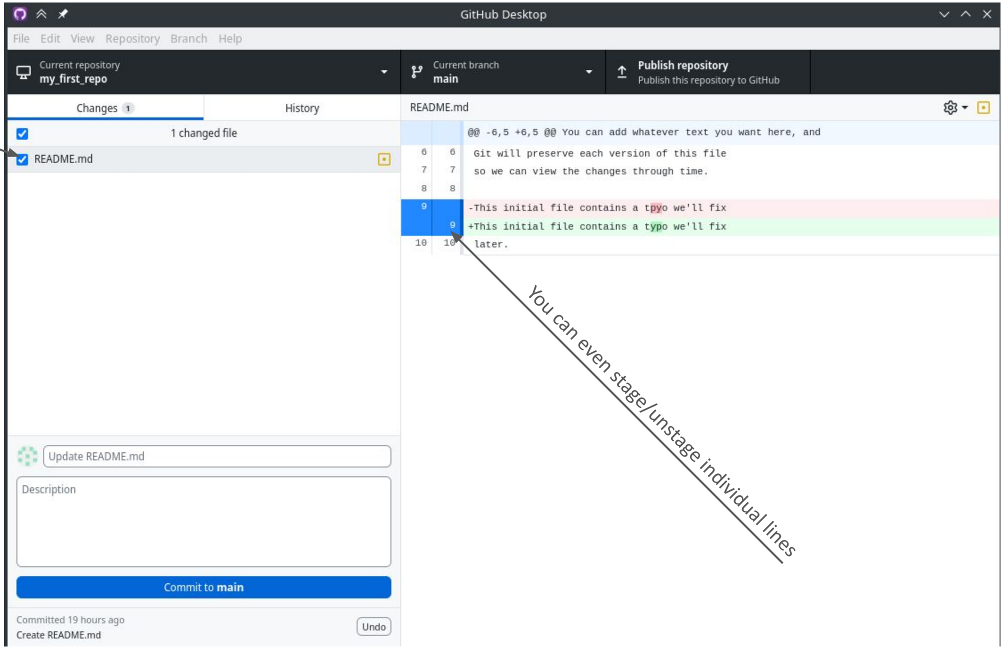
Undoing Changes

- One of the purposes of version control is to let you undo changes
- There are different changes we might want to undo:
  - Changes to the working tree
  - Change files staged for the next commit
  - Changes made in previous commits
  - History (!!) (not covered here)

Undoing Changes to Working Tree

- The simplest!
- Github Desktop: right-click -> **Discard Changes**
- CLI: `git restore <file>...`
  - For example: `git restore README.md`
  - If you forget, run `git status` for a reminder
- **DANGER:** git isn't tracking these files, so there's no way to undo this undo!

Undoing Staged Changes



```
@@ -6,5 +6,5 @@ You can add whatever text you want here, and
 Git will preserve each version of this file
 so we can view the changes through time.

-This initial file contains a tpyo we'll fix
+This initial file contains a typo we'll fix
 later.
```

Stage/unstage

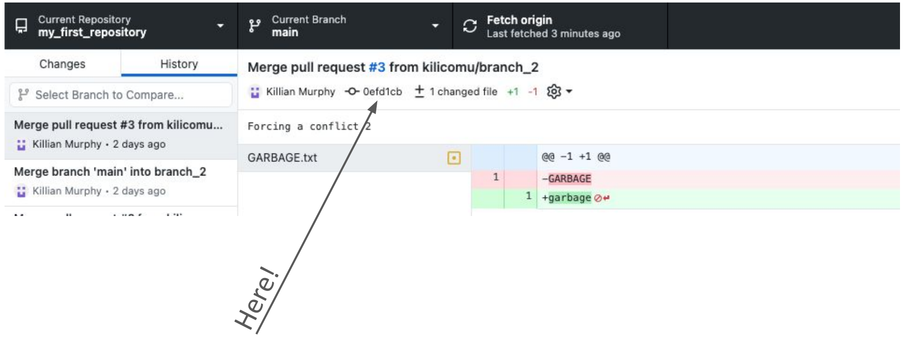
You can even stage/unstage individual lines

Undoing Staged Changes

- CLI: `git restore --staged <file>...`
  - For example: `git restore --staged README.md`
  - If you forget, run `git status` for a reminder
- This is exactly the opposite of `git add <file>...`

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



Current Repository my_first_repository | Current Branch main | Fetch origin Last fetched 3 minutes ago

Changes | History

Select Branch to Compare...

Merge pull request #3 from kilicomu...
Killian Murphy • 2 days ago

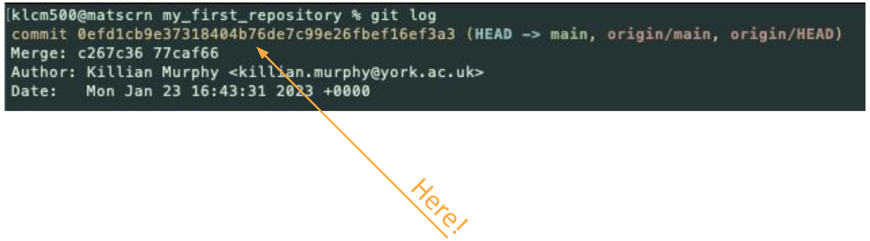
Merge branch 'main' into branch_2
Killian Murphy • 2 days ago

Merge pull request #3 from kilicomu/branch_2
Killian Murphy 0efd1cb 1 changed file +1 -1

```
Forcing a conflict 2
```

GARBAGE.txt

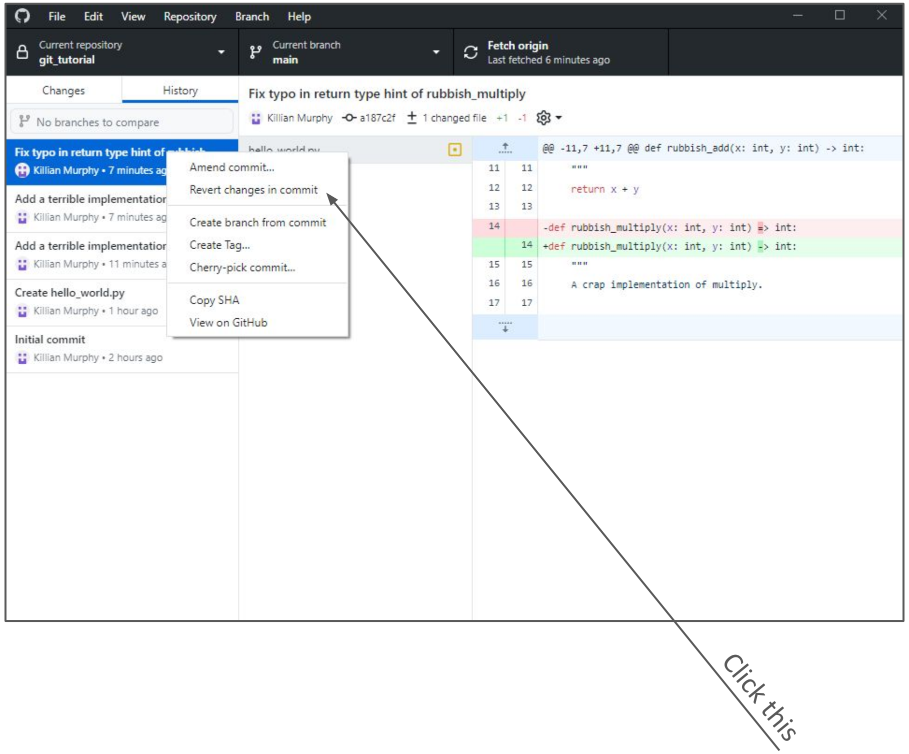
```
@@ -1 +1 @@
1   -GARBAGE
  1 +garbage
```



```
[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
```

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



Fix typo in return type hint of rubbish_multiply

Killian Murphy a187c2f 1 changed file +1 -1

Fix typo in return type hint of rubbish...
Killian Murphy • 7 minutes ag

Add a terrible implementation
Killian Murphy • 7 minutes ag

Add a terrible implementation
Killian Murphy • 11 minutes a

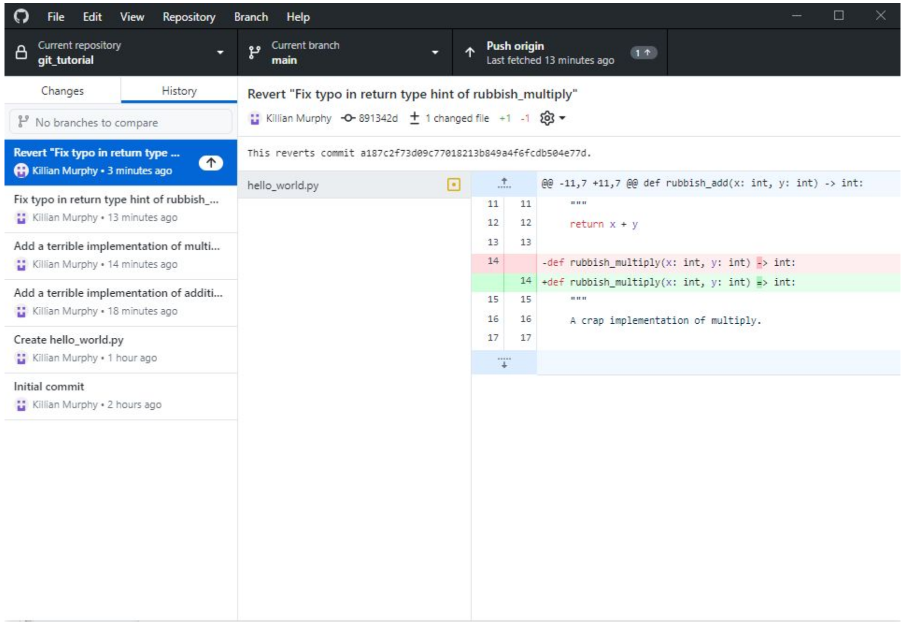
Create hello_world.py
Killian Murphy • 1 hour ago

Initial commit
Killian Murphy • 2 hours ago

```
@@ -11,7 +11,7 @@ def rubbish_add(x: int, y: int) -> int:
     """
     return x + y

-def rubbish_multiply(x: int, y: int) => int:
+def rubbish_multiply(x: int, y: int) -> int:
     """
     A crap implementation of multiply.

```

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



Revert "Fix typo in return type ... — Killian Murphy • 3 minutes ago

Fix typo in return type hint of rubbish_... — Killian Murphy • 13 minutes ago

Add a terrible implementation of multi... — Killian Murphy • 14 minutes ago

Add a terrible implementation of additi... — Killian Murphy • 18 minutes ago

Create hello_world.py — Killian Murphy • 1 hour ago

Initial commit — Killian Murphy • 2 hours ago

Revert "Fix typo in return type hint of rubbish_multiply"

Killian Murphy 891342d 1 changed file +1 -1

```
This reverts commit a187c2f73d09c77018213b849a4f6fcdb504e77d.
```

hello_world.py

```
@@ -11,7 +11,7 @@ def rubbish_add(x: int, y: int) -> int:
     """
     return x + y

-def rubbish_multiply(x: int, y: int) -> int:
+def rubbish_multiply(x: int, y: int) => int:
     """
     A crap implementation of multiply.

```

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 only 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

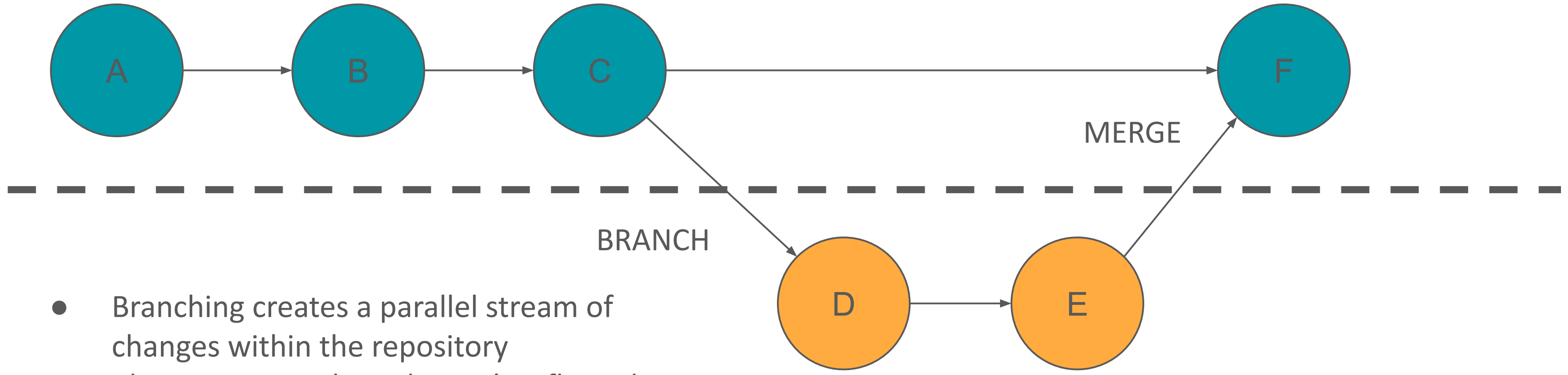
- If we are happy with the reversion, we can push the changes as with any other commit

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!

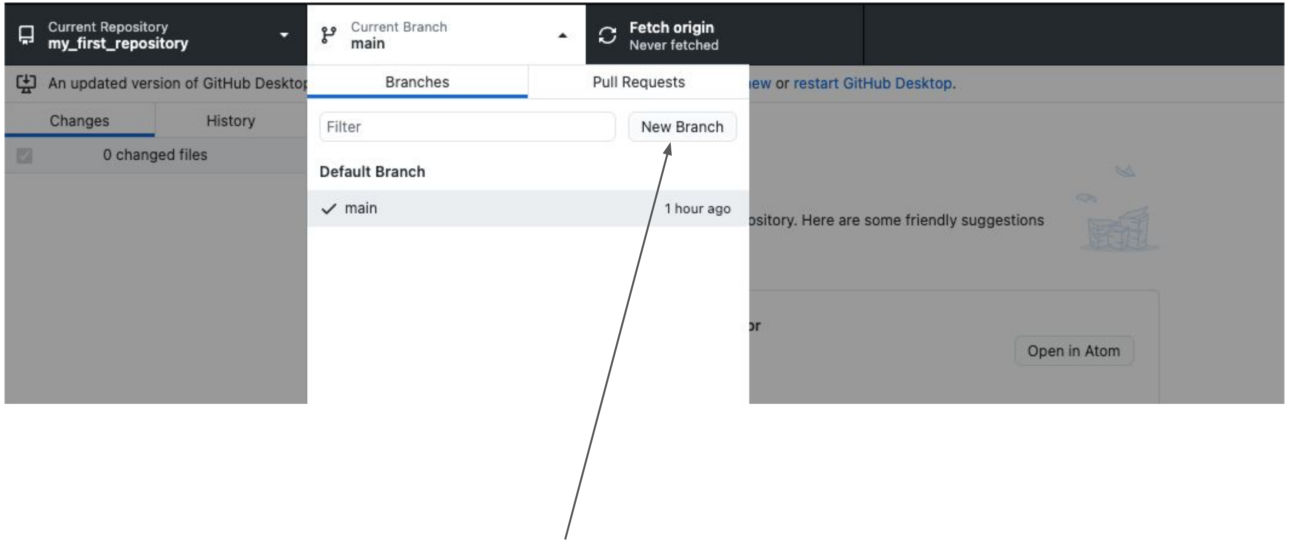
Branches - Overview

- 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**: to bring changes from one branch into 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
- **Fork**: make a copy linked to the original repo

Creating A Branch - GUI



Click this and call it "sandwiches"

Creating A Branch - CLI

```
git switch --create sandwiches
```

^ Create a new branch called "sandwiches" and immediately switch to it

Note: this is a newer wrapper around older commands, so you might see people online referring to `git branch` and `git checkout`. Those commands will still work, and you might have to use them if you have a git version older than 2.27

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 sandwiches`
- Now other people could see your branches when they clone your repository

Add a new file

sandwich.md

~~~
# A tasty sandwich

```
bread
bread
```


## Todos:
- [ ] add filling
~~~

- Let's add a new file: sandwich.md
- We'll fill it in later!
- Make sure you're on your new branch ("sandwiches" rather than "main")
- Add the new file, and commit it with a useful message

Working On Branches

- Switch back to your main branch. What do you notice?
  - CLI users: `git switch main` (note no `--create` flag!)
  - CLI users: try `git switch m<tab>`
- Have a look in your file explorer / finder / directory listing in your terminal in between switching branches

Working On Branches

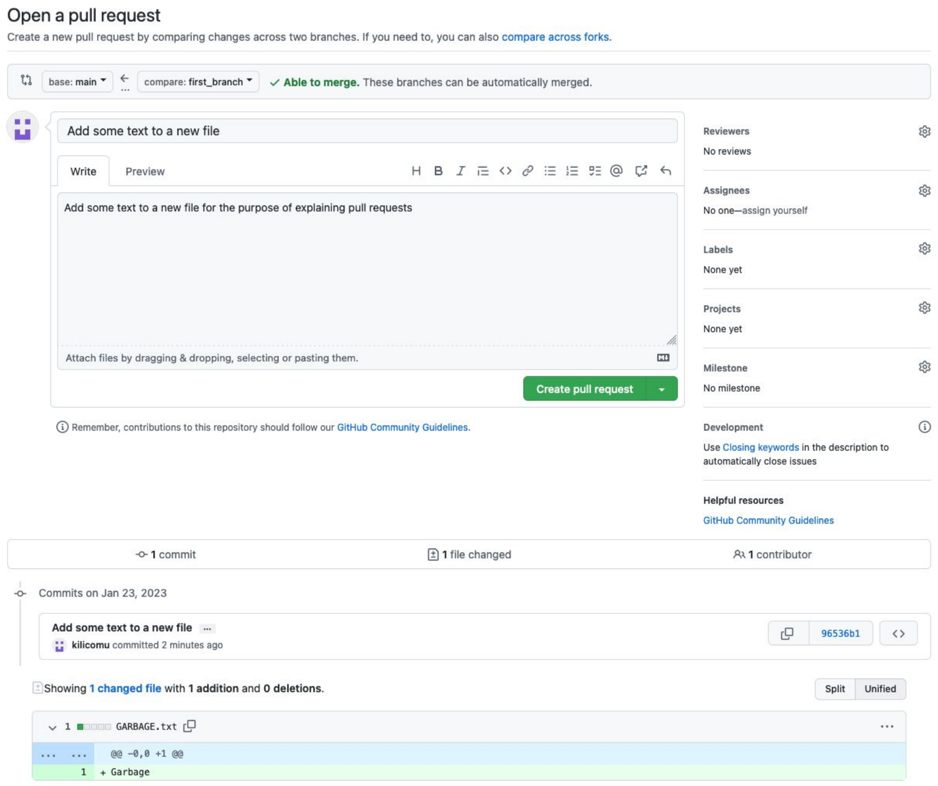
- Git is keeping track of changes to these branches separately
- Files that only exist on one branch will not be visible to you if you have switched away from that branch
- If we now want our main branch to reflect changes made on sandwiches, we need to **merge** sandwiches into main
- Before we do this, we can review the changes using a **Pull Request**
    - Slight misnomer from Github here, we're really requesting to **merge** branches. Oh well, there are two hard problems in computer science…

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 into which changes will be merged) and sandwiches 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


Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks.

base: main ← compare: first_branch ✓ Able to merge. These branches can be automatically merged.

Add some text to a new file

Add some text to a new file for the purpose of explaining pull requests

Attach files by dragging & dropping, selecting or pasting them.

Remember, contributions to this repository should follow our GitHub Community Guidelines.

Reviewers: No reviews
Assignees: No one—assign yourself
Labels: None yet
Projects: None yet
Milestone: No milestone
Development: Use Closing keywords in the description to automatically close issues
Helpful resources: GitHub Community Guidelines

1 commit · 1 file changed · 1 contributor

Commits on Jan 23, 2023

Add some text to a new file — kilicomu committed 2 minutes ago — 96536b1

Showing 1 changed file with 1 addition and 0 deletions.

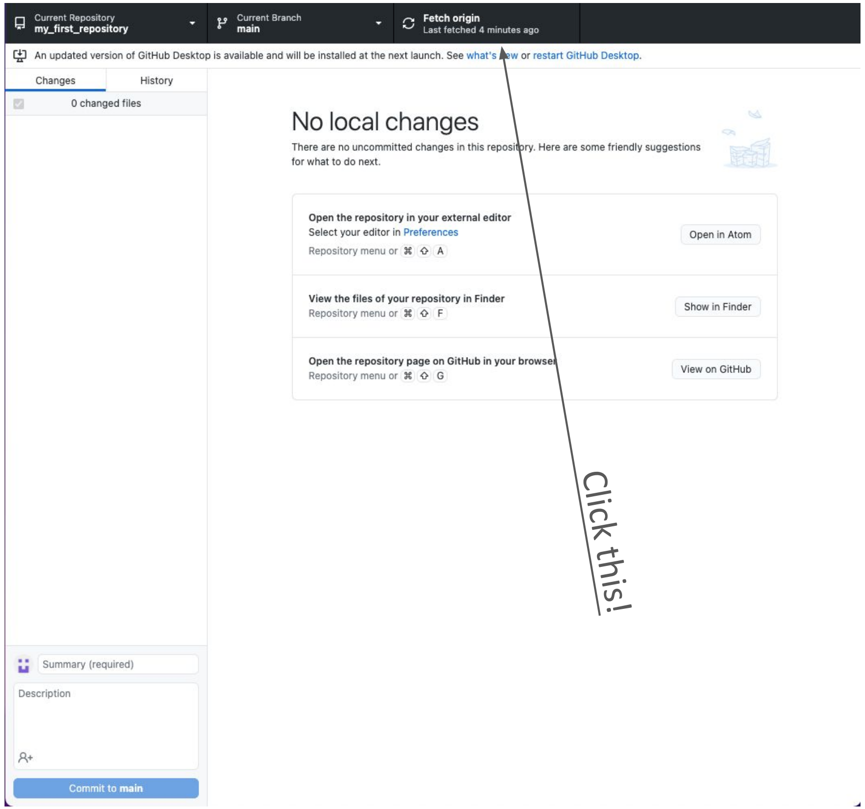
```
1 GARBAGE.txt
@@ -0,0 +1 @@
1 + Garbage
```

Merging A Pull Request

- Once you are done experimenting with your Pull Request, 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



Current Repository
my_first_repository

Current Branch
main

Fetch origin
Last fetched 4 minutes ago

An updated version of GitHub Desktop is available and will be installed at the next launch. See what's new or restart GitHub Desktop.

Changes | History

0 changed files

No local changes

There are no uncommitted changes in this repository. Here are some friendly suggestions for what to do next.

Open the repository in your external editor
Select your editor in Preferences
Repository menu or ⌘ ⇧ A
Open in Atom

View the files of your repository in Finder
Repository menu or ⌘ ⇧ F
Show in Finder

Open the repository page on GitHub in your browser
Repository menu or ⌘ ⇧ G
View on GitHub

Summary (required)

Description

Commit to **main**

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 sandwiches
```

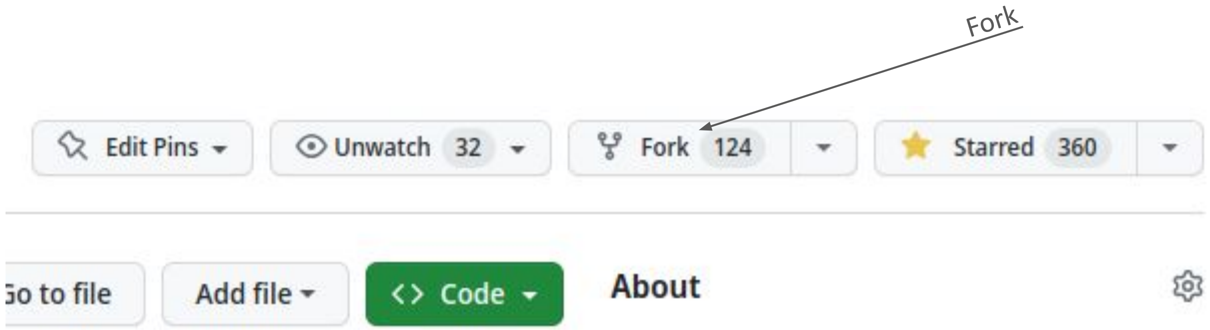
^ Delete the local copy of sandwiches, as we have finished with it

Collaborating With Others

- Git is distributed — every copy of a repo has all of its history, can make branches, and can pull branches from other copies
- We usually want to have one "official" repo that is the main version
- On Github we can add collaborators to our repos
  - Settings > Collaborators > Add People
  - But this has to be done by maintainers of the repo

Collaborating With Others

- What if we want to contribute to someone else's repo?
- **Fork**: make a copy linked to the original repo
- Github allows us to make PRs from forks into the original without needing to be added as a collaborator
  - Only approved people can actually merge them though!



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

Conflicting PRs

- Partner up with the person next to you
- Decide who is "person A" and who is "person 1"

Person A

- Fork Person 1's repo (you might need to rename it!)
- Clone it locally (**Code** to get the URL)
- Make a new branch person-A
- Add a filling to sandwich.md and check the box ([ ] → [x])
- Commit and push to your fork of the repo
- Open a PR of your person-A branch into Person 1's main branch

Person 1

- Make a new branch person-1
- Add a different filling to sandwich.md and check the box ([ ] → [x])
- Commit and push to your repo
- Open a PR of your person-1 branch into your main branch

Resolving 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 this
- 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
- 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
- Notice how the checkbox didn't cause a conflict, even though you both changed it?
- If you merge on the command line, git puts these conflict markers straight into the file and expects you to fix them before you merge

~~~
# A tasty sandwich

```
bread
<<<<<<< person-1
oh look, a conflict
=======
hummus
>>>>>>> main
bread
```

## Todos:

- [x] add filling
~~~

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
- https://git-scm.com