

# Source Control with Git

Generation

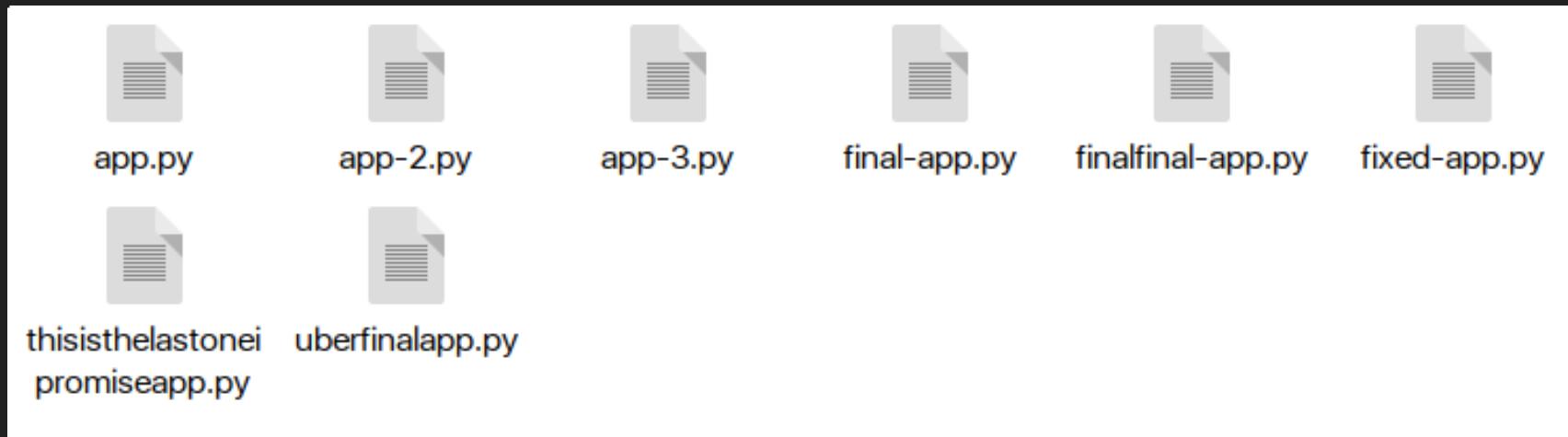
# Overview

- Source Control
- Git
- Sharing Code with git
- Branching
- Pull Requests

# Learning Objectives

- Describe what source control is, and why we use it
- Add and commit features, and push to remote
- Create branches
- Create, review, and merge pull requests
- Create a Git repository

# A World Without Version Control



- If we wanted to keep historical versions of a file, we would need to manually create a file for each version with a new name.
- Keeping track of this would become cumbersome.

# What is Source Control?

- Source control allows you to keep track of your files and every change made to them, ever.
- It also allows you to share these files and their change history with others, so they can make changes too.

# Git

- A distributed source control management system (SCM)
- It has become the most widely used tool to manage source code nowadays

# Exercise (part 1/2)

Check you already have it installed:

```
$ git --version  
git version 2.24.3 (Apple Git-128)
```

Install Git using your package manager if you haven't:

```
$ sudo apt-get install git    # Unix  
$ brew install git           # macOS – you need Homebrew first
```

Windows: browse to <https://gitforwindows.org/>

## Exercise (part 2/2)

Configure Git on your machine. Run each of these one line at a time.  
Use your own name and email:

```
git config --global user.name "Alice Bloggs"  
git config --global user.email "alice.bloggs@someserver.com"  
git config --global init.defaultBranch main  
git config --global core.pager cat  
git config --global core.editor "nano"  
git config --global pull.rebase false  
git config --global fetch.prune true
```

(Windows users should do this in Powershell and all again in GitBash.)

# Checking we can reach GitHub on port 22

In a Bash or GitBash window run this:

```
ssh -T git@github.com
```

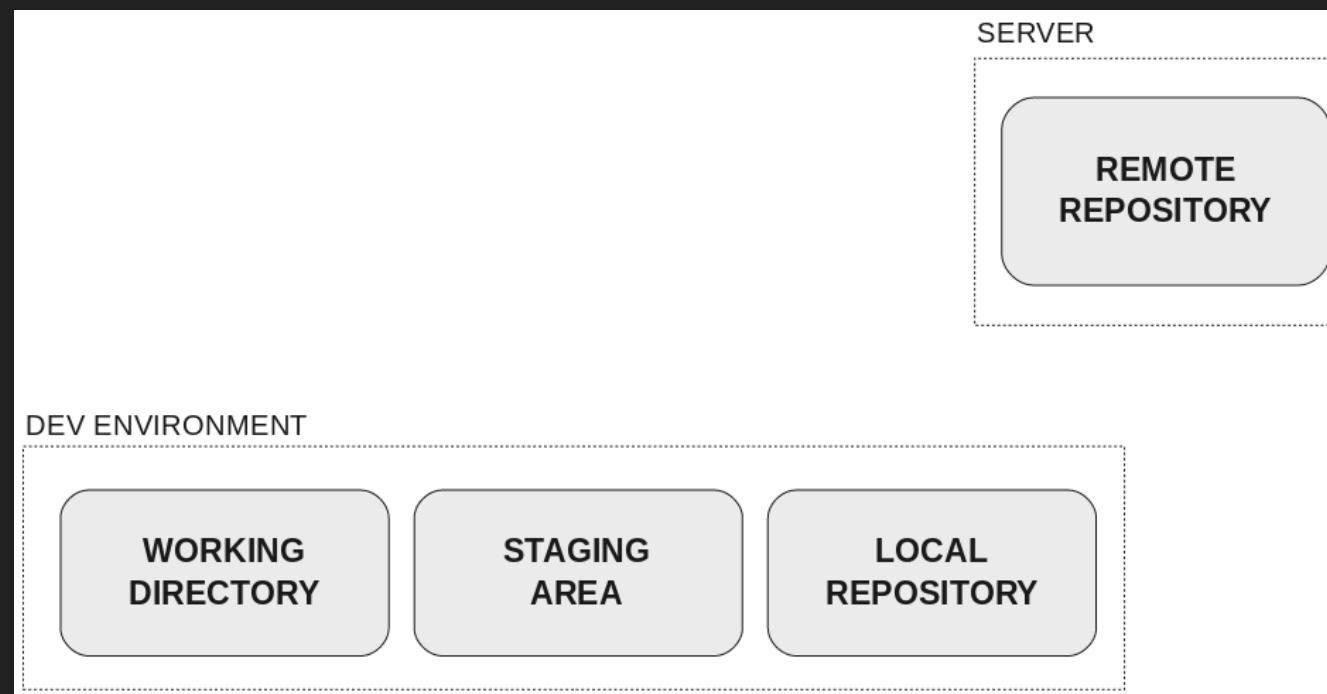
You should see something like this, which indicates success:

```
ross.cullen@c11-b8svfsiu8e1 MINGW64 /c/code/ross-c-de-sot-sept-2024 (main)
② $ ssh -T git@github.com
Hi ross-cullen! You've successfully authenticated, but GitHub does not provide shell access.

ross.cullen@c11-b8svfsiu8e1 MINGW64 /c/code/ross-c-de-sot-sept-2024 (main)
```

If you don't, you may ned to check your home router or the office WiFi.

# Key Git components

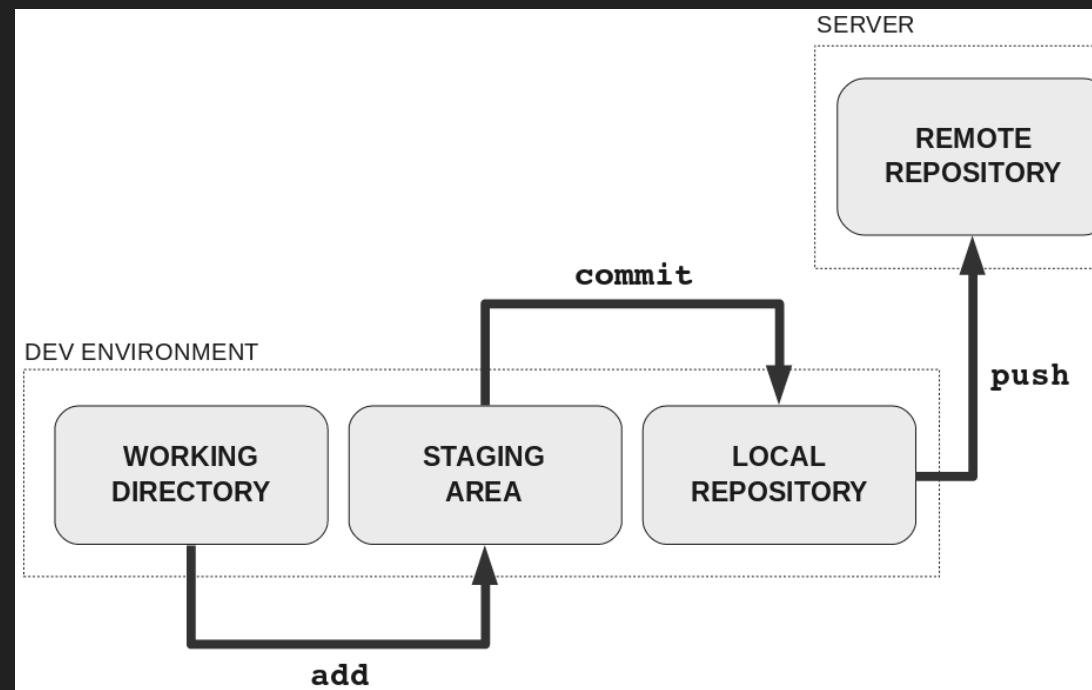


Fear not, this diagram will be explained on the next slide.

# Key Git components

- **Working directory:** your private workspace, where changes are made. It may contain tracked and untracked files.
- **Staging area:** all changes you want to bundle up in the next commit (transaction) to your local repository.
- **Local repository:** your local version of the Git repository.

# Making changes in Git



# Making changes in Git

- You can think of the steps you take to send your local changes to the remote repository as a bit like what you do when you send a letter:
- First, you take your letter (i.e. your local changes) and put it in an envelope. This like **staging** your local changes.
- Then you double check that the letter is really ready to be sent, and once you're satisfied, you seal the envelope, add the address, and put on a stamp. This is like **committing** your staged changes to your local repository.
- Once all of this is done, and you're really satisfied and ready to share your letter, you can take it to the postbox and send it. This is like **pushing** your committed changes from your local repository to the remote.

# Making changes in Git



# git add

Mark files in your working directory as tracked so Git starts keeping a track record of all changes to them.

If a file is already tracked and has some changes, move it to the staging area to include it in the next commit.

```
# Stage one file  
git add myfile.py  
# Stage everything in the current directory (and subdirectories)  
git add .
```

# git commit

Bundle up all staged changes (or new files) in a new commit to apply to the local repository.

```
# Write commit message in text editor  
git commit  
# Or inline commit message  
git commit -m "Update function definition"
```

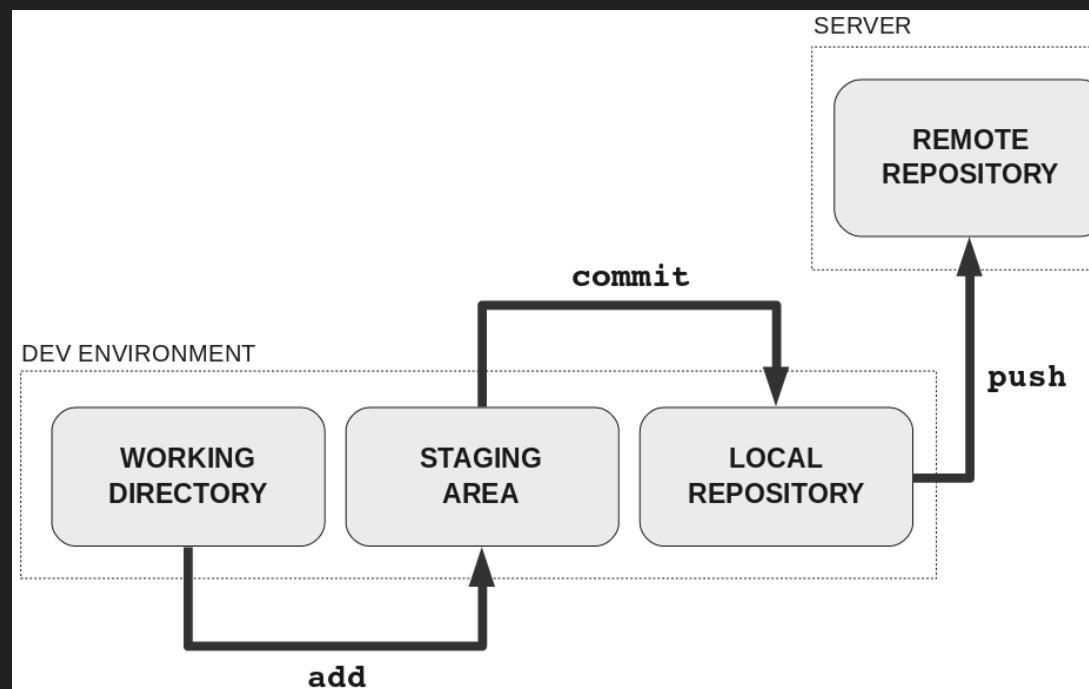
# git push

Upload any changes made in your local repository to the remote repository:

```
git push
```

# git add, commit and push

Let's revisit that sequence:



## Emoji Check:

Do you feel you understand the git sequence? Say so if not!

1. 😢 Haven't a clue, please help!
2. 😕 I'm starting to get it but need to go over some of it please
3. 😌 Ok. With a bit of help and practice, yes
4. 😊 Yes, with team collaboration could try it
5. 😃 Yes, enough to start working on it collaboratively

# When should I commit?

Commits should:

- Pass all tests, and not have any known bugs to be fixed in a future commit
- Small self-contained changes that work
- Have short and meaningful commit messages
- Act as a save point for your code changes

Example commit message: change function to return integer value rather than string value

# One important message about commits

**NEVER** commit sensitive information! This includes:

- Passwords
- Keys
- Private information about a person or entity

As professional software engineers, we can use handy tools like [git-secrets](#) to catch when we might accidentally commit sensitive information.

Check out the [academy-git-secrets-example](#) repo for a demonstration of how this tool can be used in your projects.

# When should I push?

"Frequently"

- Code that isn't pushed exists only on your machine. So while it is a "save point" per commit, it's not backed up, and your team can't see it or use it. And if your machine crashes, it's still lost!
- Some teams like to push with every commit.
- At least - do it when you leave your machine, like lunchtime and at the end of the day.

# .gitignore

- We can use a special file called a `.gitignore` to tell git not to track certain files
- This is a good way to protect sensitive information
- It can also come in handy for lots of other things, e.g. local config files, or very large package-related directories

# Demo: .gitignore

The instructor can show you how this works

Quiz Time! 😎

**Which git command would you use to apply your locally staged changes to your local repository?**

1. `git status`
2. `git add`
3. `git commit`
4. `git push`

**Which git command would you use to apply your locally staged changes to your local repository?**

1. `git status`
2. `git add`
3. `git commit`
4. `git push`

Answer: 3

## Emoji Check:

Do you feel you understand git better now? Say so if not!

1. 😢 Haven't a clue, please help!
2. 😕 I'm starting to get it but need to go over some of it please
3. 😌 Ok. With a bit of help and practice, yes
4. 😊 Yes, with team collaboration could try it
5. 😃 Yes, enough to start working on it collaboratively

# Sharing Code

# Sharing Git repositories

- Git is distributed, which means most of the changes happen in people's individual repositories (repos) rather than in a central location.
- Everyone's repos have all information about the repo.
- There is no 'one and only' source of repo information.

# Sharing Git repositories

- Because Git is distributed, all information about a repository is copied every time it is cloned.
- You upload your local changes to remote repositories hosted on private or public servers, and also download changes made by other people into yours.

# Sharing Git repositories

- Git repositories can be hosted in a private Git server, although this is something few people do.
- Most people prefer to utilise code hosting services like BitBucket, SourceForge, GitLab or GitHub.

# GitHub

- GitHub is the most popular code hosting website.
- GitHub is free and offers loads of tools to aid coders:
  - Code hosting
  - Code discovery and search
  - Bug reports
  - Project task boards
  - Project wikis

# Exercise - Set up a GitHub account

- [Create a GitHub Account](#)
- Set up an SSH key for GitHub
  - Follow the [Generating a new SSH key instructions](#)
  - Then follow the [Adding a new SSH key to your GitHub account](#)
  - Then follow the [Testing the SSH connection](#)
  - Then setup [Multi-Factor Authentication \(MFA\)](#) on your Github Account

# Exercise - Join a GitHub organisation

- GitHub organisations are shared accounts where groups or businesses can collaborate across many projects at once.
- Share your GitHub username with your instructor. They will invite you to an organisation for your cohort.
- Once you are invited, you will receive an email invitation.
- You can also accept the invitation at  
<https://github.com/orgs/{organisation-name}/invitation>  
(you will need to update the link to use the organisation for your cohort).

# Exercise - Clone a repository

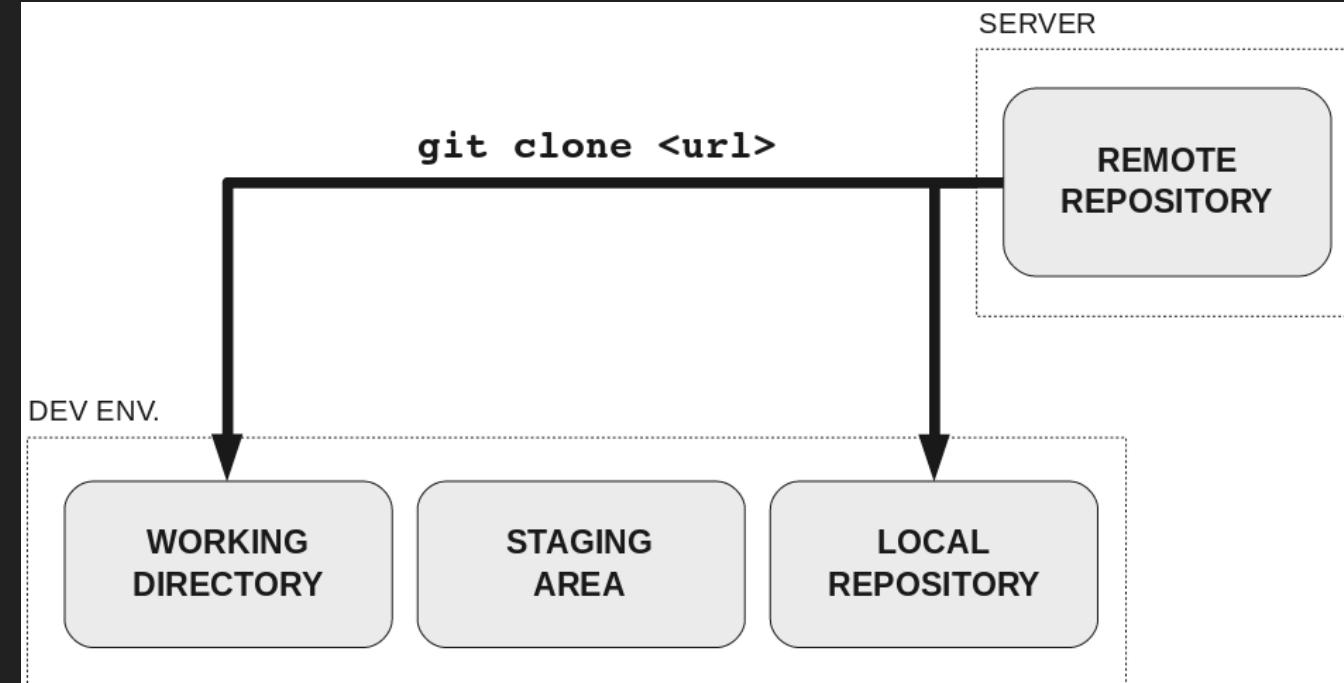
- Follow the instructor!
- Navigate to your programme directory (wherever you are storing your files for this programme) and clone the repository (the instructor will give you the correct repository link):

```
git clone <repo-ssh-link>
```

- This should create a new directory that you can `cd` into

# Git clone - what did we do?

This "copies" the remote information into your machine:



# Lets make a change

- In this repo make a new text file and call it your name (e.g. linzic.txt) and write your name in there
- Once you are happy let's get this committed and pushed, who can remember the commands we need to do this?

# Push those changes!

- git add <my-name.txt>
- git commit -m "Added a new file called my name"
- git push

# What's changed?

- Once you have pushed your changes, have a look at the remote repo and see if you can find the file you just pushed
- <https://github.com/{organisation-name}/{repo-name}>
- Hopefully you should see your file up there (along with some others...?)

## Emoji Check:

Do you feel you understand how to push changes now? Say so if not!

1. 😢 Haven't a clue, please help!
2. 😕 I'm starting to get it but need to go over some of it please
3. 😌 Ok. With a bit of help and practice, yes
4. 😊 Yes, with team collaboration could try it
5. 😃 Yes, enough to start working on it collaboratively

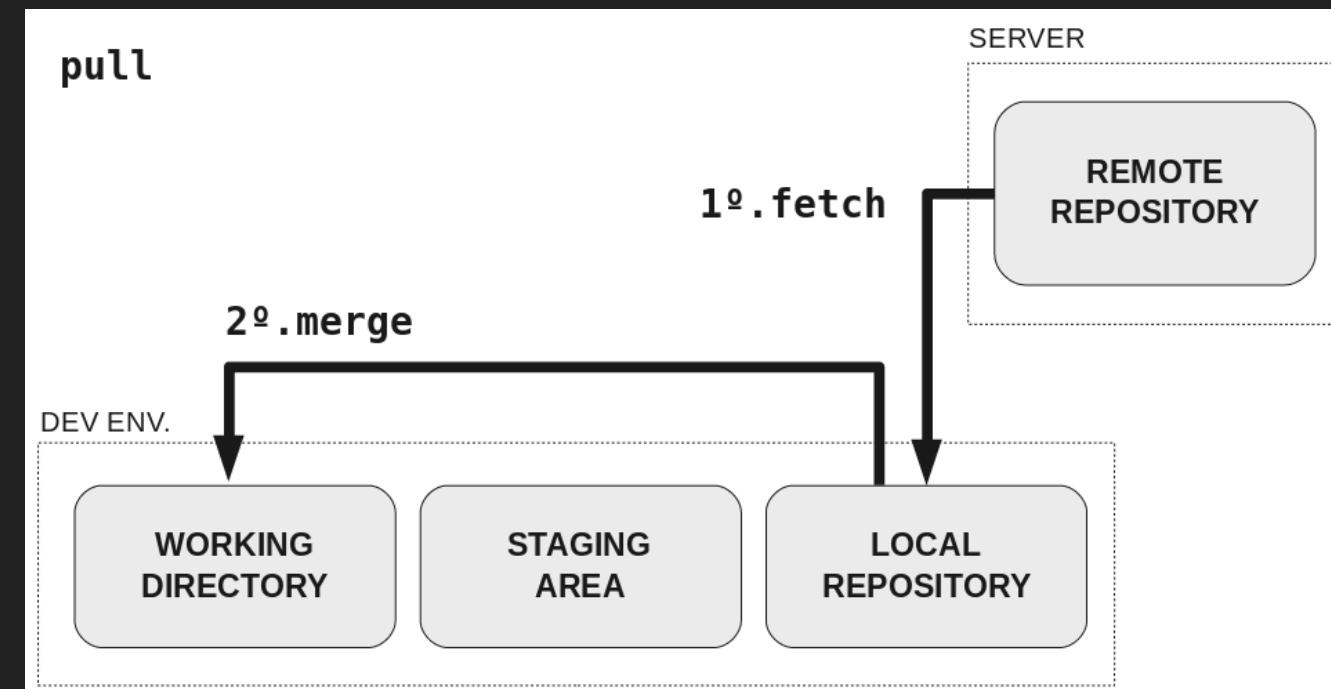
# Local updates

What if other folks have also sent work to the remote sever, and you don't have those yet?

- I can't see those new files on my machine
- This is because the remote has been updated, but your local repo is still behind

# Local Updates - Pull

- To update the files locally you can use the `git pull` command to apply any changes to your local files that other people have made
  - This only updates your current branch - more on this later



# Conflicts

- Lets try and edit a file that everyone has access to
- In the repository you should see a file called `your-hobbies.md`
- Now write something about yourself in the designated section and then `git add`, `git commit` and `git push` that code

# Conflicts

- Once you have done this, try and pull down the changes.  
There's a chance you see the following message:

```
→ NJA-source-control-2021 git:(main) git pull
Auto-merging about-me.txt
CONFLICT (content): Merge conflict in about-me.txt
Automatic merge failed; fix conflicts and then commit the result.
→ NJA-source-control-2021 git:(main) x █
```

- This is known as a conflict, where git was unable to merge changes because two separate sources have changed the same line

# Conflicts

- If you open up vscode again, if you navigate to the offending file then you will see something funky like this:

Add your name and a fun fact about yourself between the lines:

-----

Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes | Start Live Share Sessi-

<<<<< HEAD (Current Change)

I sure hope this doesn't conflict

=====

Tom C - I like Leeds United. They don't like me, they made me sad.

>>>>> 49152ffdbf2e2e20b0f6d275da77498032ede6f5 (Incoming Change)

-----

- This will let you know what your change is (current change) and what the conflicting change in the repo is (incoming change)

# Fixing mistakes

- The simplest way to fix a mistake is to change them back in your local directory, and then make a new commit
- There are other ways to achieve this, but they involve potentially re-writing git history and need to be treated with caution
- For now, let's just delete our changes and make new commits

# Viewing our previous changes

You can use `git log` to take a look at the commit history for a repository.

```
commit dffadb72336f40887589a76de53bf10f8b3b70b3 (HEAD -> first-branch, origin/first-branch)
Author: Rowan <rowan.gill@infinityworks.com>
Date: Mon Sep 13 16:58:46 2021 +0100

    first-branch commit

commit 0460a1026b9cc8592eaa55fce99b8ef1b50a06ec (origin/main, origin/HEAD, main)
Author: Rowan <rowan.gill@infinityworks.com>
Date: Mon Sep 13 15:03:35 2021 +0100

    removing any text

commit ae606e0c0b439bd2b2c9fa22715987f425b42381
Merge: b4df622 49152ff
Author: Rowan <rowan.gill@infinityworks.com>
Date: Mon Sep 13 15:03:13 2021 +0100

    Merge branch 'main' of github.com:infinityworks/NJA-source-control-2021

commit b4df62233c3a8cd0342a750caa95ad7075bc548
Author: Rowan <rowan.gill@infinityworks.com>
Date: Mon Sep 13 15:03:09 2021 +0100

    fixing example

commit 49152ffdbf2e2e20b0f6d275da77498032ede6f5
Author: Tom Carabine <12697250+tcarabine@users.noreply.github.com>
Date: Mon Sep 13 14:50:10 2021 +0100

    added information about Tom

commit 55fde2734bf6c580840af6e61529c1ae2fb3115e
Author: Rowan <rowan.gill@infinityworks.com>
Date: Mon Sep 13 14:49:20 2021 +0100

    about me, sure hope it doesn't conflict
```

## Emoji Check:

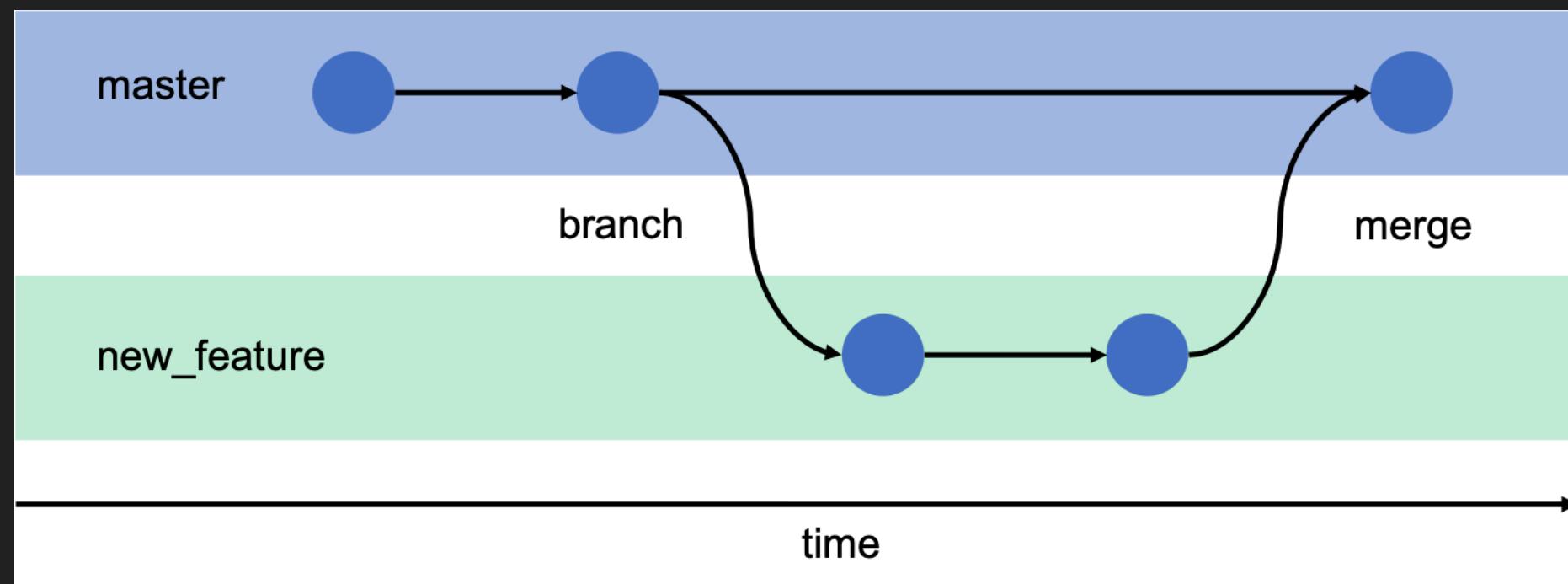
Do you feel you understand conflicts? Say so if not!

1. 😰 Haven't a clue, please help!
2. 😕 I'm starting to get it but need to go over some of it please
3. 😃 Ok. With a bit of help and practice, yes
4. 😊 Yes, with team collaboration could try it
5. 😄 Yes, enough to start working on it collaboratively

# Branches

- A branch is an alternative development path in your repository
- Branches are usually created from `main`, but they can branch from anywhere (like a tree)
- Branches allow us to commit our work-in-progress, without affecting the original branch

# What is a branch?



# Creating and checking out branches

```
git branch <new_branch>
git checkout <new_branch>
```

```
# You can create and switch to a branch in one go
git checkout -b <new_branch>
```

# Pushing branches

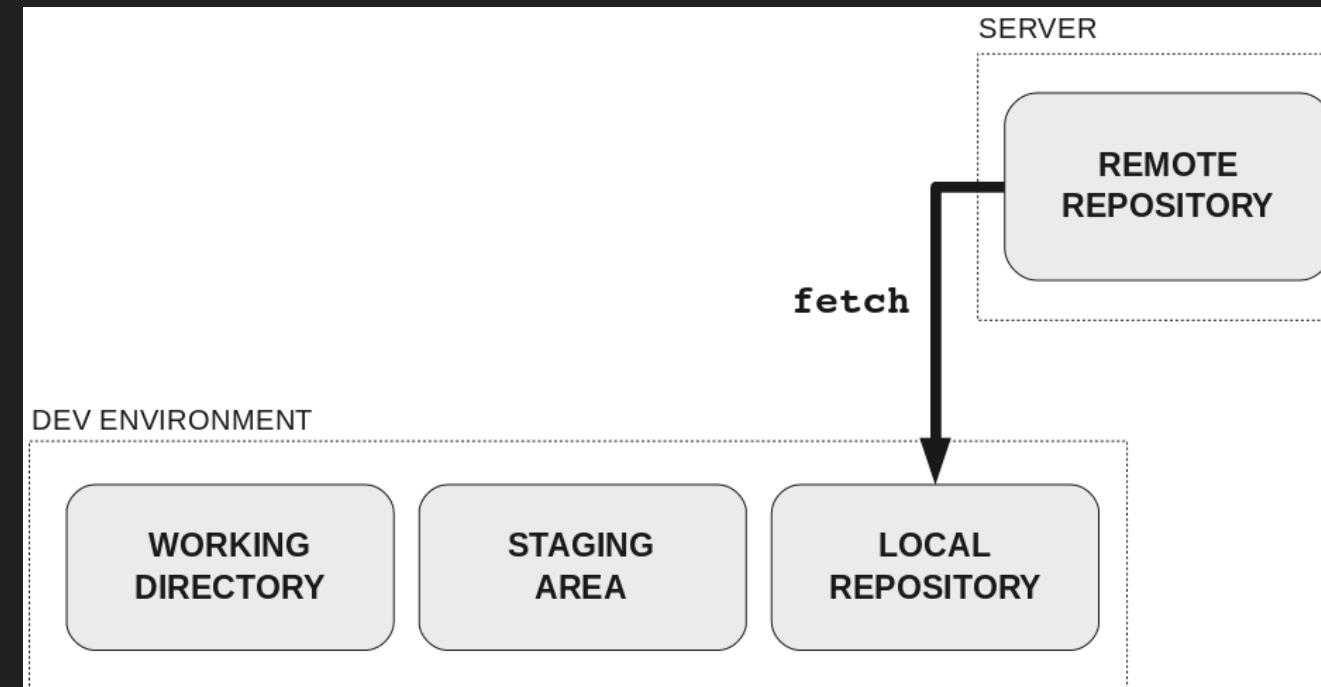
--set-upstream creates an **upstream** branch in the remote that matches our local branch of the same name.

```
git push --set-upstream origin <new_branch>
```

(This is not necessary if pushing to **main**, as that's the default)

# Local Updates - Fetch

- To get your local machine up to date with all the remote branch names, we use `git fetch`
- There is an extra `--prune` option to make sure our list of branches has old ones removed
- Because of the global setting we used previously, the `--prune` is now your default



# Scenario

Alice has written some code on a feature branch and pushed her branch up to the remote. You want to have a look at her code and try it out for yourself. How can you do this?

1. `git branch alices-branch`
2. `git fetch && git branch alices-branch`
3. `git checkout alices-branch`
4. `git fetch && git checkout alices-branch`

# Scenario

Alice has written some code on a feature branch and pushed her branch up to the remote. You want to have a look at her code and try it out for yourself. How can you do this?

1. `git branch alices-branch`
2. `git fetch && git branch alices-branch`
3. `git checkout alices-branch`
4. `git fetch && git checkout alices-branch`

Answer: 4

# Exercise - Our first branch

- So we've seen how to make and checkout a branch
- Create a new unique branch with your name (eg linzi-first-branch) and change a line in the file you made earlier  
`yourname.txt`
- When you are happy commit and push that branch
- Open GitHub in the browser and find it

## Emoji Check:

Do you feel you understand branches? Say so if not!

1. 😢 Haven't a clue, please help!
2. 😕 I'm starting to get it but need to go over some of it please
3. 😌 Ok. With a bit of help and practice, yes
4. 😊 Yes, with team collaboration could try it
5. 😃 Yes, enough to start working on it collaboratively

# What now?

So we have a branch pushed up to the remote repository. We want to merge those changes to our `main` branch. How do we do this?

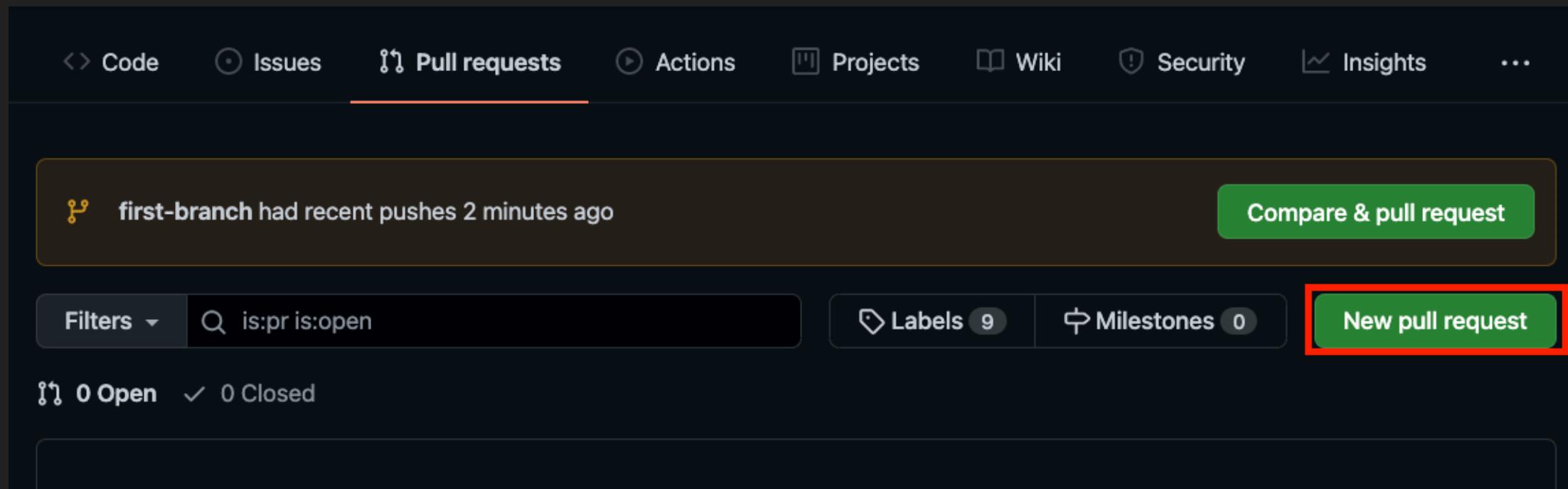
- There are a few ways to achieve this
- One of the most common is to raise a pull request

# Pull Requests

- Pull requests allow us to control changes going into our **main** branch
- They can be made to need passing test suites before they can be merged, creating a safety net for your code
- Provides an opportunity for review pairing and knowledge share

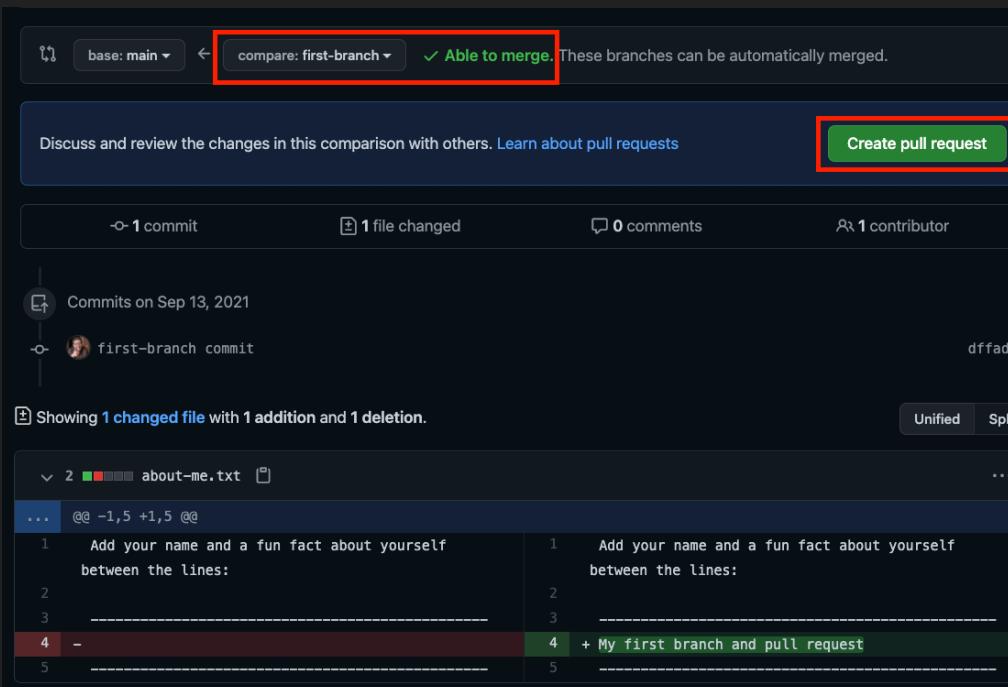
# Raising a Pull Request

- To raise a pull request head to the github page for the repository and select the pull requests tab



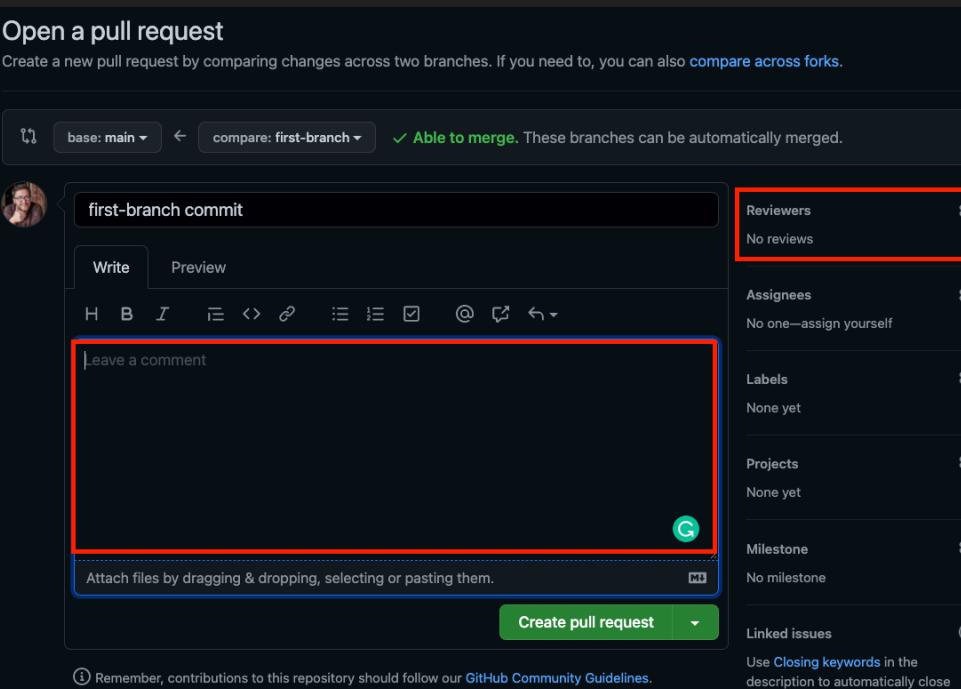
# Setting up a PR

- There's some important information you need to provide to create a pull request, such as specifying the branch you want to merge



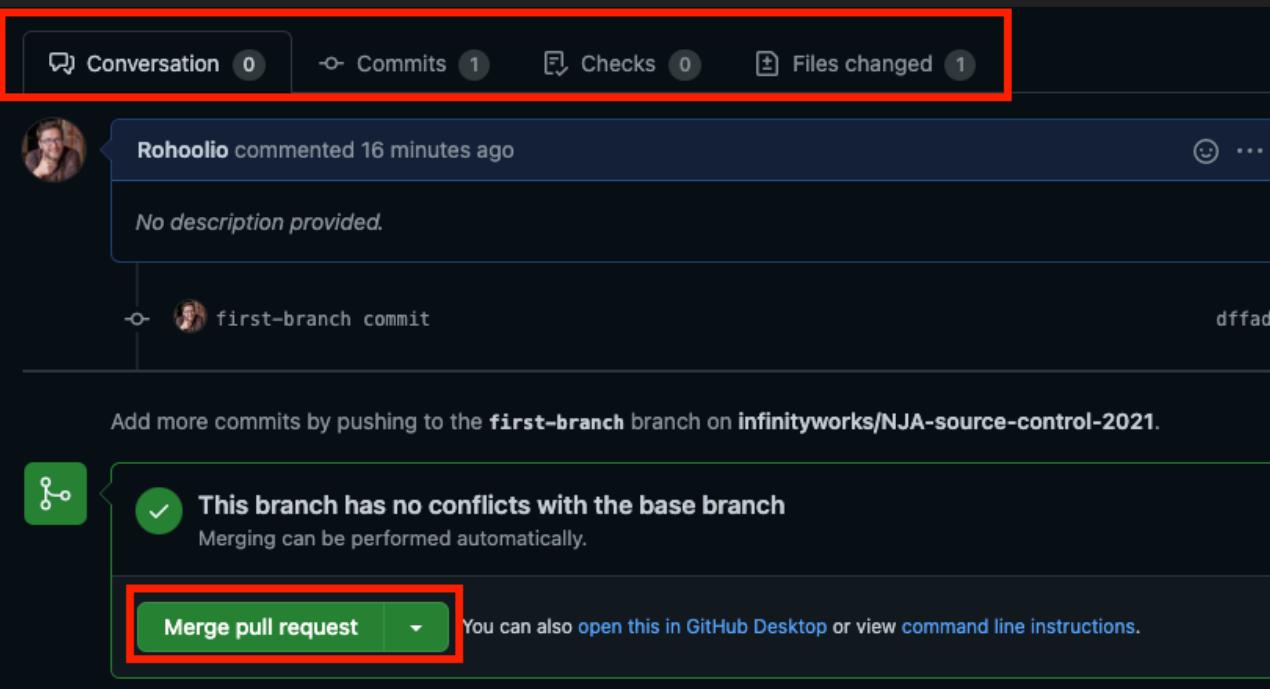
# Setting up a PR

- As well as some useful additional information like the description and adding reviewers (these can still be changed after the pull request is made)



# Complete a PR

- When all the previous steps are done others can then view, comment and suggest changes to the code



Whomever made the PR can't also Approve it - why not?

## Emoji Check:

Do you feel you understand pull requests? Say so if not!

1. 😢 Haven't a clue, please help!
2. 😕 I'm starting to get it but need to go over some of it please
3. 😌 Ok. With a bit of help and practice, yes
4. 😊 Yes, with team collaboration could try it
5. 😃 Yes, enough to start working on it collaboratively

# GitHub organisation

We have a shared GitHub org.

This is where you will add your own mini project repositories, and later, ones for your group projects.

Let's take a look at it now...

- <https://github.com/{organisation-name}>

# Your own GitHub repository

Over the next few days, you will be getting started on a daily git kata.

Your first tasks will be to create a remote repository for your mini-project, and push your project files to it.

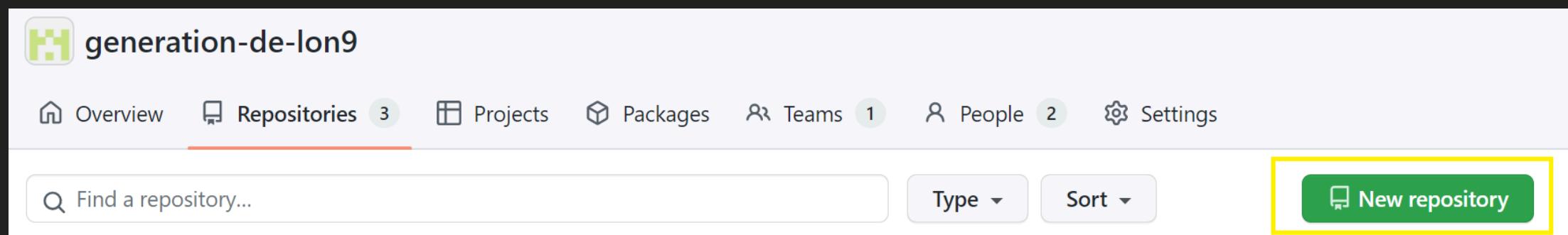
**You will be starting on this tomorrow morning, so watch closely!**

# Demo - New repository

- Create a new private [GitHub repository](#) for your mini-project
- Make sure to select our organisation as the **Owner**
- Set the **Repository name, including your own name**
  - e.g. **sue-smith-mini-project**
- Add a description
- Set it to private
- Add a default README file
- Add a default Python .gitignore file
- Clone this repository to your local machine

Images on next slides!

# Create a new repo



# Create a new repo

Select owner and add a repo name

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Repository template

Start your repository with a template repository's contents.

No template ▾

Owner \*  generation-de-long ▾

Repository name \*  your-name-mini-project ✓

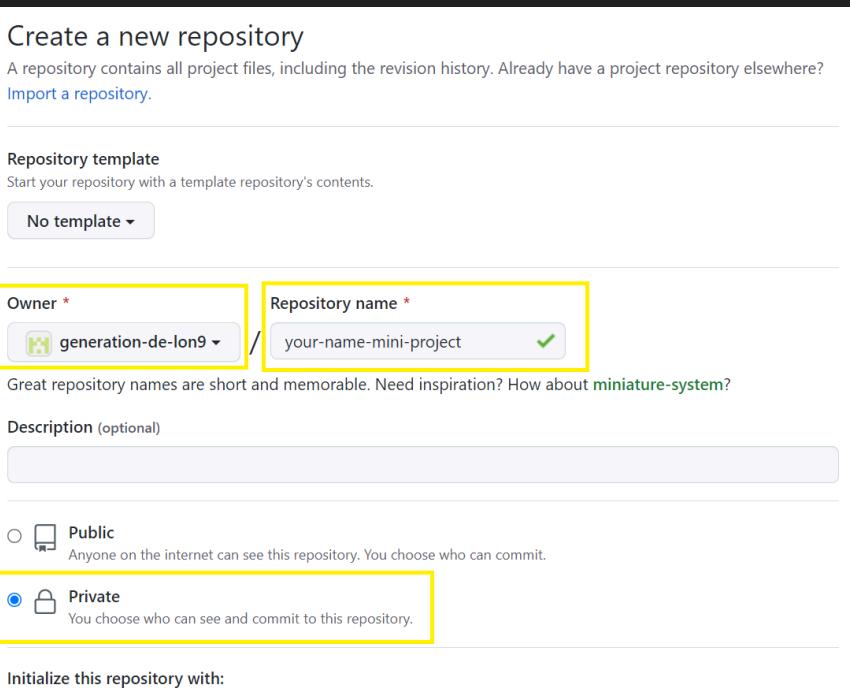
Great repository names are short and memorable. Need inspiration? How about [miniature-system](#)?

Description (optional)

Public Anyone on the internet can see this repository. You choose who can commit.

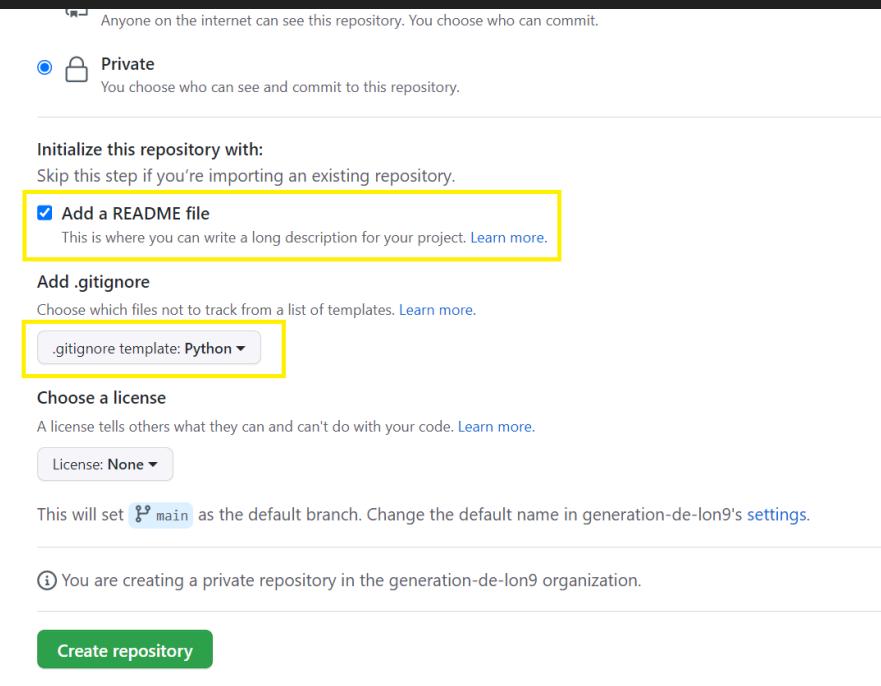
Private You choose who can see and commit to this repository.

Initialize this repository with:



# Create a new repo

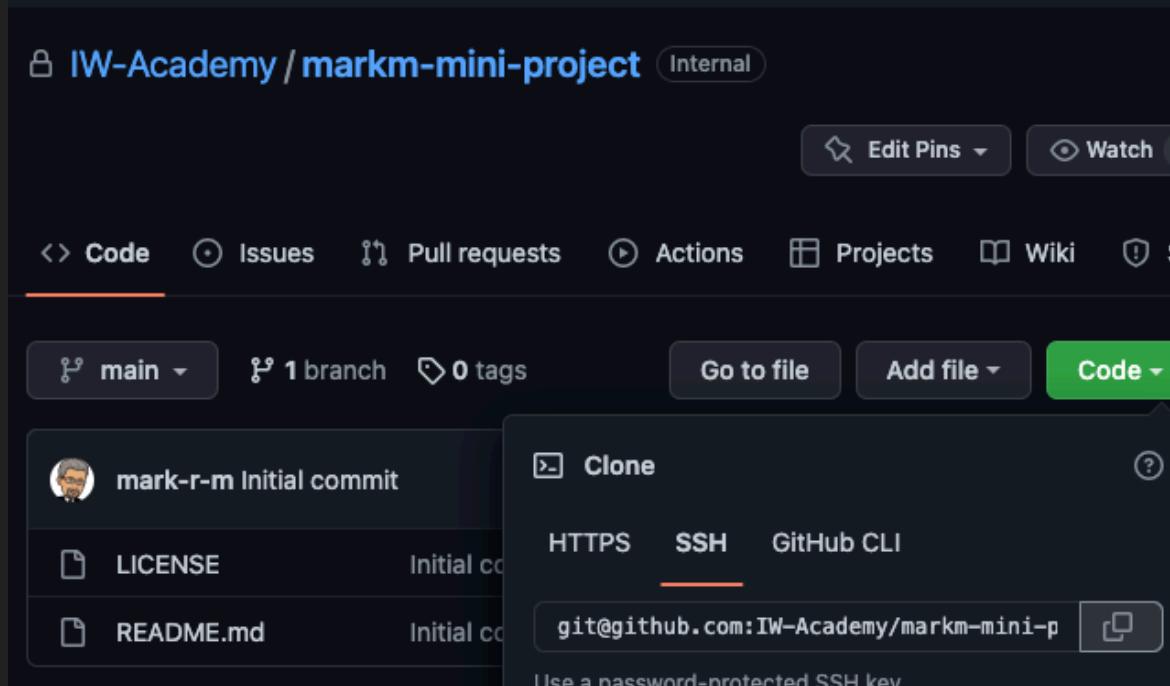
## Update repo settings



# Create a new repo

Clone the repo using the SSH details

When on the **Code** tab on your repo, select the green **Code** button and pick the SSH option to get the URL to clone



```
git clone <ssh-link-from-green-code-button>
```

# Exercise - new repository

- Create a new private [GitHub repository](#) for your mini-project
- Make sure to select our organisation as the **Owner**
- Set the **Repository name, including your own name**
  - e.g. **sue-smith-mini-project**
- Add a description
- Set it to private
- Add a default README file
- Add a default Python .gitignore file
- Clone this repository to your local machine

# Exercise - Fill your GitHub repository

- Move your mini project files into the newly cloned repository

We suggest a folder structure like this:

```
.gitignore  
README.md  
src/*  
tests/*  
requirements.txt
```

This way you can put all of your app code into `src/` and unit tests (more on that in a future session) into `tests/`.

You'll have a `data/` directory too in your local repo containing any persisted data for your app. Add `data/*` to your `.gitignore` file so that personal data doesn't get pushed to the remote repo.

# Exercise - Push to your GitHub repository

- In a terminal:
  - Add all your files to staging
  - Commit all your files with a message
  - Push your files
- Then in your web browser:
  - Check you can see your files in your repository in GitHub

# Introducing the daily kata exercises

Because git is really really important, you will be practicing the commands we have covered today (as well as a few new ones) in a daily kata, starting tomorrow.

An instructor will run you through the exercise at the end of the day, and then you will practice it in the morning.

# Terms and Definitions - recap

- **Repository:** Stores all your code, and all history and tracking metadata
- **Commit:** A number of changes bundled up as a single transaction
- **Branch:** Alternative code path originating from a specific commit in another branch
- **Staging:** Telling Git what changes we want to include in the next commit

# Terms and Definitions - recap

- **Clone:** Copy a remote Git repository and all of its metadata into our local environment
- **Fork (GitHub):** Make a copy of a repo hosted online into another hosted repo under our account so we can freely change it
- **Pull Request / Merge Request:** Request to merge a branch/fork into the main branch/fork it originated from (upstream)

# Overview - recap

- Source Control
- Git
- Sharing Code with git
- Branching
- Pull Requests

# Objectives - recap

- Describe what source control is, and why we use it
- Add and commit features, and push to remote
- Create branches
- Create, review, and merge pull requests
- Create a Git repository

# Further Reading

- [Roadmap.sh - Learn Git and GitHub](#)
- [Pro Git Book \(free\)](#)
- [git tutorials](#)
- [Pull requests tutorial](#)
- Credits to [Rachel Carmena](#)
- [Great introductory article to git](#)
- `man git`

## Emoji Check:

On a high level, do you think you understand the main concepts of this session? Say so if not!

1. 😢 Haven't a clue, please help!
2. 😟 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 😊 Yes, with team collaboration could try it
5. 😃 Yes, enough to start working on it collaboratively