

Getting Started with Git & GitHub

Garrett Speed & Aristotelis Kandylas

An introductory workshop for researchers to begin tracking their code development with Git through GitHub Desktop and publishing on GitHub.com



Geo Data Team - Utrecht University

Acknowledgements

Thank you to Dorien Huijser and Jelle Treep for reviewing the drafts of this manual.

This manual is licensed as CC-BY-SA, please share and adapt for your uses.



This manual was developed spring 2023 and finalized in June 2023. This workshop was developed by the Geo Data Team of the Faculty of Geosciences at Utrecht University. For more information about the team, workshops, and services, please visit their [website](#).

Cover art by Garrett Speed in Blender with 3D models from Thingiverse:

- Foreground [octopus](#) by user boris3dstudio (CC-BY)
- Front background octopus and back cover [Octopus](#) by user glad3Dprintable (CC-BY-SA)
- [Python snakes](#) by user Jakim (CC-BY)
- [Matlab logo](#) by user thaabomb (CC-BY-SA)

Table of Contents

Acknowledgements	2
Table of Contents	4
Introduction	6
<i>Introductory Concepts</i>	<i>6</i>
<i>Conventions Used</i>	<i>8</i>
<i>Incomplete List of Text-Based Files</i>	<i>8</i>
<i>On GitHub.com</i>	<i>9</i>
Creating Git Repositories	9
<i>In GitHub Desktop</i>	<i>15</i>
<i>Markdown</i>	<i>18</i>
Cloning a GitHub Repository to Your Local Computer with GitHub Desktop	20
<i>Cloning a Repository that You Own or Administer</i>	<i>21</i>
<i>Cloning a Repository from Someone Else</i>	<i>22</i>
<i>After Selecting a Repository to Clone</i>	<i>23</i>
<i>Changing your GitHub Desktop Integrated Code Editor</i>	<i>24</i>
<i>Making and Committing Changes in GitHub Desktop</i>	<i>25</i>
Viewing – Tracking changes in your files with GitHub and GitHub Desktop	25
<i>Viewing Commit History in GitHub Desktop</i>	<i>29</i>
<i>Viewing Commit History on GitHub.com</i>	<i>30</i>
<i>Complex Commit: Multiple Files at the Same Time</i>	<i>33</i>
<i>Making a Remote Change in the GitHub.com Code Editor</i>	<i>35</i>
Catch up with changes from GitHub	35
<i>Pulling Remote Changes in GitHub Desktop</i>	<i>39</i>
Committing Code Changes	42
<i>New Code File and First Functions</i>	<i>42</i>
<i>Documenting Code</i>	<i>44</i>

<i>Fixing and/or Enhancing Existing Functions</i>	45
<i>Making Multiple Related Changes for a Group of Functions</i>	47
<i>Making Multiple Unrelated Changes for a Group of Functions</i>	48
<i>Other Times to Commit Changes</i>	50
<i>First .gitignore</i>	51
.gitignore: How to Avoid Leaking Sensitive Data	51
<i>Adding to .gitignore</i>	53
<i>Ignoring a Folder</i>	53
<i>Wildcard (*) Patterns</i>	55
<i>Basic Filetype ignore</i>	56
<i>Ignore any file type with a common name</i>	57
<i>File names with variables</i>	58
<i>Ignoring Folders</i>	59
<i>Full .gitignore for our example</i>	60
<i>Turning on File Name Extensions</i>	61
Git History and Reverting Changes	62
<i>Basic History Navigation</i>	62
<i>Reverting to a Previous Commit</i>	64
Adding Collaborators to a Repository	66
<i>Utrecht University GitHub Organization</i>	69
Other Repository Administration Procedures	70
<i>Changing the Repository Name</i>	70
<i>Danger Zone!</i>	72
<i>Changing the Visibility of the Repository: Public to Private</i>	72
<i>Changing the Visibility of the Repository: Private to Public</i>	75
<i>Archiving a Repository</i>	77
<i>Deleting a Repository</i>	78
<i>Transfer Ownership</i>	79
<i>Mirror</i>	79

Introduction

1

Welcome to the Geo Data Team's Getting Started with Git and GitHub manual. This is a companion manual to the Getting Started with Git and GitHub workshop.

The goal of this manual and workshop is to get you (an individual) started with using Git and GitHub. The workshop demonstrates this with a GitHub.com account, GitHub Desktop, and an integrated code editor that works with GitHub Desktop, the author will use Visual Studio Code (VS Code) in the screenshots throughout this manual, but other code editors are available. A list of code editors that integrate with GitHub Desktop can be found in the GitHub Desktop documentation, we suggest Sublime, R-Studio, or VSCodium for beginners: [Configuring a default editor in GitHub Desktop - GitHub Docs](#)

This manual will introduce Git through the GitHub Desktop User Interface, and is intended for individuals to track changes in their code.

A workshop is in development to demonstrate how to collaboratively manage and develop code with Git and GitHub.

Introductory Concepts

Git is a version control program intended to track changes in code. It was developed for managing the millions of lines of code and thousands of programmers contributing to the Linux project.

Since its creation in 2005, Git has become the most popular version control software for programmers managing commercial, academic, and personal projects.

Technically, Git tracks changes in any basic text file, which can include code files from most (if not all) programming languages, certain kinds of documents, CSV tables, and many other file formats. See below for an incomplete list of examples of text-based files.

Git tracks changes in a **repository**. For simplicity's sake, you can consider a folder on your computer as a repository. When Git starts tracking a repository (folder), it takes a snapshot of all the files and subfolders at that time.

Changes to files and folders are recorded in Git when you perform a Git **commit**, not when you save a file in the repository. Commits record the changes made in the different files and folders of a repository, the time when the commit was made, and the user who made the change. All commits to a repository are saved and result in a timeline (aka **commit history**) of the project/repository. This allows the user to always go back in time and undo a particular commit.

All commits to the repository require a message (**commit message**) to explain the change made in the repository. Optionally, a **commit description** can additionally be included if

the commit message offered too little space to explain the change(s) made: While commit messages should be at most 5 words, descriptions can be up to a paragraph. By adding good commit messages and descriptions the user can efficiently find what changes are made when.

This means that the user doesn't need to save copies of different versions of files. The most recent version will be visible, but all changes are saved in the user's commit history so all previous versions may be restored if necessary.

Git has great power in large collaborative projects, with users writing code distributed across many computers. Git is capable of managing and resolving these changes by different people at different times. The way this happens will not be elaborated here as it is out of scope of this manual.

Git repositories can be hosted in multiple locations such as your local computer or on a remote server or website. You should always have a local version and a remote (online) version of your repository. Typically, users work in the local copy of the repository and after creating a commit they update the remote repository.

GitHub is a company (owned by Microsoft) that operates GitHub.com, and makes a desktop Git management software known as GitHub Desktop.

GitHub.com is a website (owned by GitHub) that hosts Git repositories, and has tools and integrations to help programmers better manage and deploy their code, collaborate with other programmers, and share their code with the world.

There are many other code sharing websites that also present Git repositories in convenient ways, such as GitLab, Gitea, and BitBucket.

GitHub Desktop is a computer program with a graphical user interface for Git, which can integrate easily into GitHub.com. It also has some other git functions to make searching through version history and viewing the changes easier.

While GitHub Desktop is highly integrated with GitHub.com, your remote repository is not required to be on GitHub.com. You can use GitHub Desktop with repositories hosted on GitLab, Gitea, and BitBucket.

Other graphical (non-command line programs) programs exist to manage Git repositories, and many are built into code editors. For example, you can also apply Git version control in RStudio, GitKraken, Atom and [many other programs](#).

Conventions Used

The directions in this manual are commonly followed by screenshots. In the chapters, the text with directions and guides will show before the screenshot. The screenshots are annotated in red, sometimes with a box containing or close by the annotation indicating the area of the screen to look at.

While in standard English writing punctuation marks are placed within quotes, the authors have made a conscious decision to place punctuation marks outside the quotes. This is so that what is exactly within quotes is exactly as it should be presented in code.

This box indicates a Tip

This box indicates a warning

This box will link to the official documentation. This manual was written in Spring 2023, and may be out of date in the future.

Incomplete List of Text-Based Files

Some files have binary and text versions and the same extension, these files are denoted in italics.

Documents

- LaTeX (.tex)
- Markdown (.md) - *We'll discuss more about Markdown in chapter 1*
- Open Document Foundation Text document (.odt)
- Microsoft Word Strict Open XML Document (.docx)
- Notepad (.txt)

Data

- Comma separated values, tab separated values (.csv, .tsv)
- EXtensible Markup Language (.xml)
- JS Object Notation (.json)
- Microsoft Office Strict Open XML Spreadsheet (.xlsx)
- Open Document Foundation Spreadsheets (.ods)

Code

- Python (.py)
- R (.r)
- Javascript (.js)
- C & C++ (.c, .cpp)
- C & C++ Headers (.h, .hpp)
- Java (.jar, .java)
- Matlab (.mat)
- Rust (.rc)
- Fortran (.f, .for, f77, f90)

2

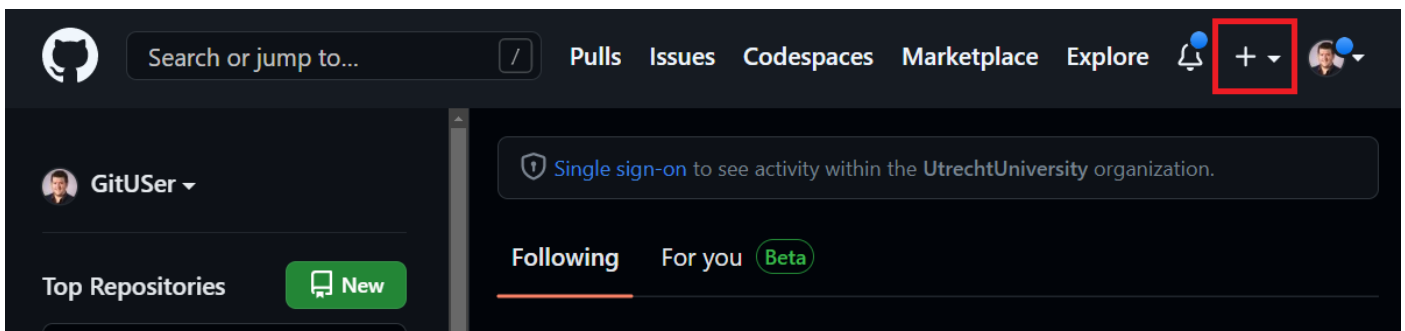
Creating Git Repositories

In this chapter, we will demonstrate how to create a Git repository both on GitHub.com and in the GitHub Desktop Application.

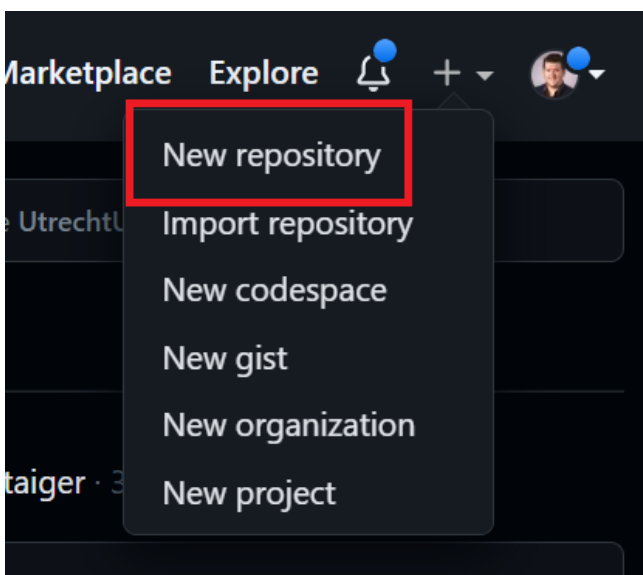
On GitHub.com

The official documentation for this process can be found here: [Create a repo - GitHub AE Docs](#)

When logged into GitHub.com, in the top right of each page there is a plus (+) icon, click on that to bring up the new content menu.



Next it will show you the new content menu, click on the option for creating a new repository.



A new page with the new repository form will appear. There are only a few fields that need to be filled in.

For users that have connected their account to another organization, such as [the Utrecht University organization](#), they will need to select an owner of the repository. You can select the owner in (1).

The second item is a name for the repository, you can enter your own name in (2), but GitHub will also suggest a random selection of words just below (3).

After that there is the description field (4) where you can write a sentence to describe what you intend your code or repository to be used for.

There are some naming conventions for GitHub repositories that may be helpful when you name a repository. This is a subjective topic, most common is to use lowercase characters and dash symbol '-' to separate words.

<https://climbtheladder.com/10-github-repository-naming-best-practices/>

Search or jump to... / Pulls Issues Codespaces Marketplace Explore

Create a new repository

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

Single sign-on to see more options within the UtrechtUniversity organization.

Owner * Repository name *

1 Select an owner / 2



Great repository names are short and memorable. Need inspiration? How about **sturdy-octo-chainsaw**? 3

Description (optional)

4

Further down the form is the visibility section of your repository, you can make it either public or private.

If you make your repository private at first, and later make it public, your Git version history will still be visible, so don't put private or otherwise sensitive information into your Git repository. In a later chapter of this manual, you will learn about adding files to a .gitignore file.

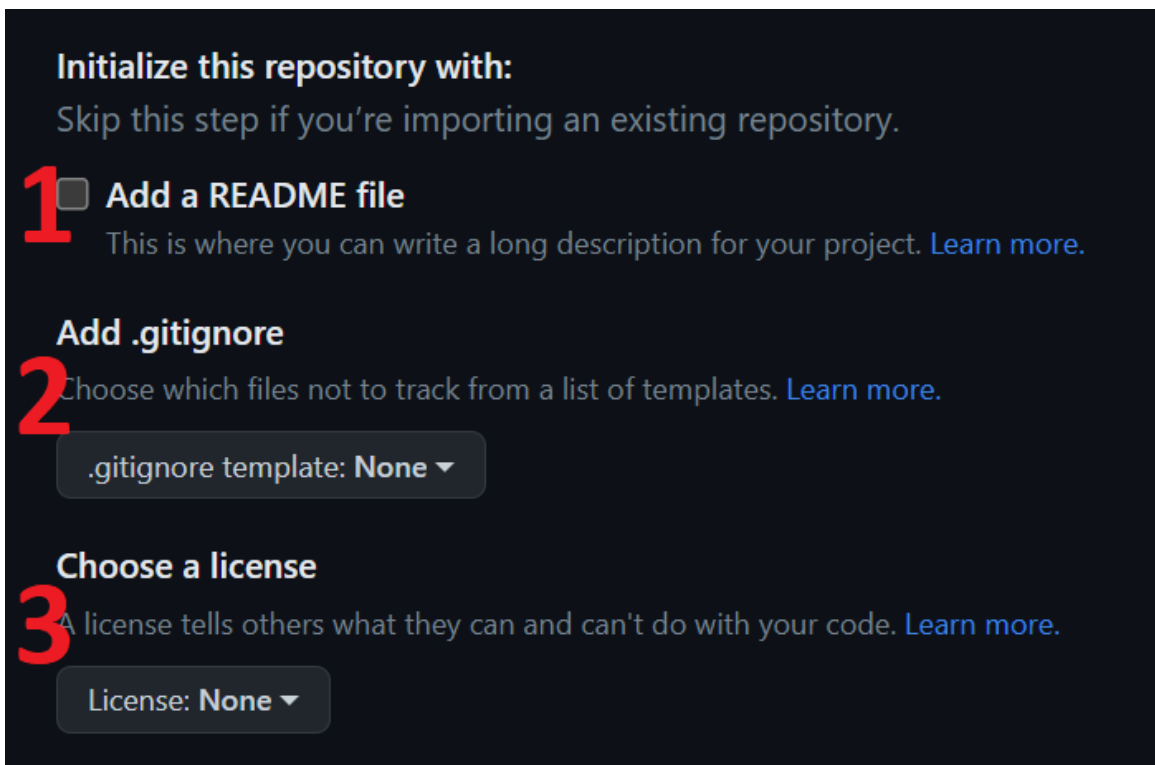
- ☐  **Public**
Anyone on the internet can see this repository. You choose who can commit.
- ☒  **Private**
You choose who can see and commit to this repository.

The last three options in the repository form are for nice things to initialize your repository with, and are always suggested.

The first (1) option is to have GitHub generate a README file. You should check the box, and GitHub will generate a README.md file with your repository name at the top of the file, a horizontal line, and then the description you wrote in in the description field.

The second (2) option is for a .gitignore file. We will discuss .gitignore files in a later chapter, but it is a helpful file to tell Git not to track certain files within the repository file. We suggest selecting the .gitignore template for your main programming language or framework, if one does not exist among the options here, select none. We will discuss more about .gitignore in a later chapter.

The third (3) option is to select a license for your code. The Geo Data Team suggests using either the GNU GPL 3 or the MIT license for code, for data or documents we suggest using CC-BY. You can learn more about code licenses on our [Code Licenses](#) page, which also has a selection tool linked.



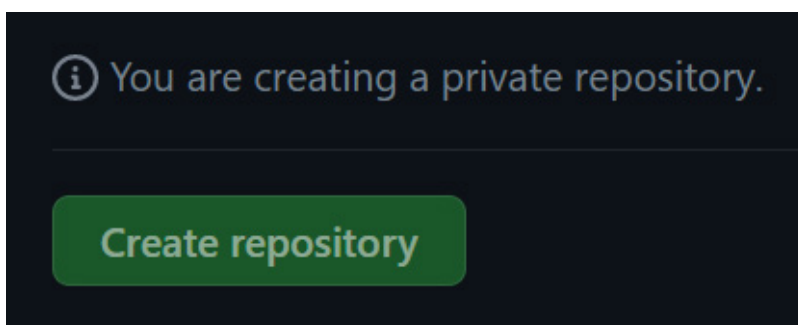
Initialize this repository with:
Skip this step if you're importing an existing repository.

1 ☐ **Add a README file**
This is where you can write a long description for your project. [Learn more.](#)

Add .gitignore
2 Choose which files not to track from a list of templates. [Learn more.](#)
.gitignore template: **None** ▼

Choose a license
3 A license tells others what they can and can't do with your code. [Learn more.](#)
License: **None** ▼

At the bottom of the page there is the confirmation and a reminder of the visibility of your repository.




i You are creating a private repository.

Create repository

As an example of a fully filled in form, we have included a screenshot of an example repository.

Owner *

 **GitUser** ▼


/

Repository name *


solis-oefenen-repo ✓

Description (optional)

Test repository for the Getting Started with Git and GitHub Workshop exercises.

☒  **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.

☒ **Add a README file**

This is where you can write a long description for your project. [Learn more.](#)

Add .gitignore


Choose which files not to track from a list of templates. [Learn more.](#)

.gitignore template: **None** ▼

Choose a license

A license tells others what they can and can't do with your code. [Learn more.](#)

License: **MIT License** ▼

 You are creating a public repository in your personal account.

Create repository

When you click “Create repository”, you should be directed to a GitHub repository page with your new information, such as from the example form above.

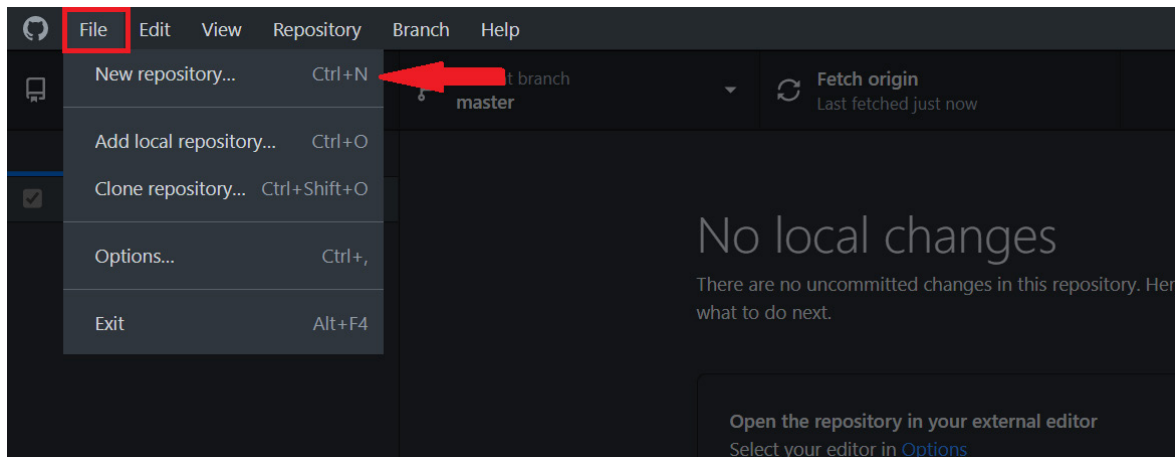
In this example, GitHub created a README.md file (1) in the repository, as well as a LICENSE file (3). GitHub, and many other Git viewers such as GitLab, Gitea, and BitBucket, automatically recognize a file named README.md as the documentation for a folder within a repository and will render the file below (2). As mentioned before, the README.md file that was automatically generated has the title of the repository, a horizontal line, and the description provided on the previous form page.

The screenshot shows a GitHub repository page for 'solis-oefenen-repo' by user 'GitUser'. The repository is public. At the top, there are buttons for 'Pin', 'Unwatch' (1), 'Fork' (0), and 'Star' (0). Below these are tabs for 'Code', 'Issues', 'Pull requests', 'Actions', 'Projects', 'Wiki', and 'Security'. The 'Code' tab is selected. Below the tabs, there are buttons for 'main', 'Go to file', 'Add file', and 'Code'. The 'Code' button is highlighted in green. Below these buttons, there is a commit history section showing 'GitUser Initial commit' 1 minute ago. The commit list shows two files: 'LICENSE' (3) and 'README.md' (1), both with 'Initial commit' and '1 minute ago'. Below the commit list, the 'README.md' file is expanded, showing the repository title 'solis-oefenen-repo' and the description 'Test repository for the Getting Started with Git and GitHub Workshop exercises.' On the right side, the 'About' section shows 'Test repository for the Getting Started with Git and GitHub Workshop exercises.' Below this, there are statistics: 'Readme', '0 stars', '1 watching', and '0 forks'. At the bottom right, the 'Releases' section shows 'No releases published' and a link to 'Create a new release'.

In GitHub Desktop

*The official documentation for this process can be found here:
[Creating your first repository using GitHub Desktop](#)*

When logged into your GitHub account through the GitHub Desktop interface, on the top left side of the screen there is a menu bar. Click on File, to bring up the content menu. On the content menu click on the option New repository...



A new dialog box with the new repository creation form will appear. There are only a few fields that need to be filled in.

The first field you need to fill in is the name of your repository (1).

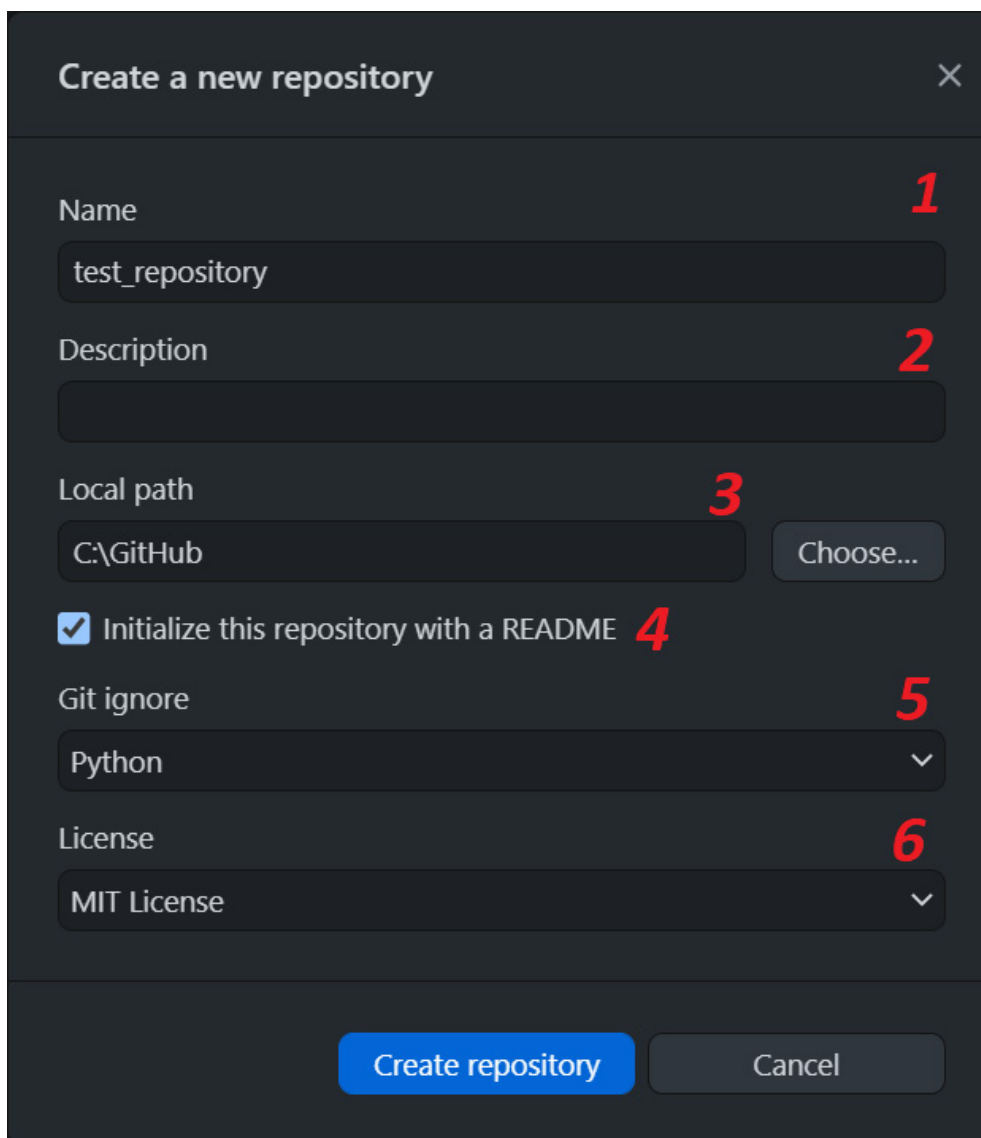
After that there is the description field (2) where you can write a short text to describe what you intend your code or repository to be used for.

The third field (3) can be used to specify the Location in your PC where this repository will be initialized-located.

Then by checking the box (4), you can choose either to initialize a README file in your repository (recommended) or not. GitHub Desktop will generate a README.md file with your repository name at the top of the file, a horizontal line, and then the description you wrote in in the description field.

The fifth (5) option is to choose to include a .gitignore file in your repository. The benefits of including such a file in your repository will be discussed later.

The final (6) option is to select a license for your code. The Geo Data Team suggests using either the GNU GPL 3 or the MIT license. You can learn more about code licenses on our Code Licenses page, which also has a selection tool linked.



Create a new repository [X]

Name **1**
test_repository

Description **2**

Local path **3**
C:\GitHub [Choose...]

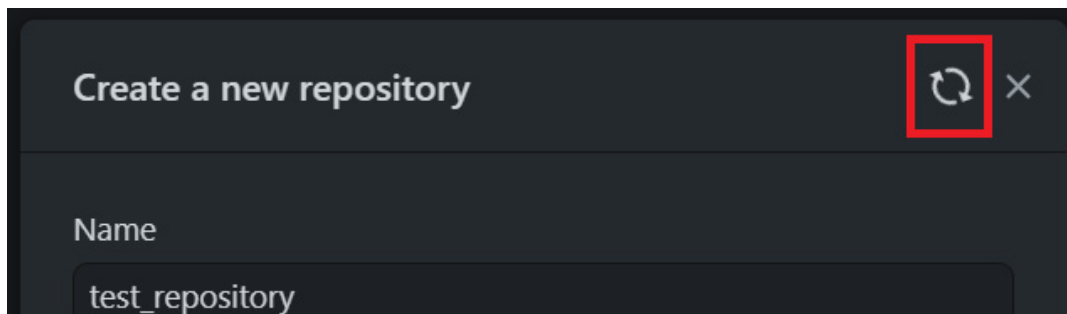
☒ Initialize this repository with a README **4**

Git ignore **5**
Python [v]

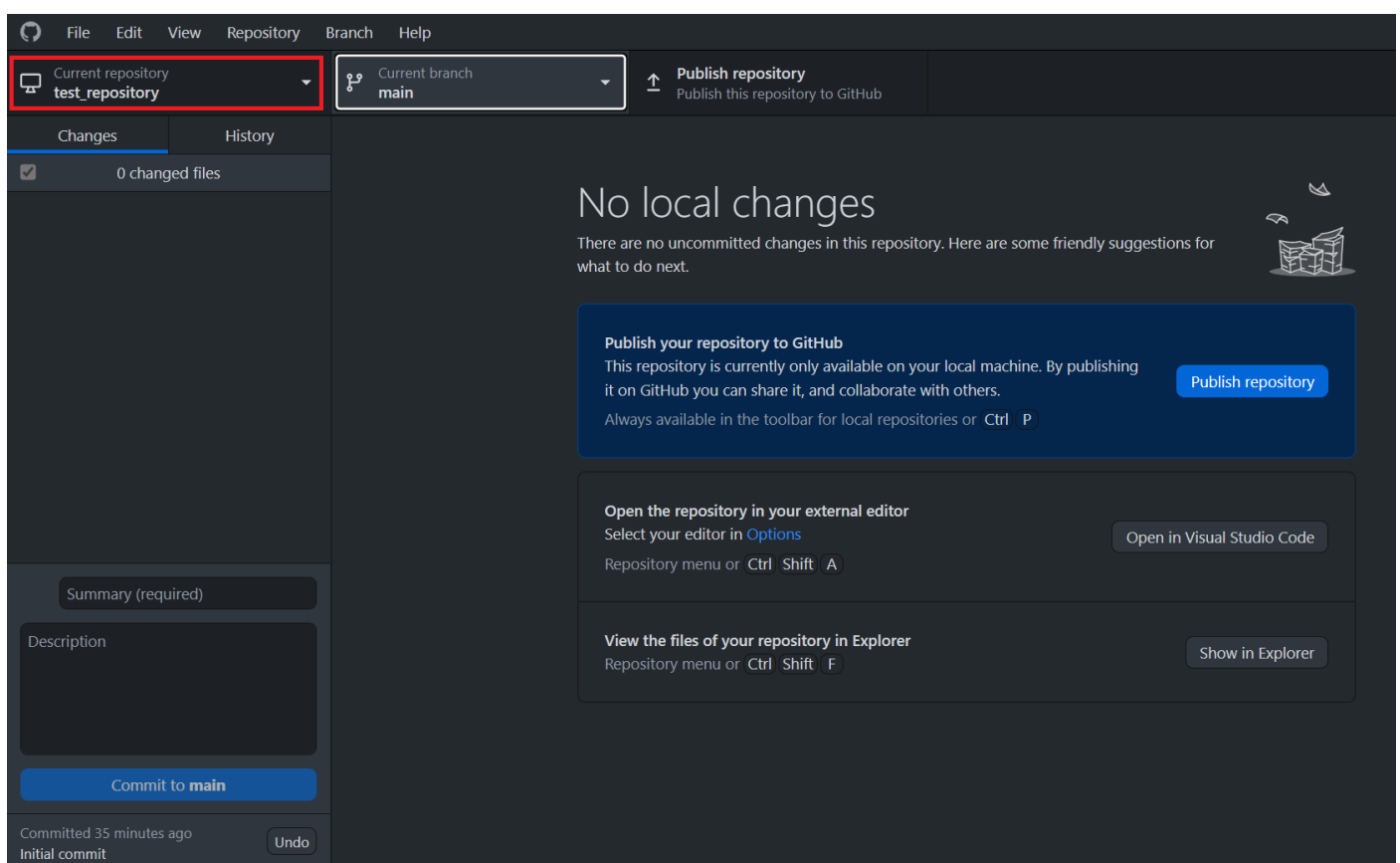
License **6**
MIT License [v]

[Create repository] [Cancel]

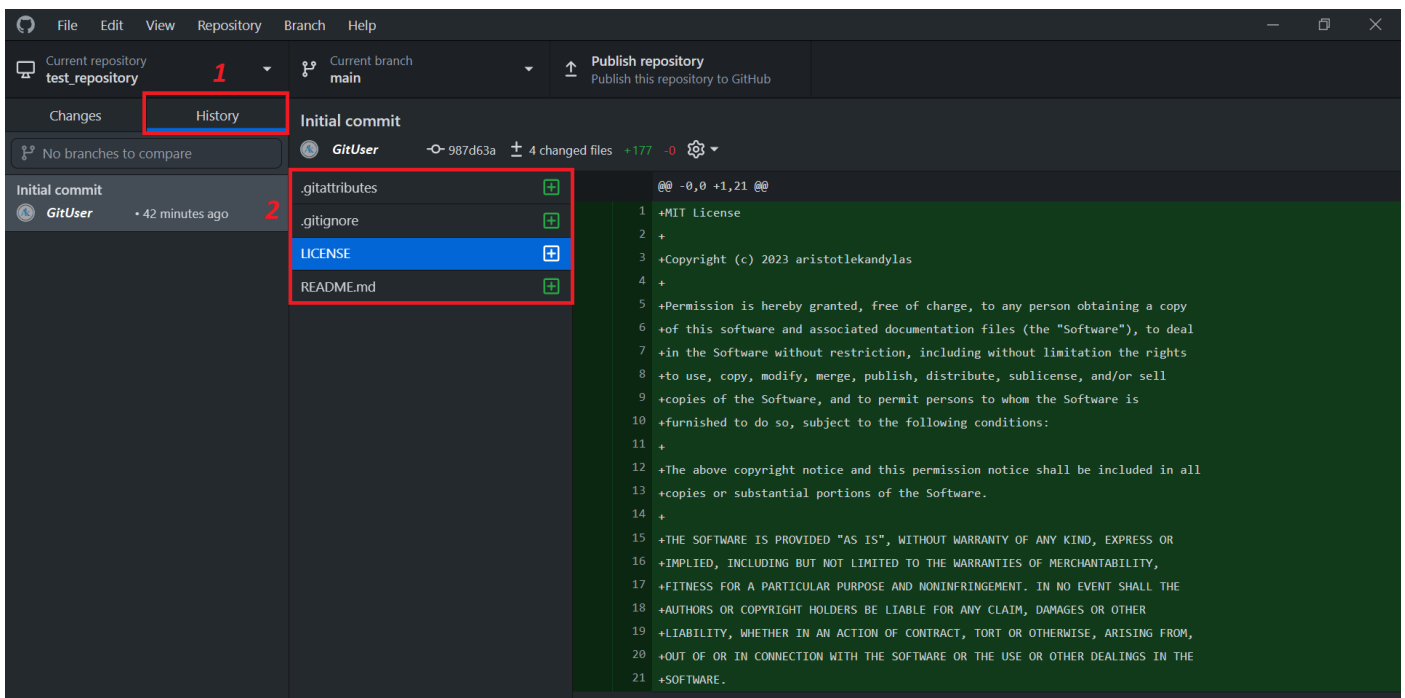
When you click on Create repository, you should see a loading icon on top of the dialog box. This icon means that your new repository is being created.



With the completion of the creation process, the new repository will appear on GitHub Desktop interface.



By clicking on History (1), you can see all the files (2) included in the repository along with the corresponding changes, after the initial commit.



Markdown

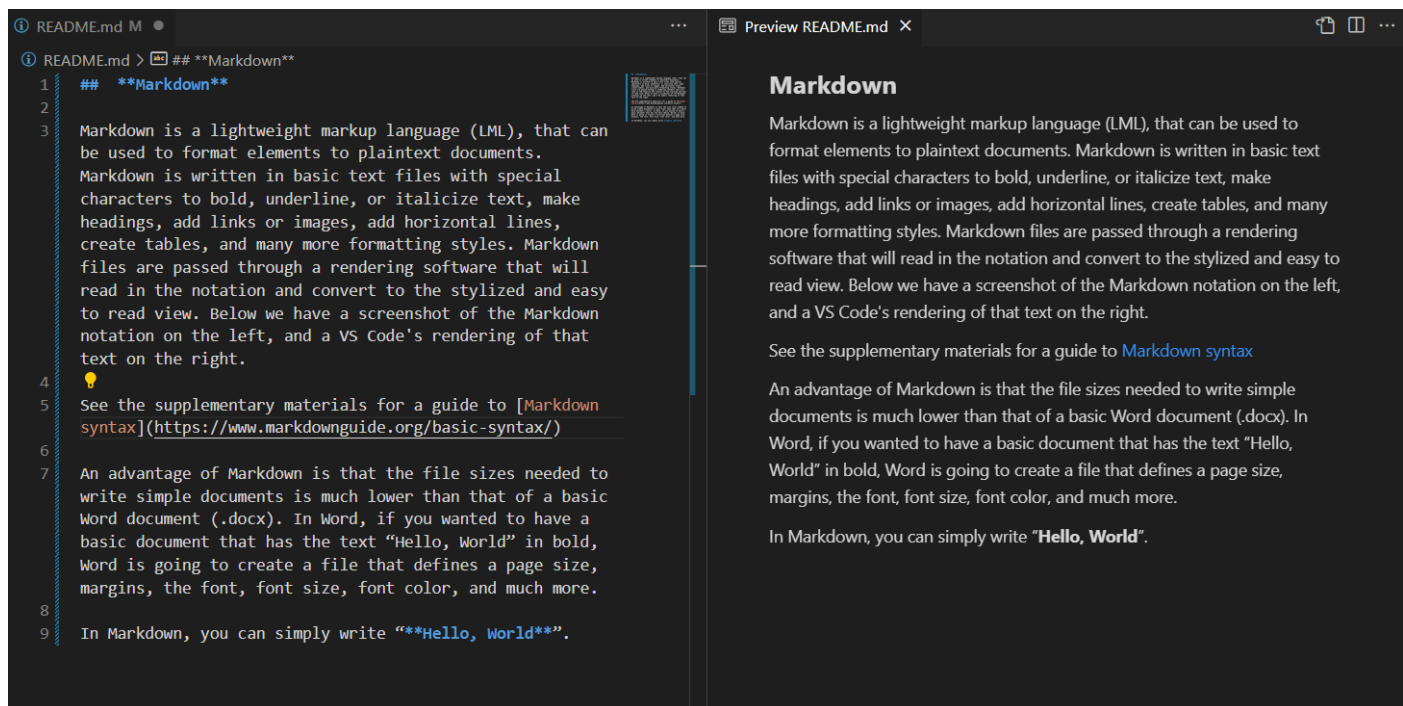
Every README file at the root of a Git(Hub) repository is a Markdown file, which you can see from the extension ".md". Markdown is a lightweight markup language (LML), that can be used to format elements in plaintext documents. Markdown is written in basic text files with special characters to bold, underline, or italicize text, create headings, add links, images, or horizontal lines, create tables, etc. In the background, markdown files are passed through rendering software that will convert the Markdown text into the a styled and easy to read view. Below, we have a screenshot of the Markdown notation on the left, and a VS Code's rendering of that text on the right.

See the supplementary materials for a guide to Markdown syntax:
<https://www.markdownguide.org/basic-syntax/>

An advantage of Markdown is that the file sizes needed to write simple documents is much lower than that of a basic Word document (.docx). In Word, if you wanted to have a basic document that has the text "Hello, World" in bold, Word is going to create a file that defines a page size, margins, the font, font size, font color, and much more. Additionally, in contrast to Microsoft Word, Markdown is an open format, so it can be read by many different Markdown editors and used by anyone without having to pay a license for Microsoft Office.

In Markdown, you can simply write "***Hello, World***".

To replicate this section of the chapter as seen here, see the screenshot below:



The README.md file created by GitHub is very basic, we have created a GitHub repository of README file templates:

[UtrechtUniversity/README-Templates](#): A collection of README templates from around the UU and other institutions.

Cloning a GitHub Repository to Your Local Computer with GitHub Desktop

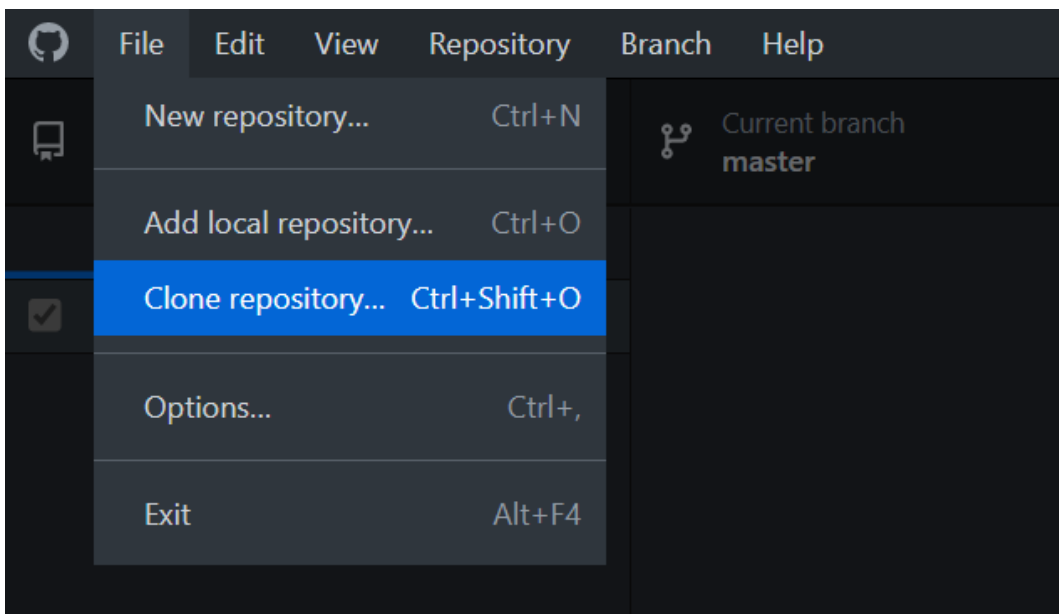
3

This chapter is for users who want to clone a pre-existing repository from GitHub.com to their local computer, such as the one just created in chapter 1.

The official documentation for this process can be found here:
[Cloning a repository from GitHub.com to GitHub Desktop](#)

When you **clone** a git repository, you are making a copy of the repository at another location, in this case on your local computer.

In GitHub Desktop, click on File, and select Clone repository.



From there, the repository cloning pop-up will appear, where you can clone from your own repositories on GitHub (1, as long as you're signed in to GitHub Desktop), or from other Git repositories (2) from either GitHub.com or another Git repository service.

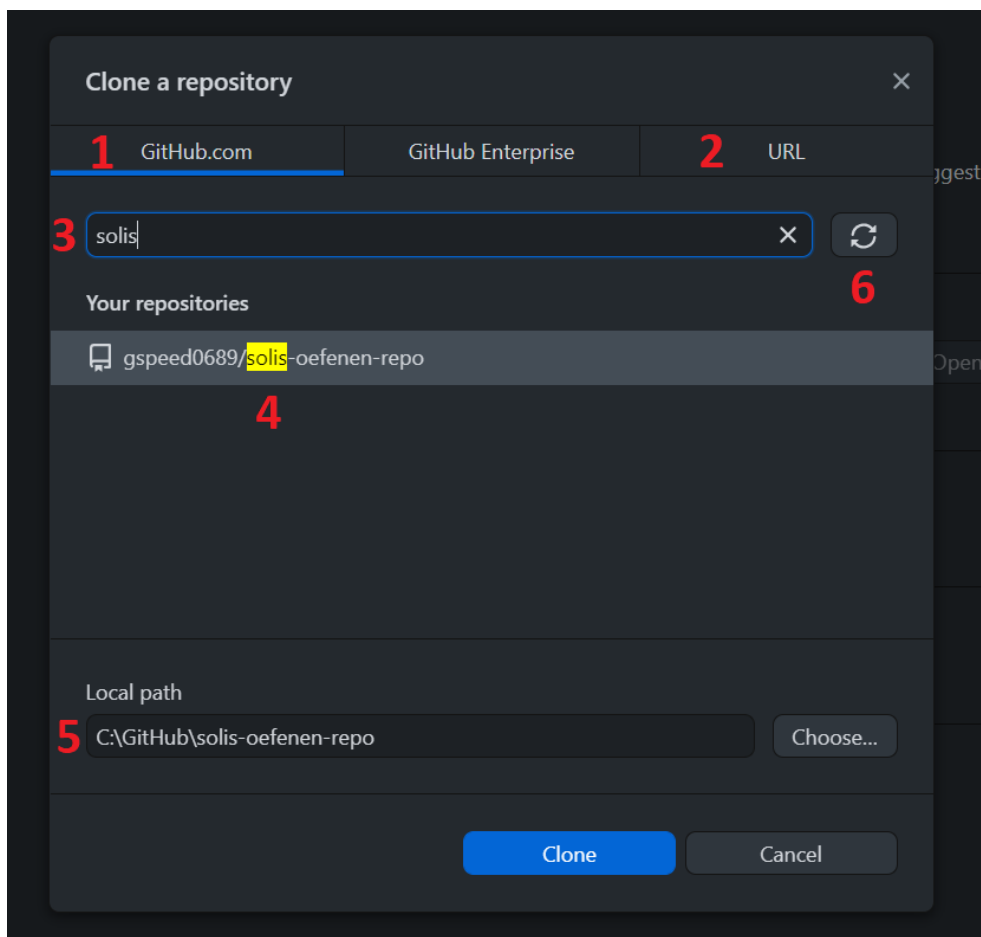
On UU owned Windows computers, there can be an issue with the synchronization features of your user folders (such as Documents, Pictures, etc.) with the U:\ drive. The Geo Data Team suggests creating a folder for Git repositories and code development at the base of your C:\ drive to avoid these synchronization issues. In the examples below, we have created a folder called "GitHub" on the C:\ drive, and GitHub will place inside that folder the example repository.

Cloning a Repository that You Own or Administer

If you have a long list of repositories in your GitHub.com account, such as the author of this manual, you can search through your repositories with the search bar (3), and GitHub Desktop will highlight the matching phrase in your repositories (4).

If your new repository is not showing up and you are logged into GitHub Desktop, click on the refresh listings button (6).

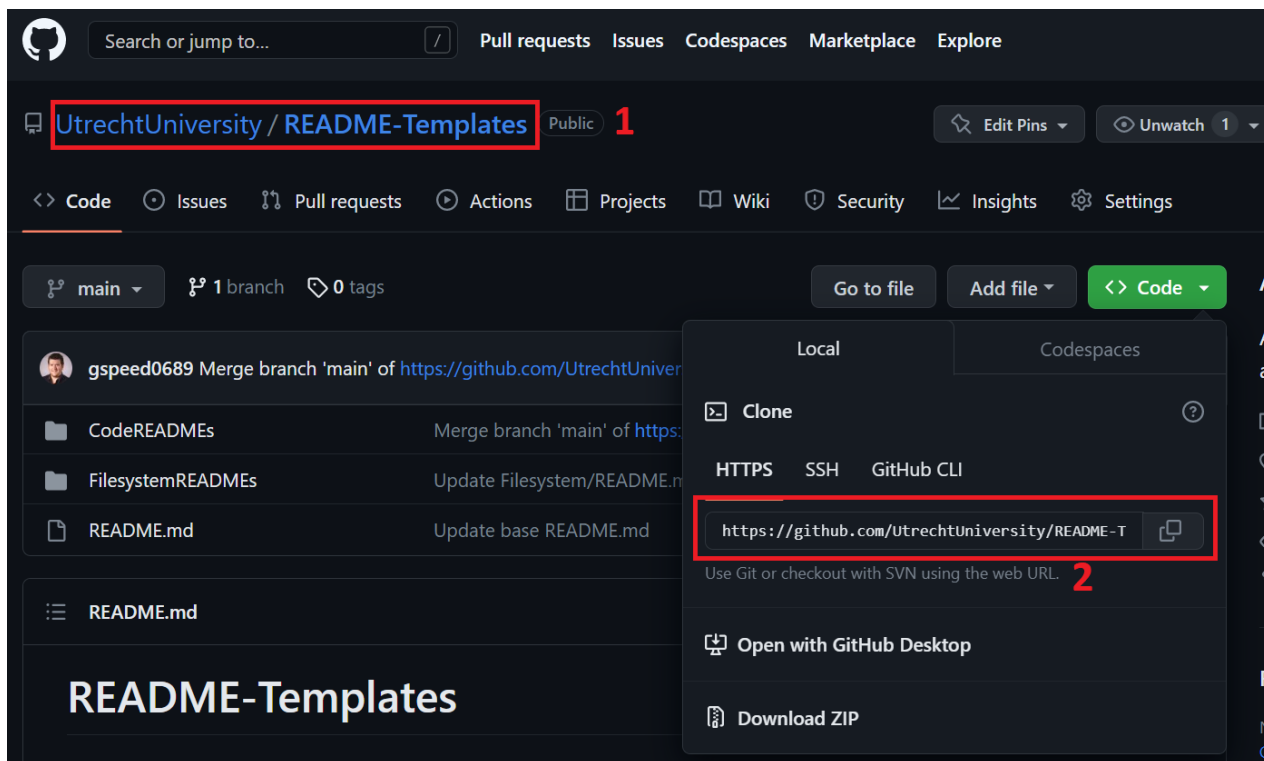
The final option when cloning your own repository is to select a local path to clone the repository into (5).



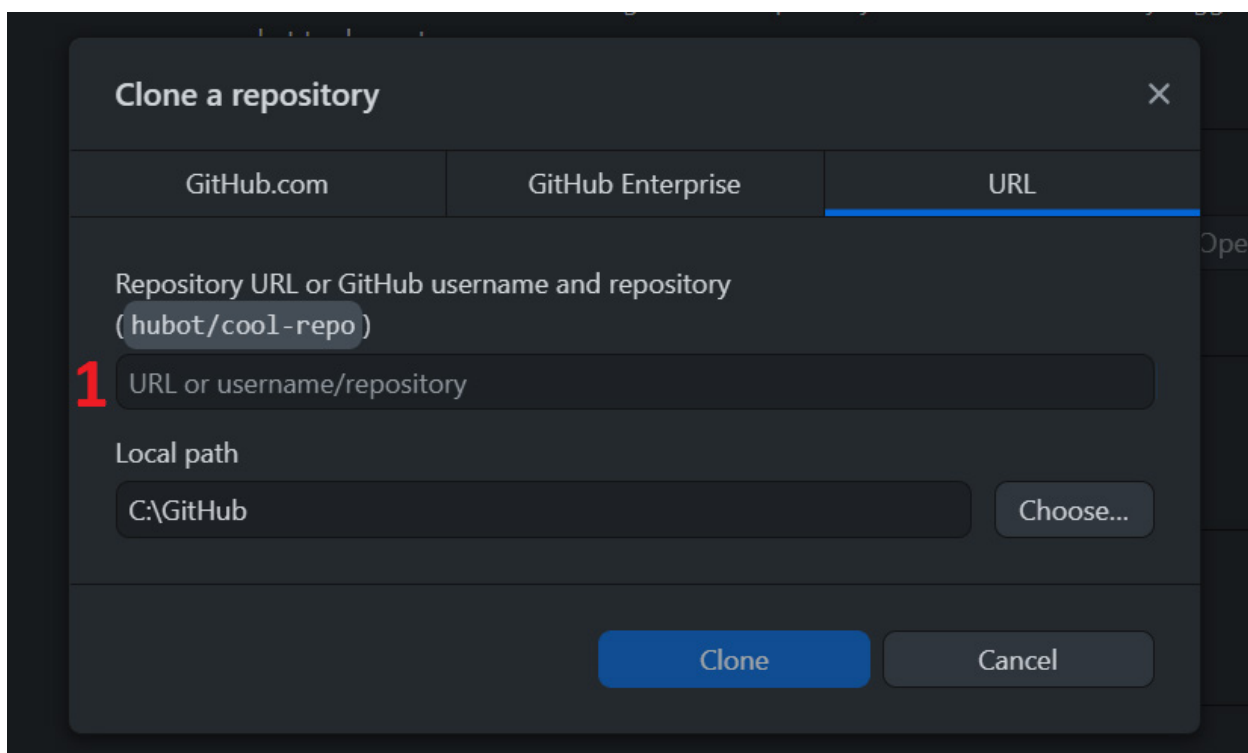
Cloning a Repository from Someone Else

If you want to clone a repository that you do not own or administer, you can use the URL option from above (2).

To find the URL of the repository you want to clone, consider this example below of the UU README-Templates repository. You can use the Username/Repository-Name format for items on GitHub (1), or the Git repository url (2, available after clicking the green <> Code button). We recommend using the Git repository URL (method 2), so you always clone the correct repository.

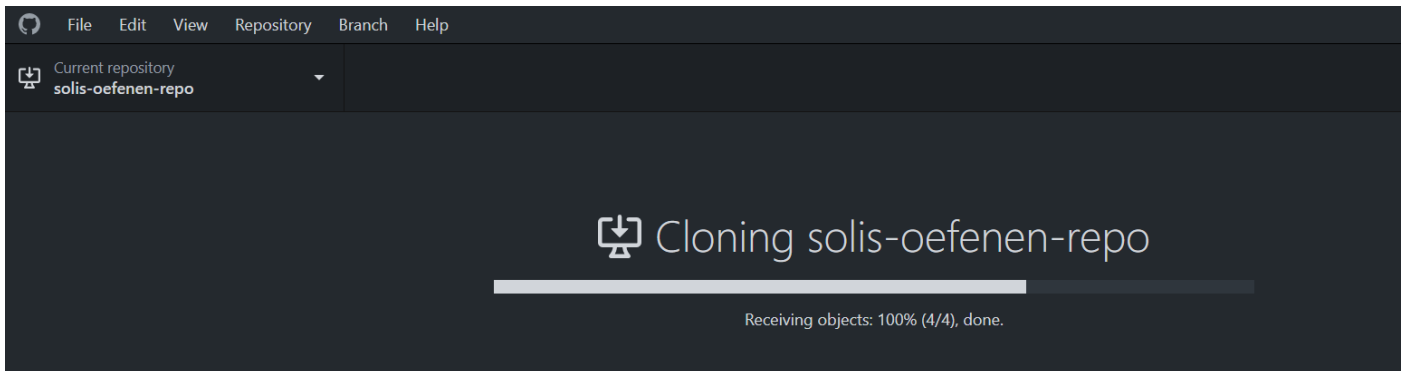


In the URL tab in GitHub desktop, place your target repository in the URL section (1).



After Selecting a Repository to Clone

After selecting your repository, selecting your copy file location, and pressing the Clone button, GitHub will begin to copy the items from the repository to your local computer.



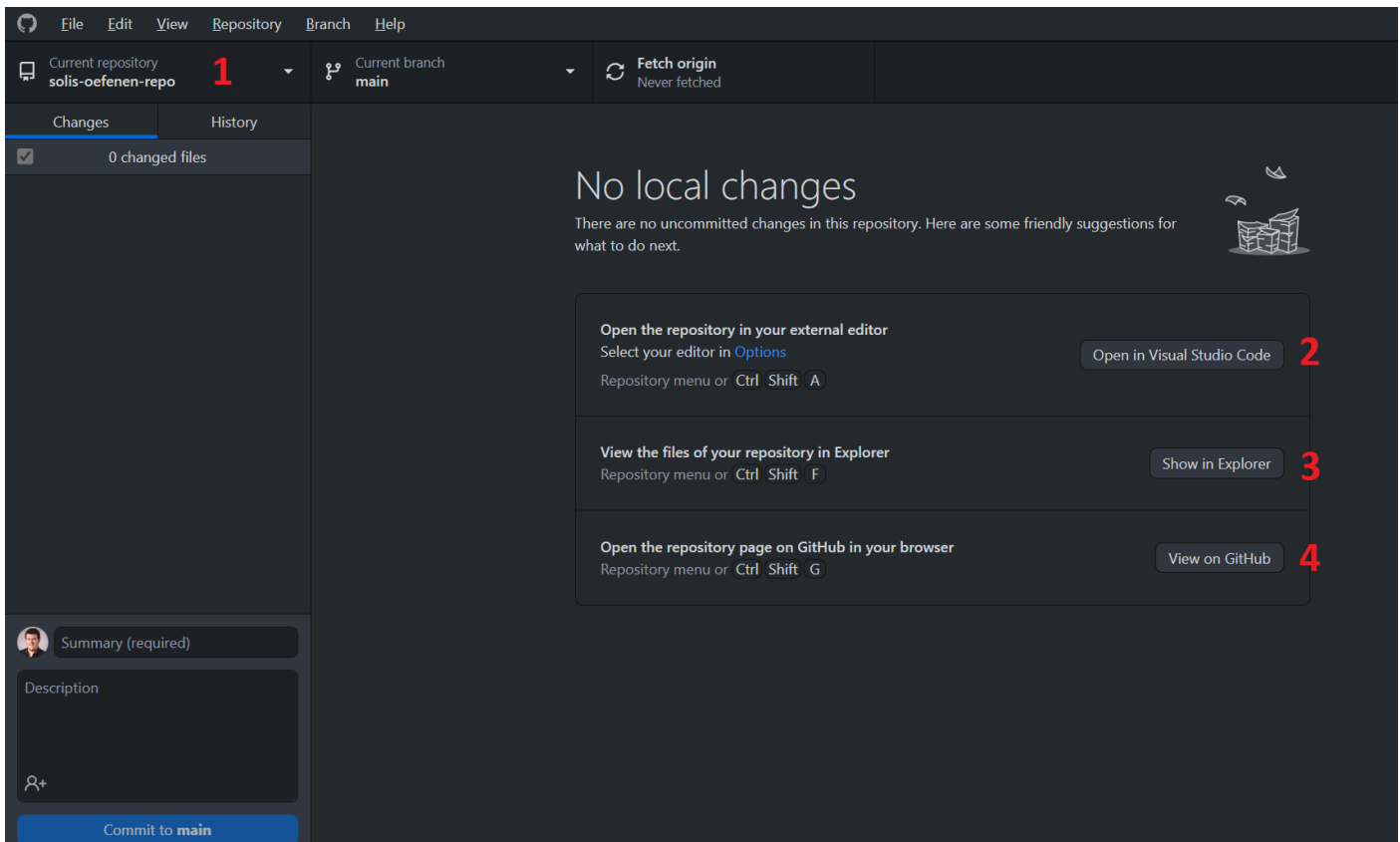
After the cloning process is finished, GitHub Desktop should look like the screenshot below. The different important sections of the screen are enumerated here:

(1) is the name of the current repository you are working on, the section to the right that says “main” will be important in a later manual.

(2) is a convenient tool to open your repository folder in your chosen code editor, and GitHub Desktop indicates which code editor is currently installed.

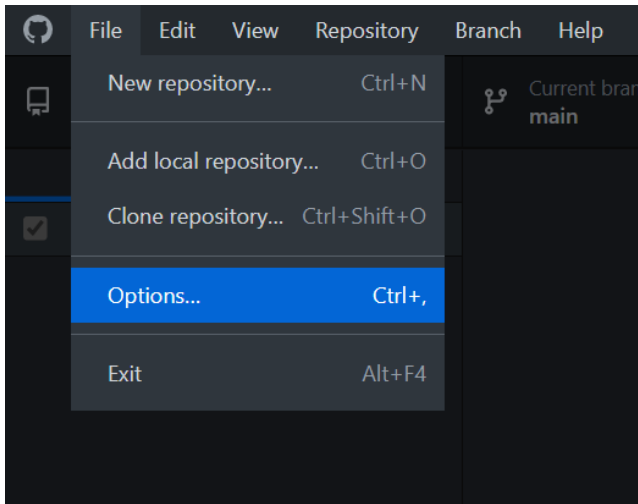
(3) is a convenient button to open the folder in your file system.

(4) is a button that will open your default web browser to the GitHub repository page, this button will not appear for repositories on other websites.

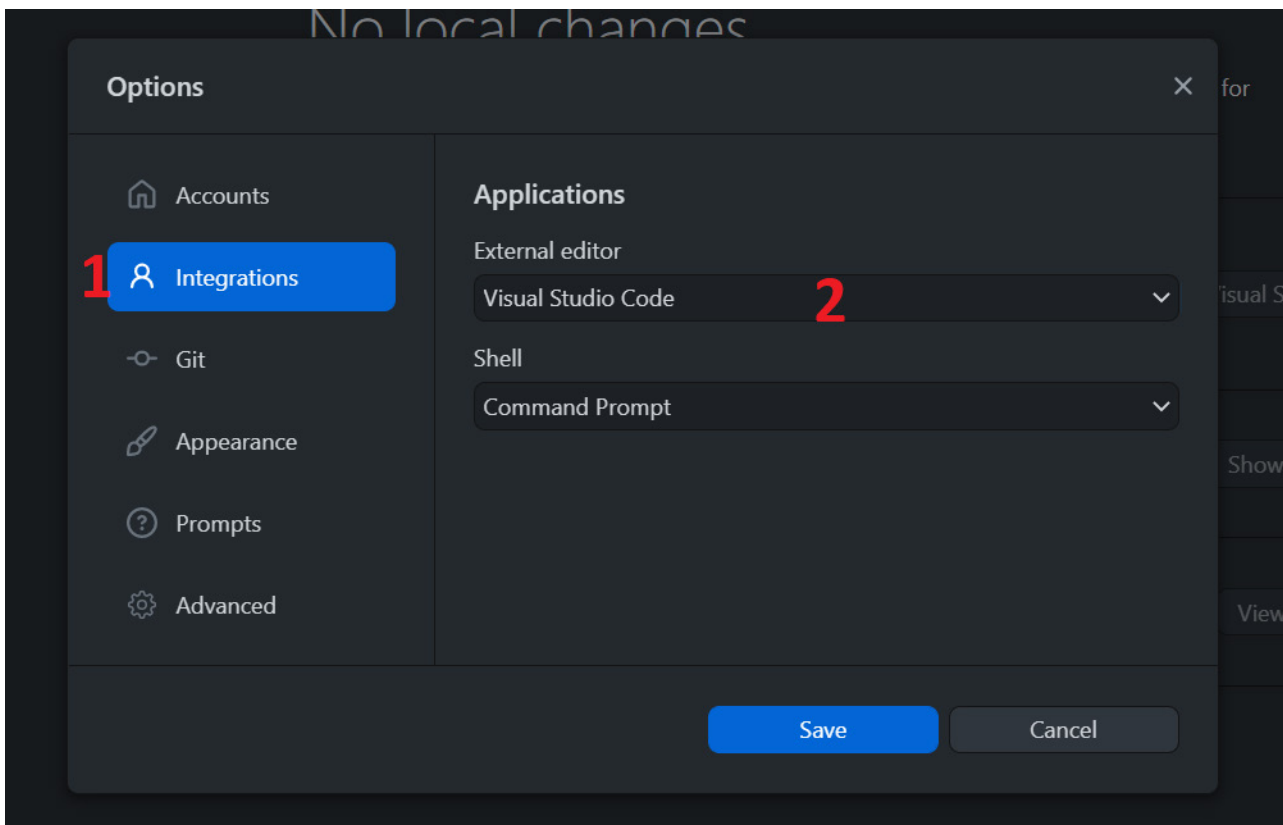


Changing your GitHub Desktop Integrated Code Editor

To change your GitHub Desktop code editor, click on File in the top left, then select options.



The options pop-up will appear, from there select the Integrations tab (1). The External Editors dropdown box (2) will show all of the code editors installed on your system that GitHub Desktop has integrations with, not all code editors are supported. A list of supported code editors is available [here](#), the Geo Data Team suggests Sublime Text, VS Code, VS Codium, or R Studio.



4

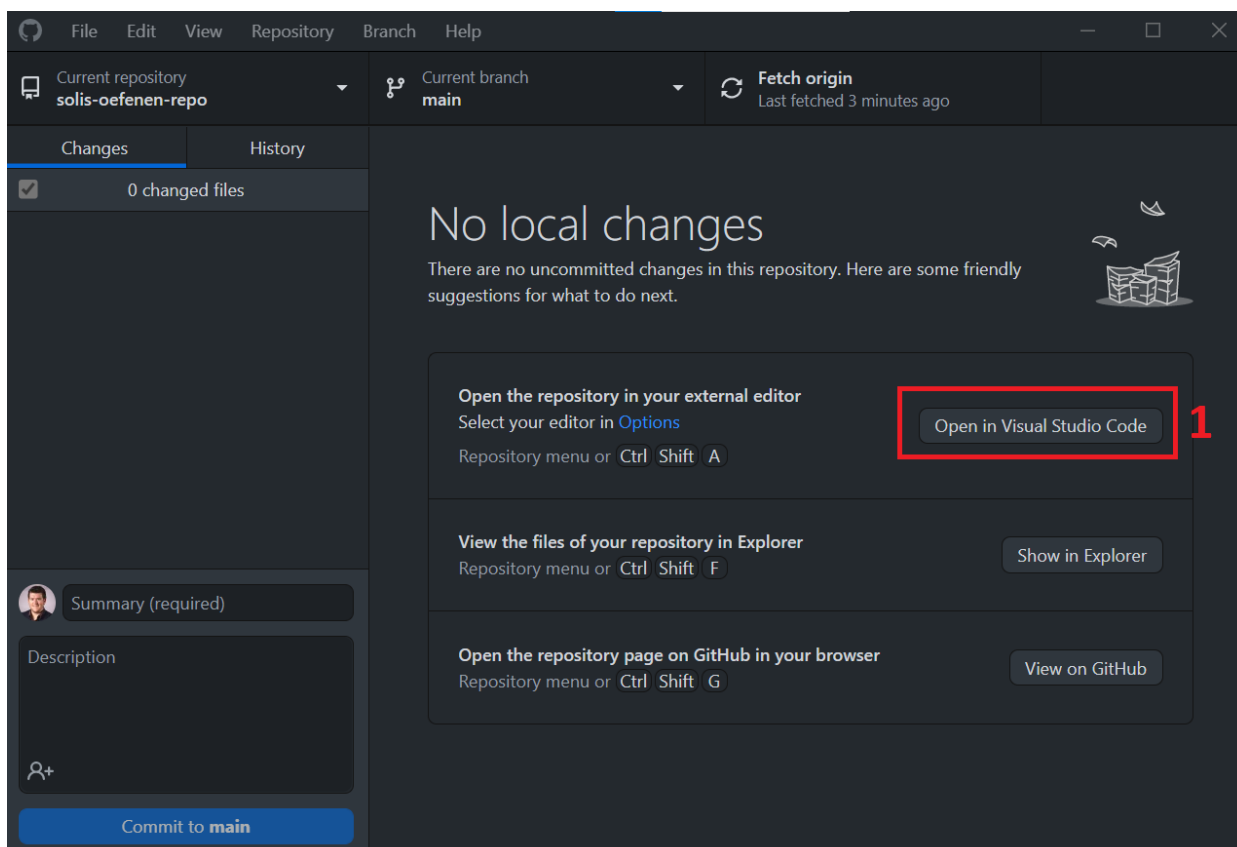
Viewing – Tracking changes in your files with GitHub and GitHub Desktop

One of the useful functionalities of GitHub and GitHub Desktop is the ability to track and review changes made in your repository. Specifically, when you add a file to git, it automatically captures a snapshot of the file's current state. Consequently, every time you update the file's content, GitHub Desktop colors the changes, based on whether they were additions (green) or deletions (red) in the file.

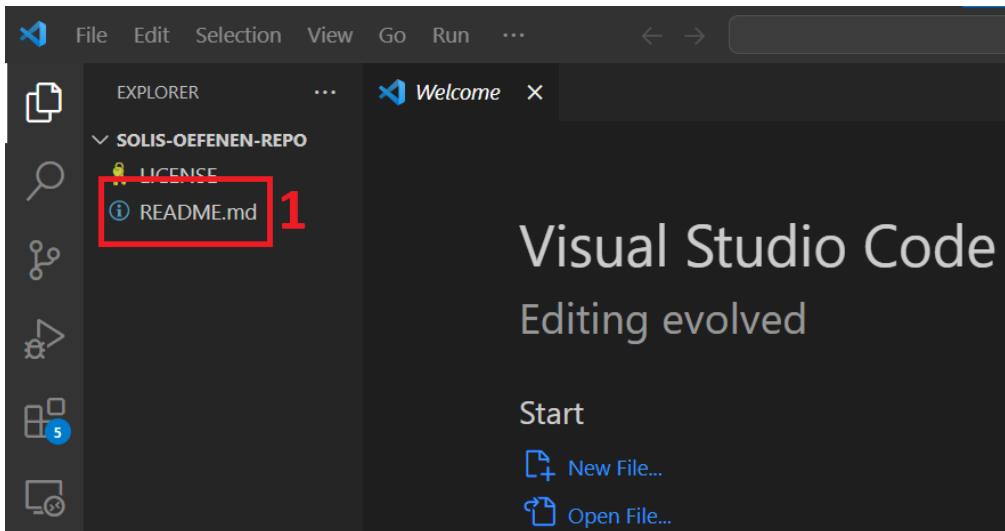
In this chapter, we will make changes on our local computers and on GitHub.com using the secret built-in code editor.

Making and Committing Changes in GitHub Desktop

First, let's make a change to a file in our repository, and for demonstration purposes we'll make a change to our README.md file. To do so, click the Open in {Code Editor} button (1). We are using VS Code in these screenshots.



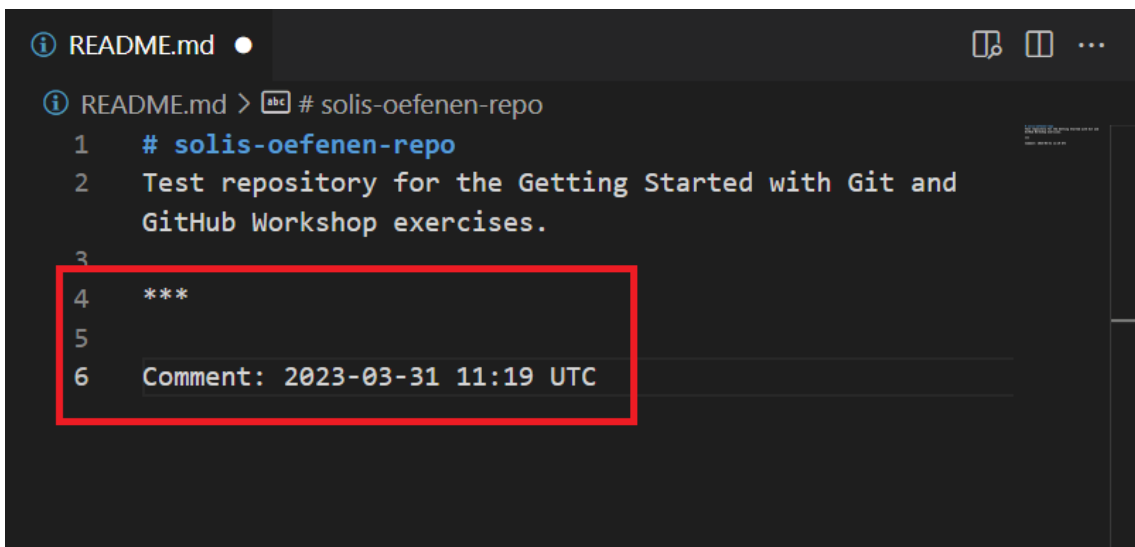
This will open your code editor with the repository folder in the built-in file explorer of your code editor. We're going to make an example change to my README file, so we're going to click on the README.md file (1) in the code editor.



This brings up the code for my README.md file.

At the end of my README.md file as is, I'm going to add 3 asterisks which will create a horizontal line on the page, and then write a comment with the time that I am making the comment in my code. See the screenshot below for the changes that I made.

.md are Markdown files, refer back to Chapter 1 for more information about Markdown files.



Click Save (or Ctrl+S), and then go back to GitHub Desktop.

GitHub Desktop should now look different, it is now in a view where you can inspect the changes from the previous state of the repository with the changes that have been saved but not committed.

If you have not saved the file, GitHub Desktop will not detect any changes were made. This brings up an important point: saving a file on your file system does not update the changes to your Git repository. The next steps will demonstrate how to commit (save) the changes to your repository.

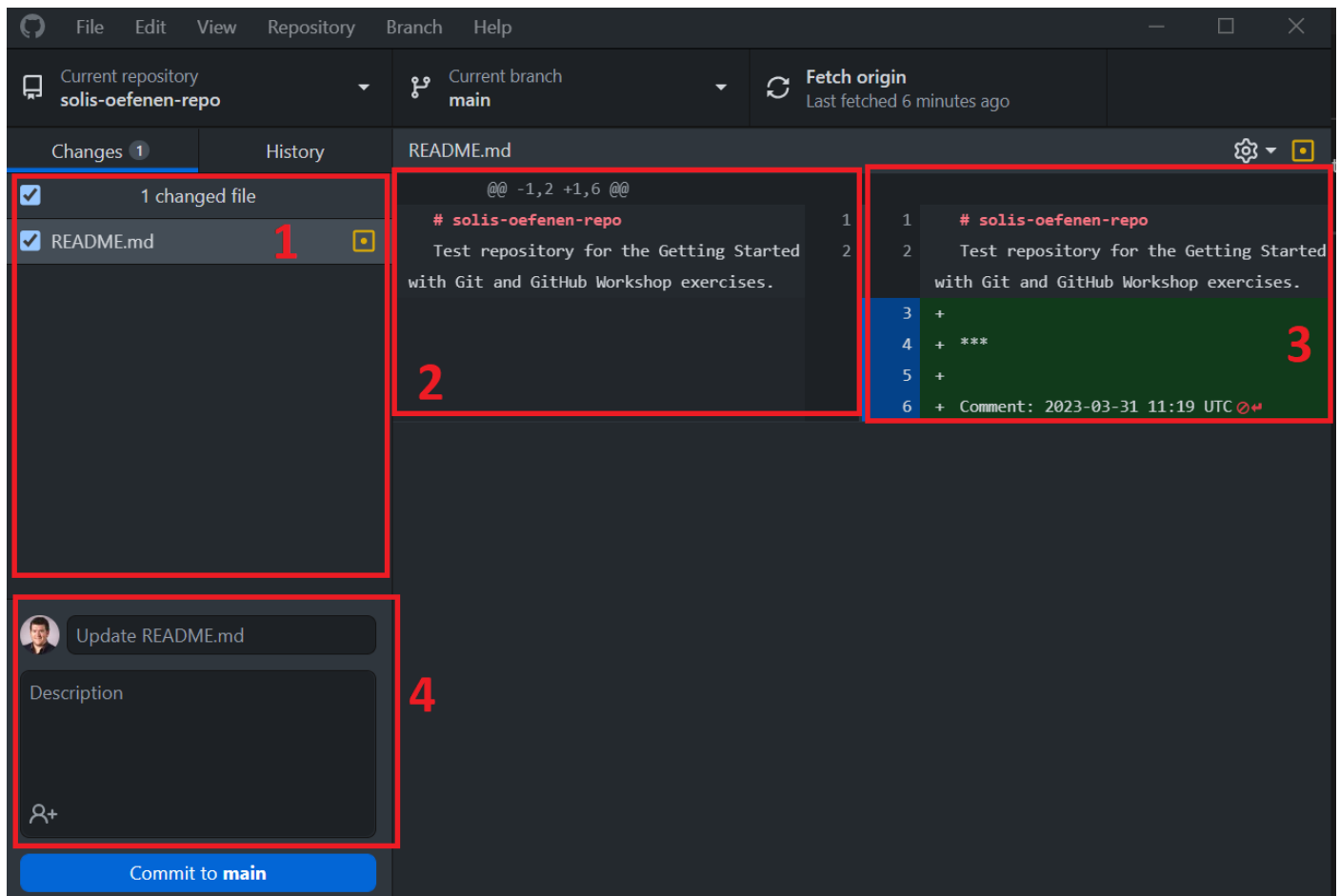
We will explain the different parts of the change view here:

(1) is the listing of the files that have been added, removed, or edited.

(2) is the current state of the files according to Git.

(3) displays the changes from the current state according to Git. Red lines indicate a line has been deleted, green lines indicate a line has been added.

(4) is where you write notes about what changes were made and why, and to commit those changes to Git.

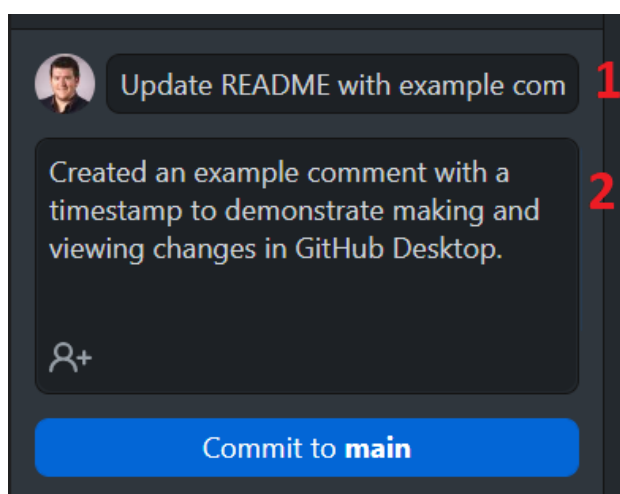


To save these changes to Git, we need to write a commit message. GitHub Desktop will autogenerate a generic (often not so informative) commit message for changes to a single file, but will require writing a message when multiple files are changed.

Commit messages are in the top single row line, and commit descriptions are in the larger text box. These are our recommendations:

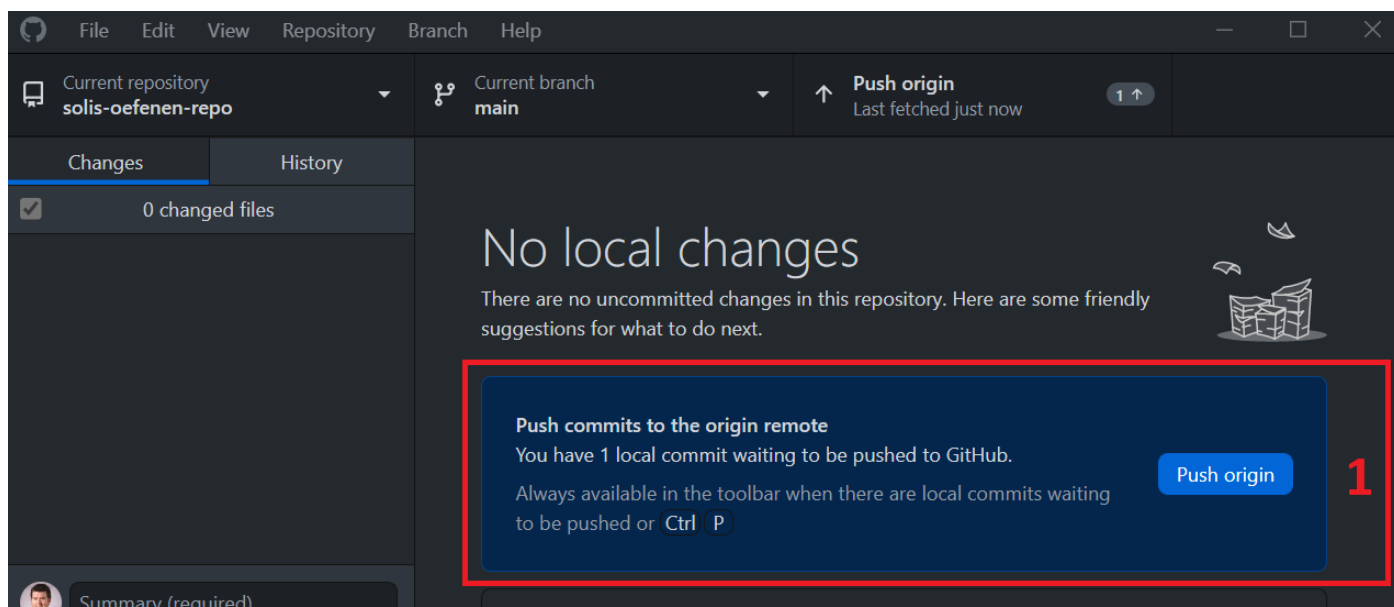
- Commit messages should be a short statement of 5 words or fewer.
- Commit descriptions are useful to further elaborate on the changes you made, but they are not required. If included, they should be at most a paragraph, but commonly a sentence is sufficient.

For better code management for yourself and others, you should write better commit messages than those suggested by GitHub Desktop. From the above example of a code change, look at the message (1) and description (2) we wrote:



See the section “Complex Commit” for an example of a commit with multiple changes made in a repository.

After committing changes (Commit to main), GitHub Desktop should return to the normal view, and have a new blue box (1) that indicates that local commits were made, but these commits are not on the remote repository, meaning the changes are only available locally, and not on GitHub.com. To send the changes to GitHub.com, click the Push origin button.



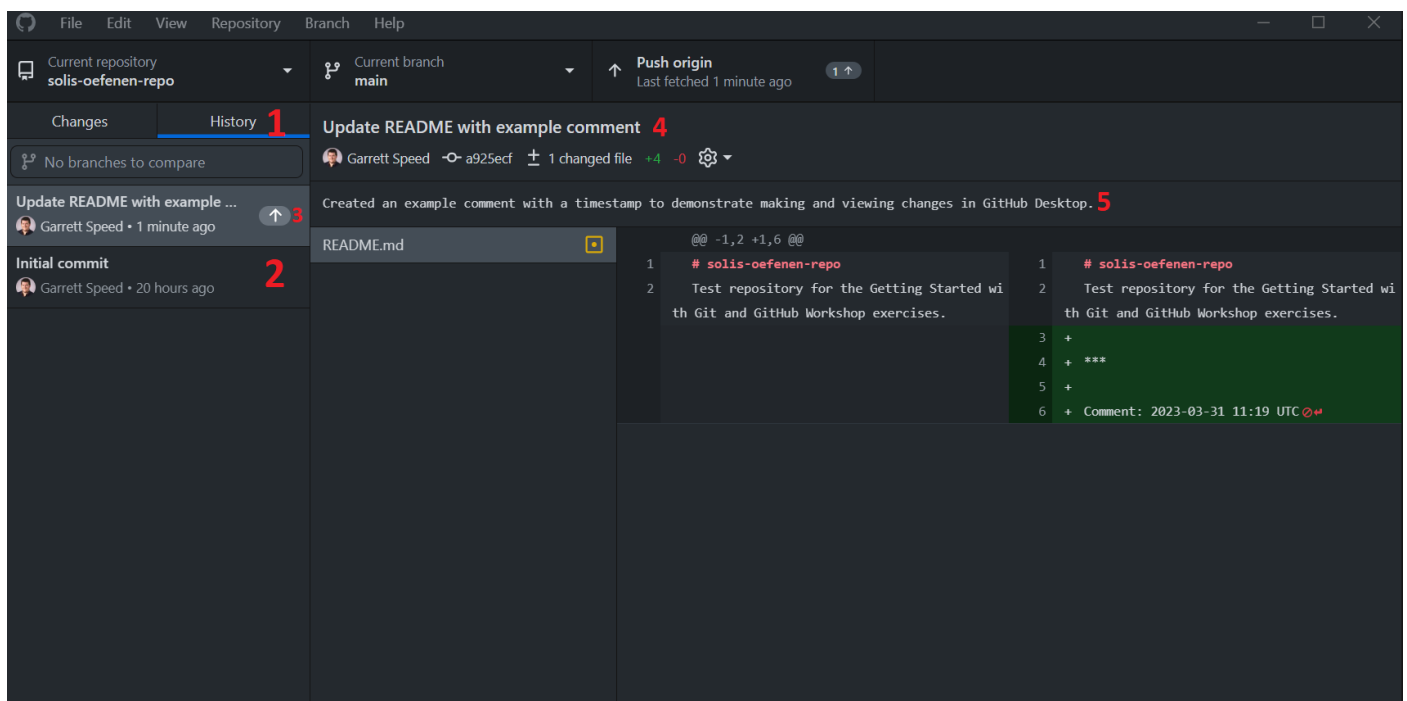
You do not have to push origin every time, you can make multiple commits before pushing to origin, however this is not recommended.

Viewing Commit History in GitHub Desktop

To see the changes made to a Git repository in GitHub Desktop, click on the history tab on the left (1)

(2) is the listing of commits made to the repository. If you clone a repository with a longer history, you will be able to see all the previous changes made to that repository since it was created, even if you did not make those changes yourself. Also as a note, the up arrow symbol at (3) indicates that the commit is still only on the local computer and not yet pushed to the remote repository on GitHub.com.

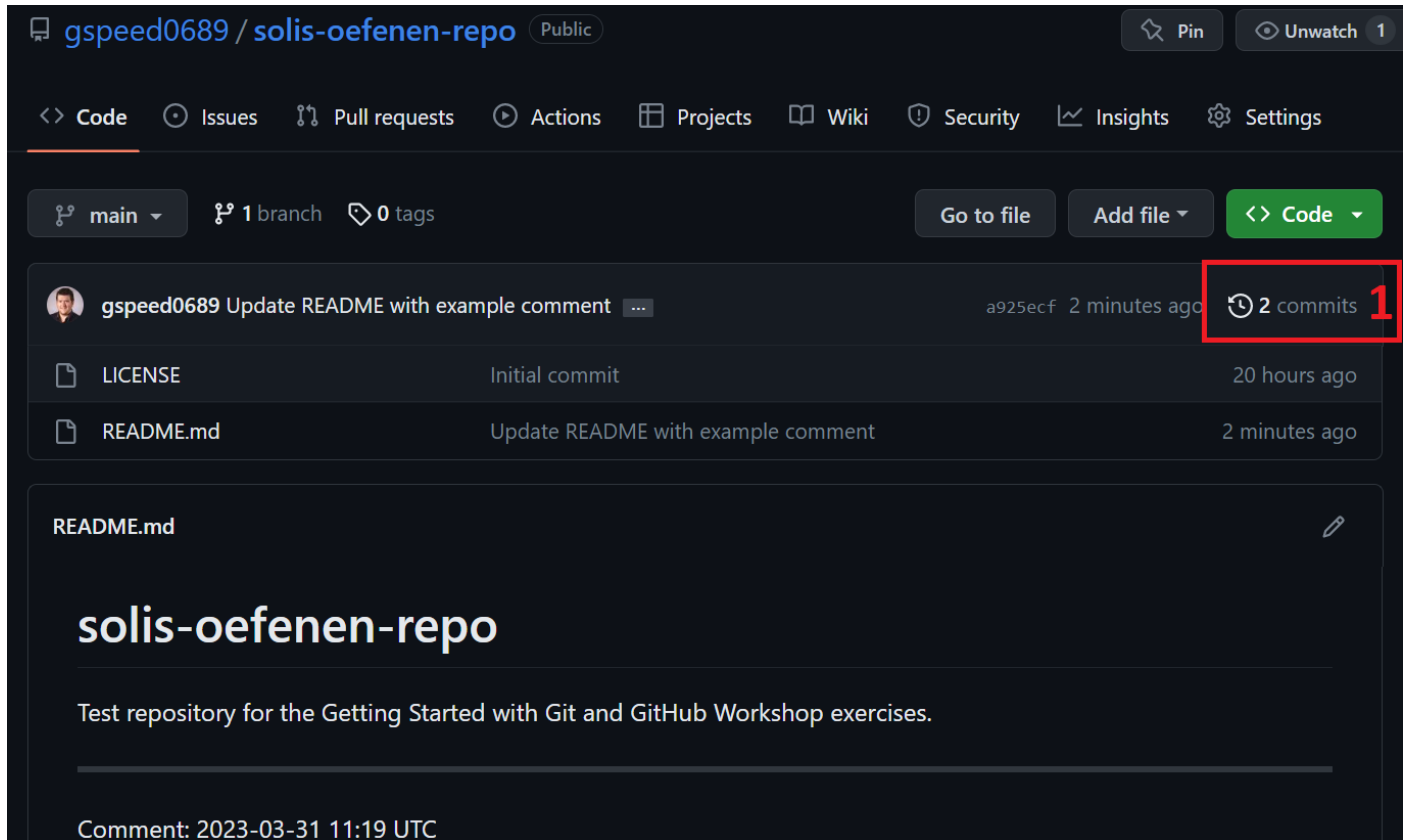
(4) is the commit message made on the currently selected commit, and (5) is the commit description from the same commit.



Viewing Commit History on GitHub.com

Navigate to your repository, and if it is on GitHub.com remember you can click the “View on GitHub” convenience button in GitHub Desktop.

From the repository on GitHub.com, click on the commits button with the counterclockwise arrow (1).



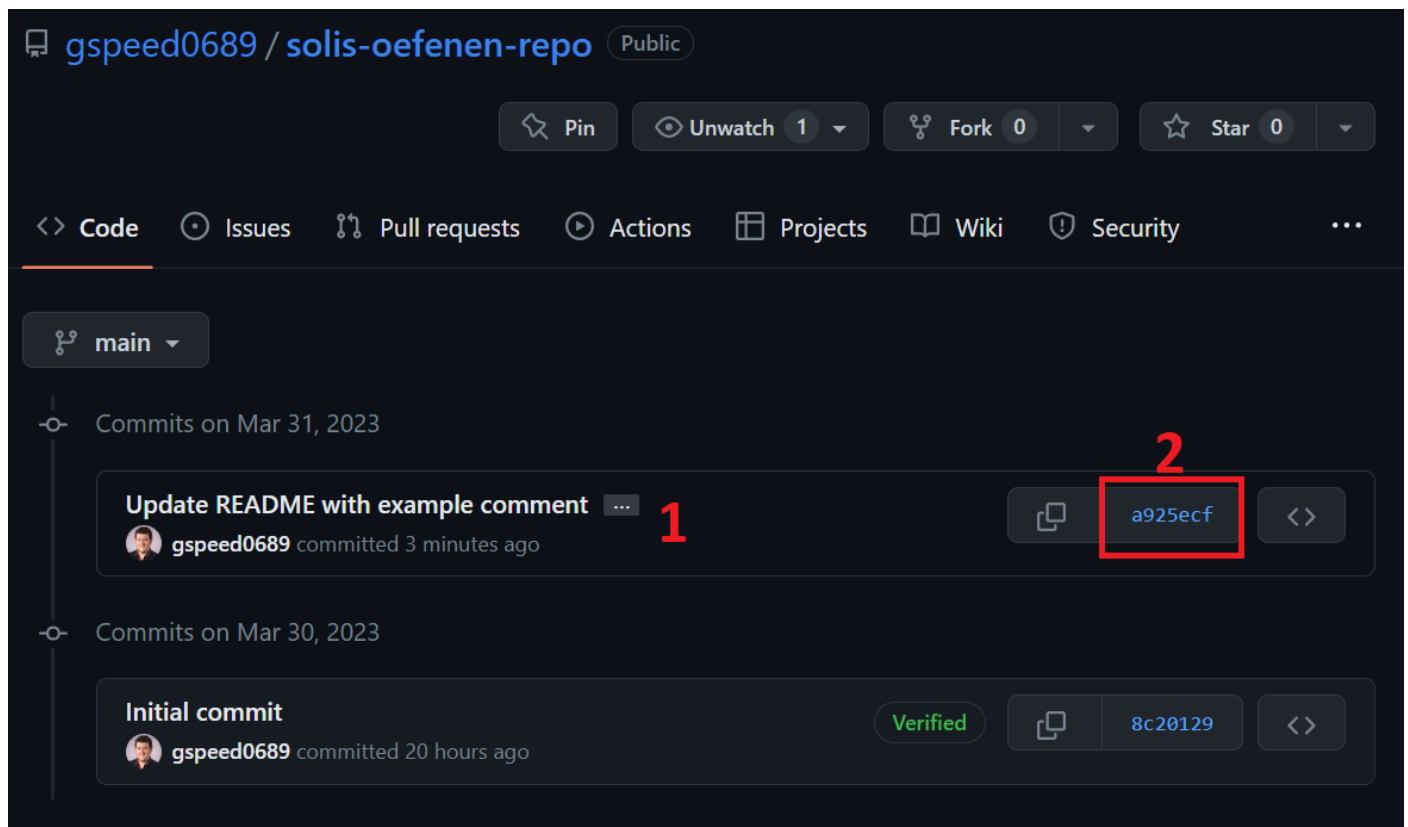
The screenshot shows the GitHub interface for a repository named 'solis-oefenen-repo' by user 'gspeed0689'. The repository is public. The top navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. Below this, there are buttons for 'main' (1 branch), '0 tags', 'Go to file', 'Add file', and a green 'Code' button. The commit history section shows a list of commits. The most recent commit is by 'gspeed0689' with the message 'Update README with example comment', commit hash 'a925ecf', and timestamp '2 minutes ago'. This commit is highlighted with a red box and a red '1' next to it, indicating it is the selected commit. Below the commit list, the README content is visible, showing the repository name 'solis-oefenen-repo' and a description: 'Test repository for the Getting Started with Git and GitHub Workshop exercises.' The footer shows the comment 'Comment: 2023-03-31 11:19 UTC'.

File	Commit Message	Commit Hash	Time Ago
LICENSE	Initial commit		20 hours ago
README.md	Update README with example comment	a925ecf	2 minutes ago

The next page will display a timeline of the commits made to the repository that have been pushed to GitHub.com.

Using the example of the change that was committed in the GitHub Desktop section of this chapter, we can see the commit with its commit message (1). In long changes made to your Git repository, having descriptive and concise commit messages is extremely helpful.

To inspect the changes made in each of these commits, click on the link on the right with the hexadecimal code (2). This hexadecimal code is a unique identifier for the commit, and is actually longer than what is displayed here.



You will then be brought to the commit inspection page. In (1) you will find the commit message and description. In (2) you will find the changes made to each of the files in the commit.

Update README with example comment

Created an example comment with a timestamp to demonstrate making and viewing changes in GitHub Desktop.

1

Browse files

main

 gspeed0689 committed 3 minutes ago

1 parent 8c20129 commit a925ecf

Showing 1 changed file with 4 additions and 0 deletions.

Split

Unified

4 README.md

2

<> [icon] [icon] ...

@@ -1,2 +1,6 @@

1 # solis-oefenen-repo

2 Test repository for the Getting Started with Git and GitHub Workshop exercises.

3 +

4 + ***

5 +

6 + Comment: 2023-03-31 11:19 UTC

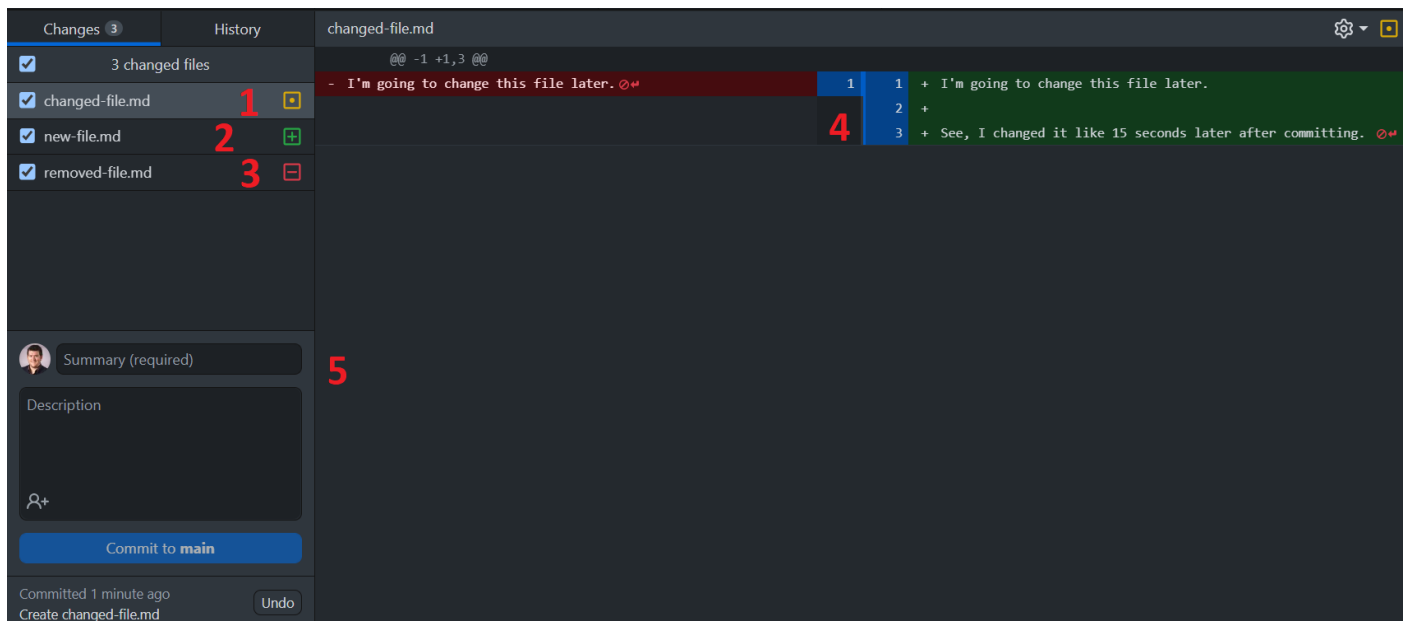
Complex Commit: Multiple Files at the Same Time

Now let's take a look at a more complex example with multiple changes to the repository. In this example we removed the "removed-file.md" file (3) from the repository, made a change to the markdown in the "changed-file.md" file (1), and added a new file called "new-file.md" (2). In this screenshot, you can also see the symbols for adding, removing, and changing files.

In the change display area (4), we are currently seeing the changed-file.md changes as that is the file selected on the left. If we clicked on the other files we would be able to see the changes made in each file.

In (5) we can see GitHub desktop is requiring a commit message, and it has not pre-generated any commit message for us, because multiple files have edits.

In the change area (4), the current state line on the left is highlighted in red, and the new version is green on the right, but there is no discernable change between the lines. This is because the line has changed, a newline character has been added to the end of the line, but newline characters are not displayed in most text editors in an easy to see way.



After committing these changes and pushing them to GitHub.com, we can see how the commit is structured in GitHub history. New file (1), changed a file (2), and removed a file (3).

Created a screenshot to demonstrate changes

[Browse files](#)

main

 gspeed0689 committed 51 minutes ago

1 parent 6330fd6 commit 6c922ba

Showing 3 changed files with 4 additions and 2 deletions.

[Split](#) [Unified](#)

4 changed-file.md 2

@@ -1,3 @@

1 - I'm going to change this file later.

2 + I'm going to change this file later.

3 + See, I changed it like 15 seconds later after committing.

1 new-file.md 1

@@ -0,0 +1 @@

1 + This is a brand new file.

1 removed-file.md 3

[Load diff](#)

This file was deleted.

34

5

Catch up with changes from GitHub

In the previous chapter, we looked at committing changes and pushing them to GitHub.com. Next, let's take a look at pulling new changes from GitHub that are not on our local computer. First, we will need to create a remote change, and then we will pull that change back to our local computer, so that our local copy is fully up-to-date. This is particularly useful if you collaborate with others on files in a GitHub repository, and want to make sure that you are not making changes that conflict with changes made by someone else.

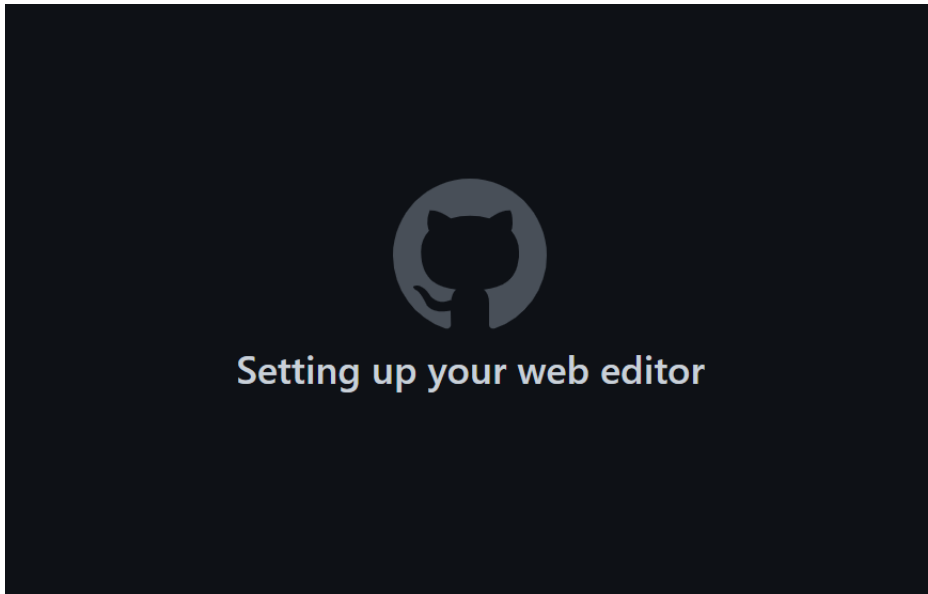
Making a Remote Change in the GitHub.com Code Editor

In order to make a remote change, we will take advantage of a "hidden" feature on GitHub.com, a built-in code editor. To access the secret code editor, in your repository on GitHub.com (such as in the screenshot below), press the "." (period/full-stop/punt) button on your keyboard.

The screenshot shows the GitHub interface for a repository named 'solis-oefenen-repo' by user 'gspeed0689'. The repository is public. The top navigation bar includes links for Pulls, Issues, Codespaces, Marketplace, Explore, and a user profile. Below the repository name, there are buttons for Pin, Unwatch (1), Fork (0), and Star (0). The main navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Wiki, and Security. The 'Code' tab is selected, showing a list of files: LICENSE, README.md, changed-fil..., and new-file.md. The README.md file is highlighted. On the right, the 'About' section provides information about the repository, including a description, README, MIT license, 0 stars, 1 watching, and 0 forks.

File	Commit Message	Time
LICENSE	Initial commit	yesterday
README.md	Update README with example...	1 hour ago
changed-fil...	Created a screenshot to demo...	1 hour ago
new-file.md	Created a screenshot to demo...	1 hour ago

This will launch a web version of VS Code in your browser with the text/code files from your repository available to edit.

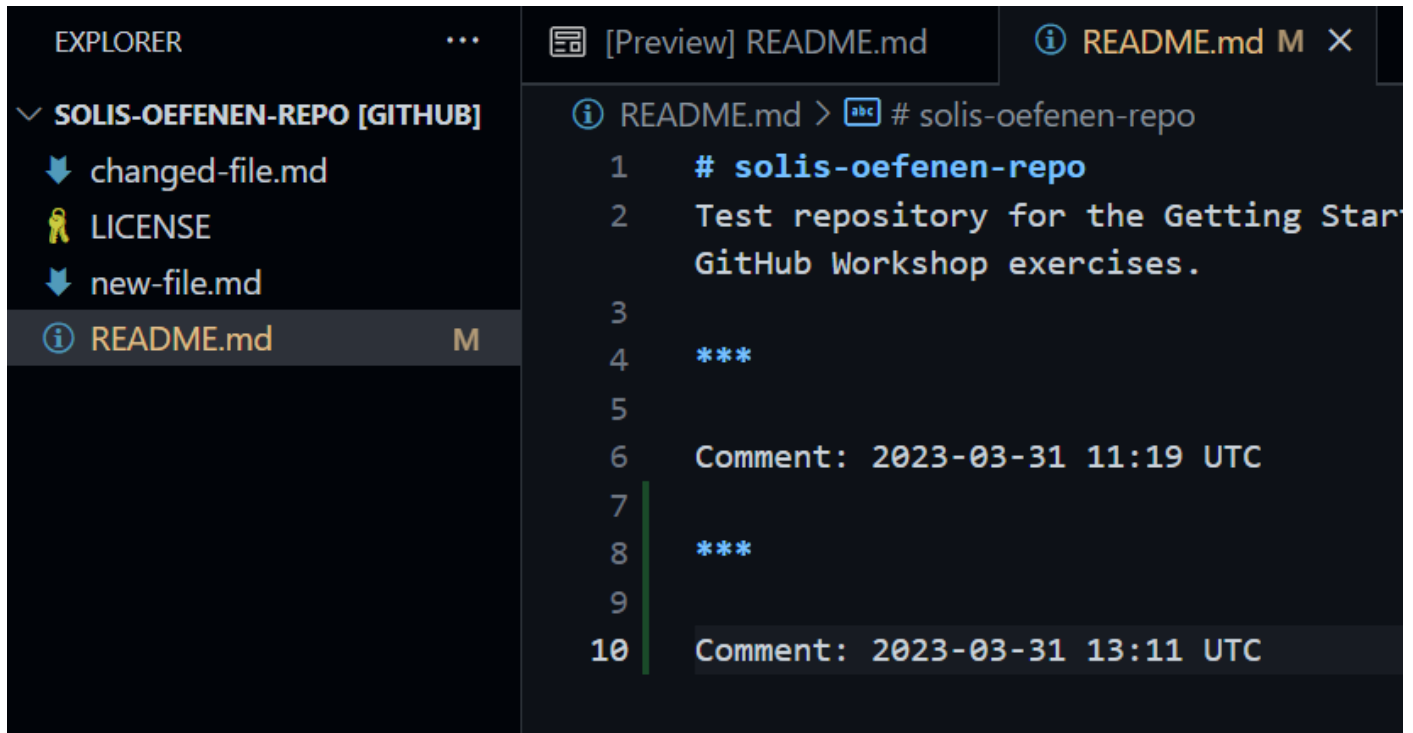


When it finally loads, you can select files and make changes to them, just like on your local computer. Here, we're clicking on the README.md file to add another comment with a timestamp.



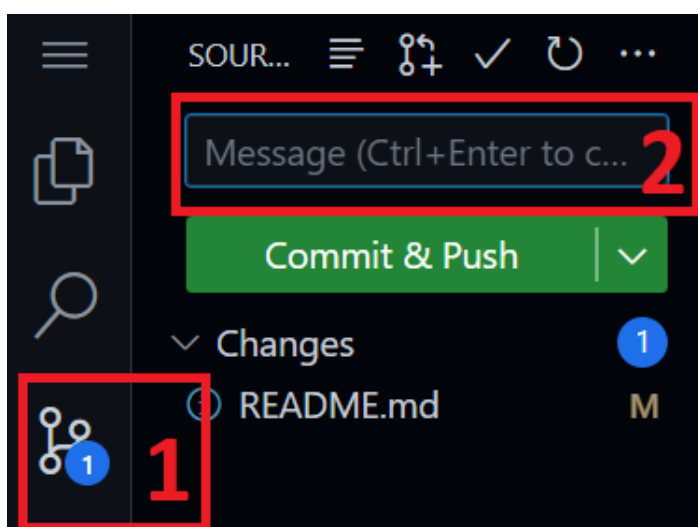
Now, we add a comment similar to the one made in the previous chapter, a short statement with a time stamp.

In the web version of VS Code, you cannot save files; you either need to commit your changes, or they will be lost if you close the window.

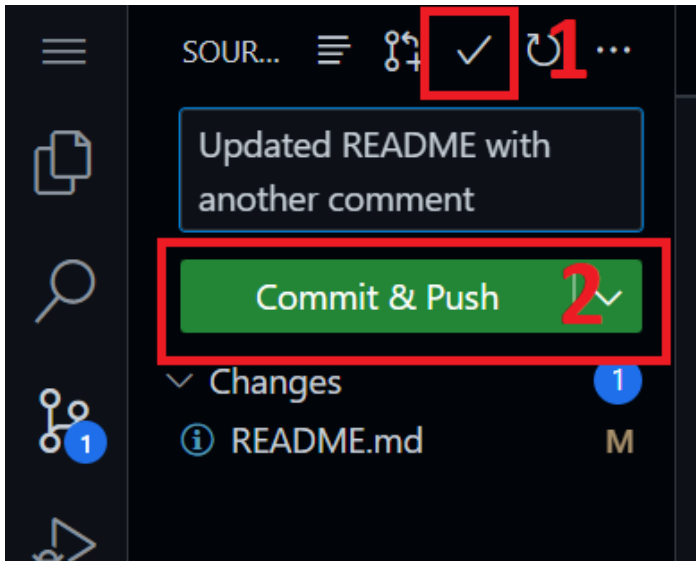


After making the change to your file, click the Source Control button (1) to commit the change to the repository. The source control button should have a number in a blue circle, indicating the number of files that have been changed within this environment.

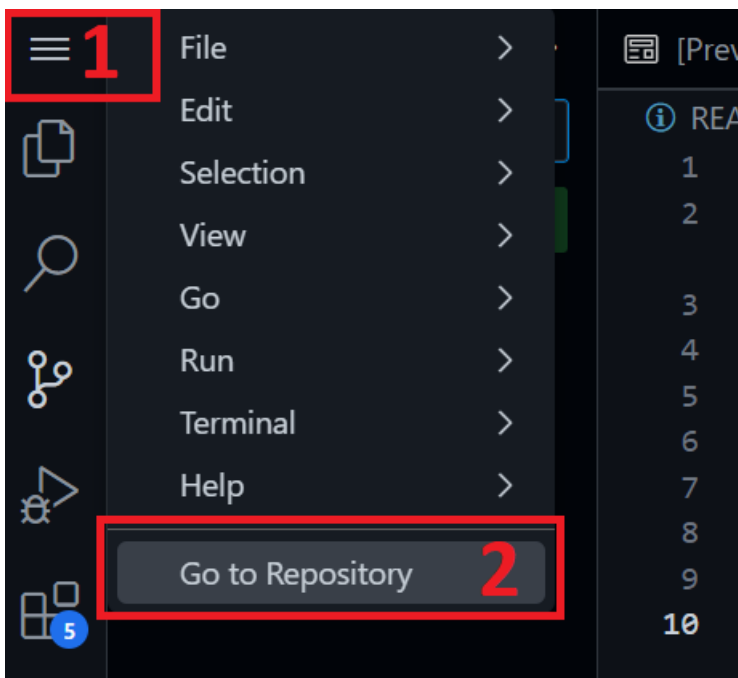
(2) is where you can write a commit message for the changes you have made.



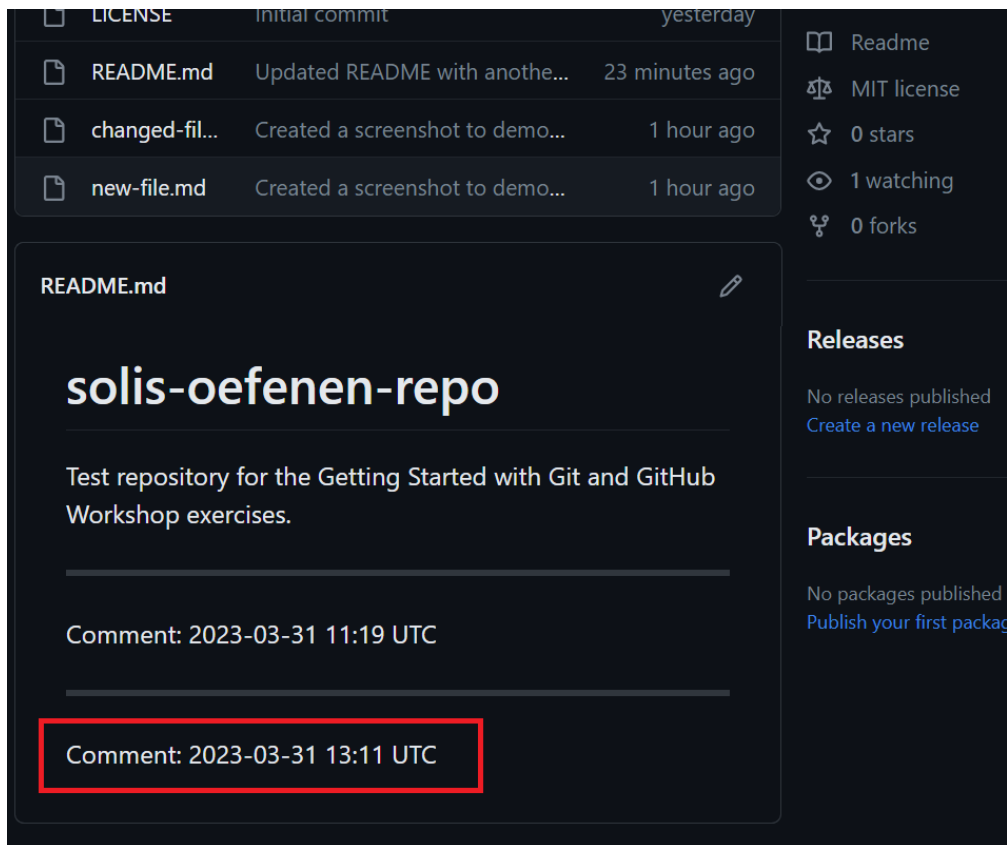
After writing your commit message, you have two options to push the commit to the repository. You can use the check mark (1) or the green “Commit & Push” button (2).



Now the change has been made to your repository and will be visible on GitHub.com. To get back to GitHub.com, click on the hamburger menu icon in the top left corner (1), and then click the option to Go to Repository (2).



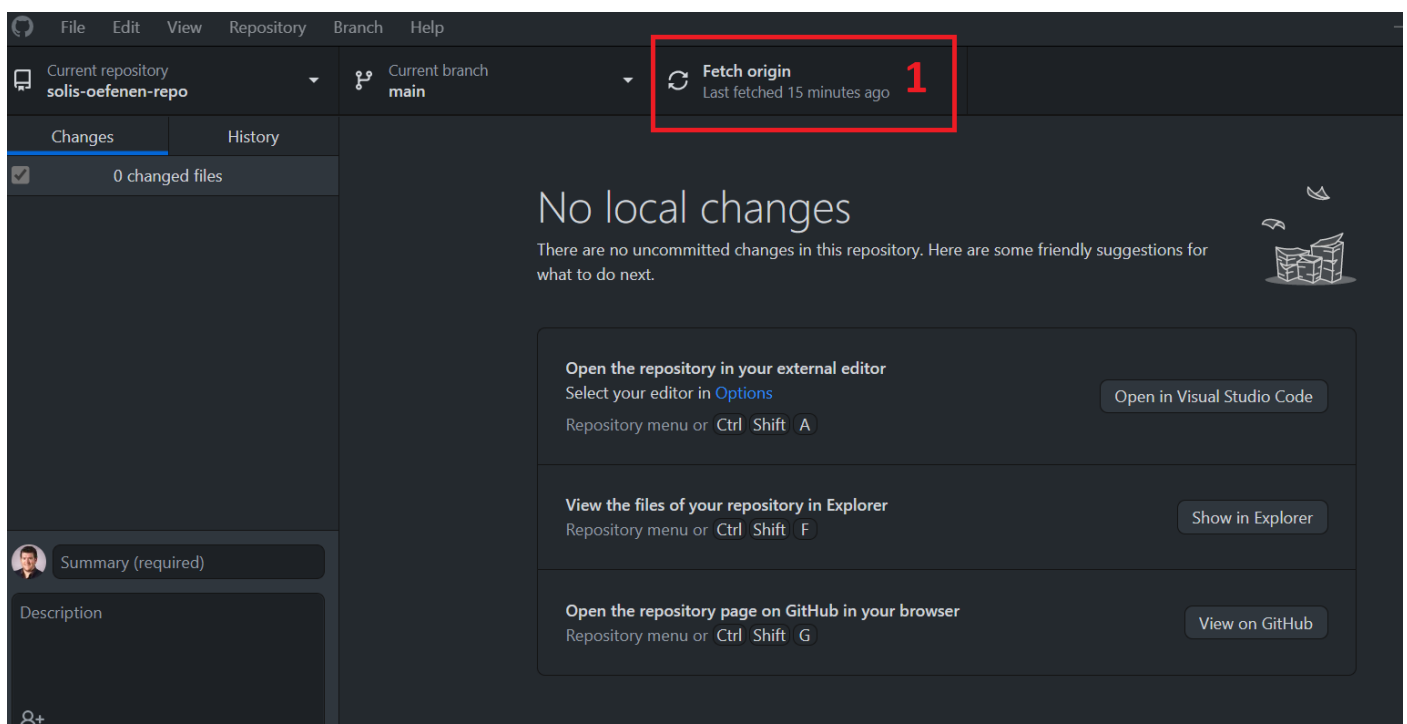
After that, you should be able to see the change made on GitHub.com



Pulling Remote Changes in GitHub Desktop

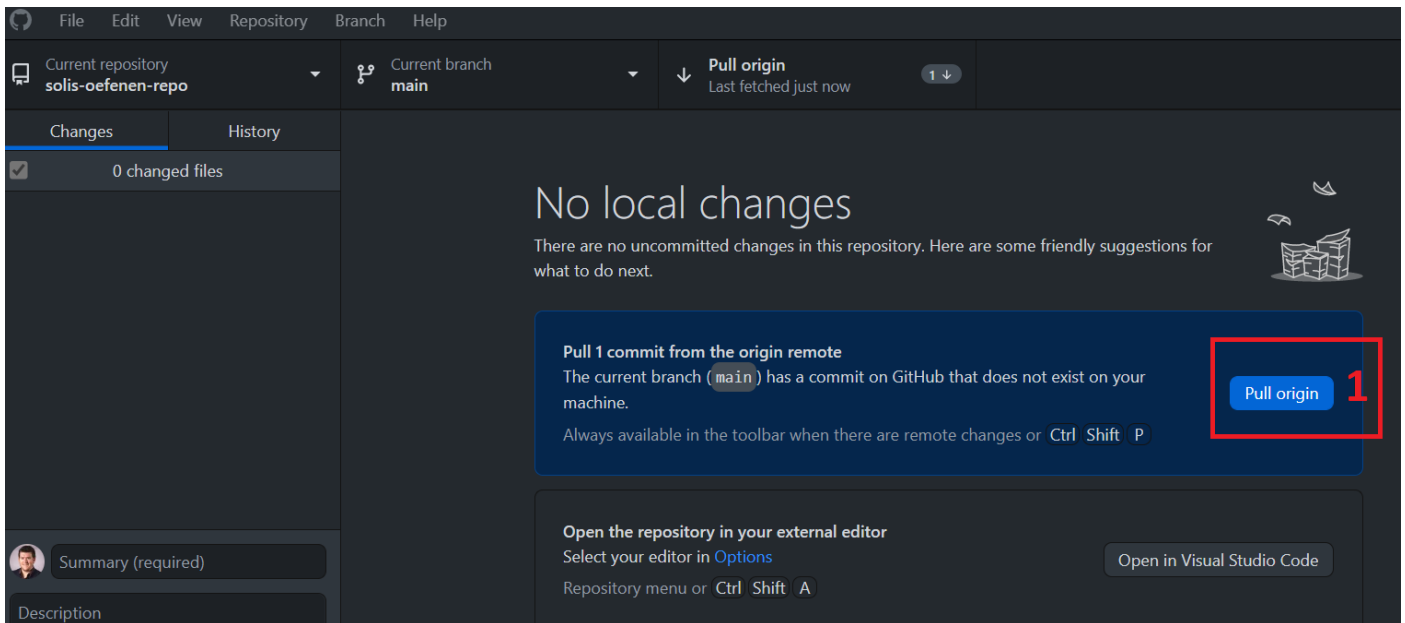
In your local copy of the repository, you are probably not seeing the edit that was made in the GitHub.com version. Go back to your repository in GitHub Desktop: it should look unchanged like below. If the application has had the chance to update on its own, or you closed and then re-opened the program, continue on to the next screenshot.

So how do we get the changes from GitHub.com? Press the Fetch Origin button (1).

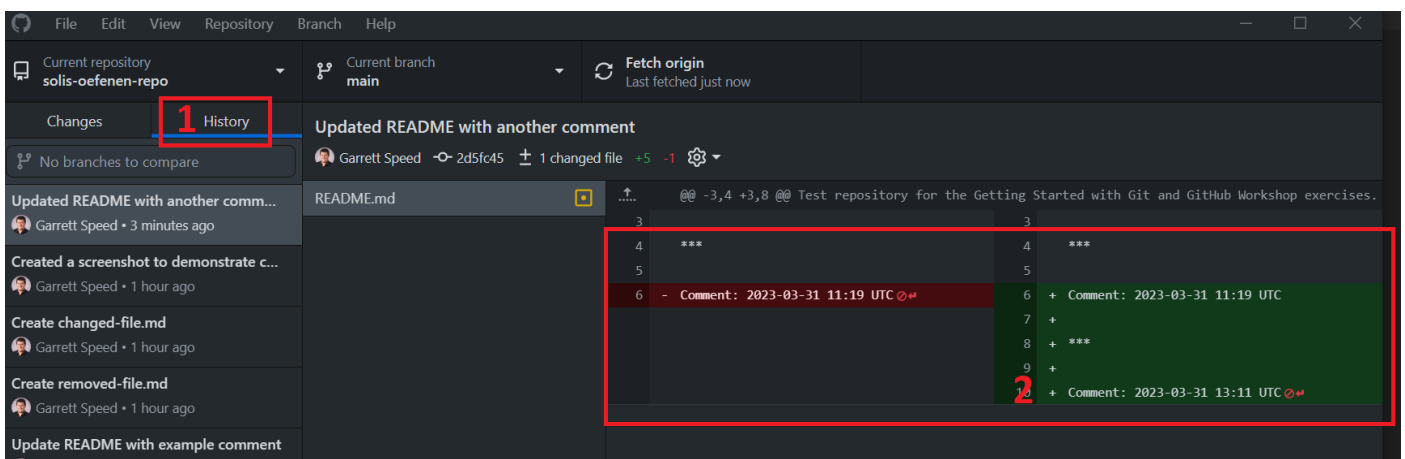


If there are changes on the remote server, a blue box should appear to pull the changes down and update your local files. To do so, click the “Pull origin” button (1)

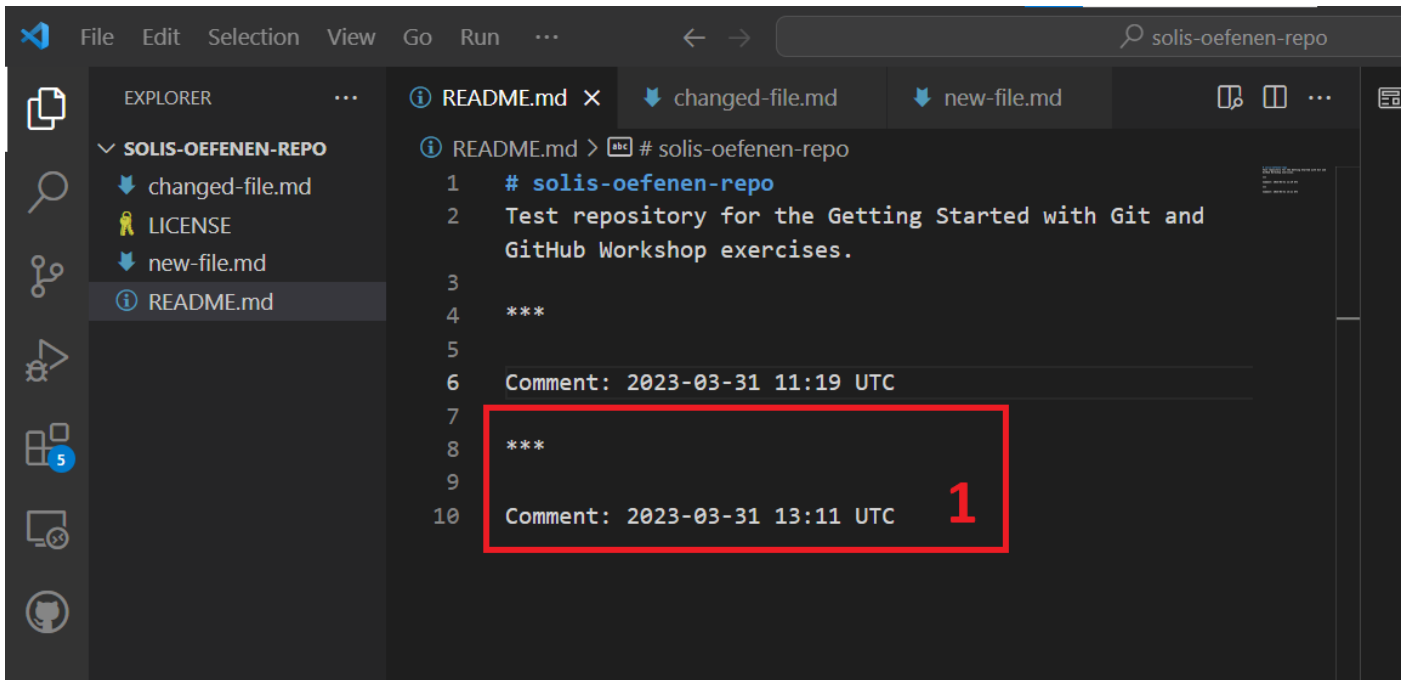
In more advanced uses (e.g. performing merge/pull requests that cannot be auto-resolved), this may fail, but that is out of scope of this manual. Our next workshop, Collaboratively Writing Code with Git and GitHub will address the situations where this may occur.



Once the pull origin button has finished synchronizing (pulling the changes from GitHub.com), you can see the changes (2) in the history tab (1).



The changes (1) are also visible in the integrated code editor connected with the repository, such as in this case with VS Code.



Committing Code Changes

6

The previous chapters of this manual have so far made examples of changes in Markdown files, but the steps of committing and pulling changes remain the same with code file changes too. Remember from the introduction that Git tracks line by line changes in text-based files, not specifically code-based files.

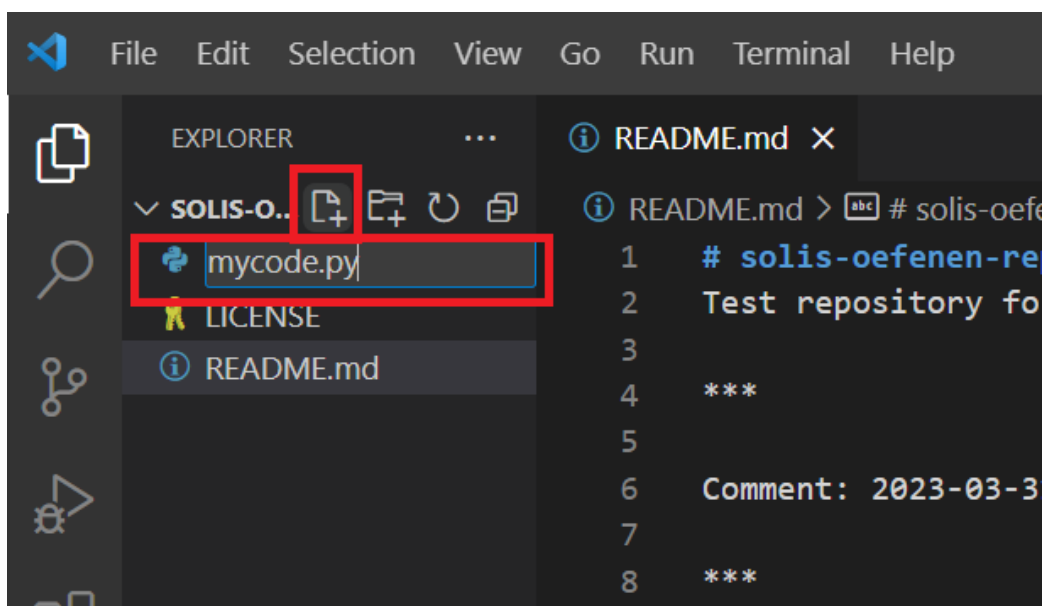
In this chapter we'll look at when you should commit code changes using an example in Python.

In general, the advice is to make a commit when you have made a significant change, such as a new file, new function within the code, or corrections to existing code.

This chapter uses Python in the examples, and one of the examples uses the term numeric object. Numeric objects include integers, floating point numbers, and decimals. The terminology in your language may differ, but for simplicity, just think about standard number variables in your programming language.

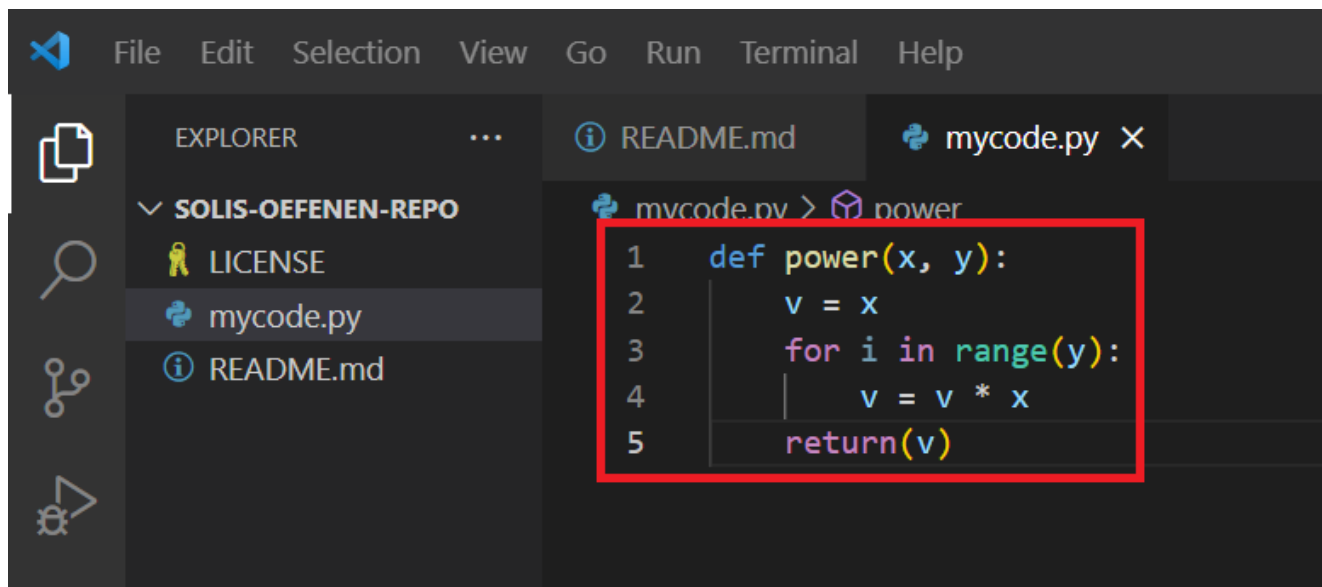
New Code File and First Functions

First, we can create a new file, I will call mine "mycode.py" and create it from within my code editor.



At this point, I might commit this change as the creation of the file if I wanted the file to exist, but was not ready to start placing code in it. I would write my commit message as “Created file mycode.py”. This might be helpful for code that has been planned for, but development is not happening right now. If I am going to start writing code in the file, I would probably not commit the file creation commit change to Git yet.

Now, I’m going to start writing code in mycode.py. I make my first function, a power function to multiply one number (x) a number of times (y) to itself. I test running it, it returns the expected results.

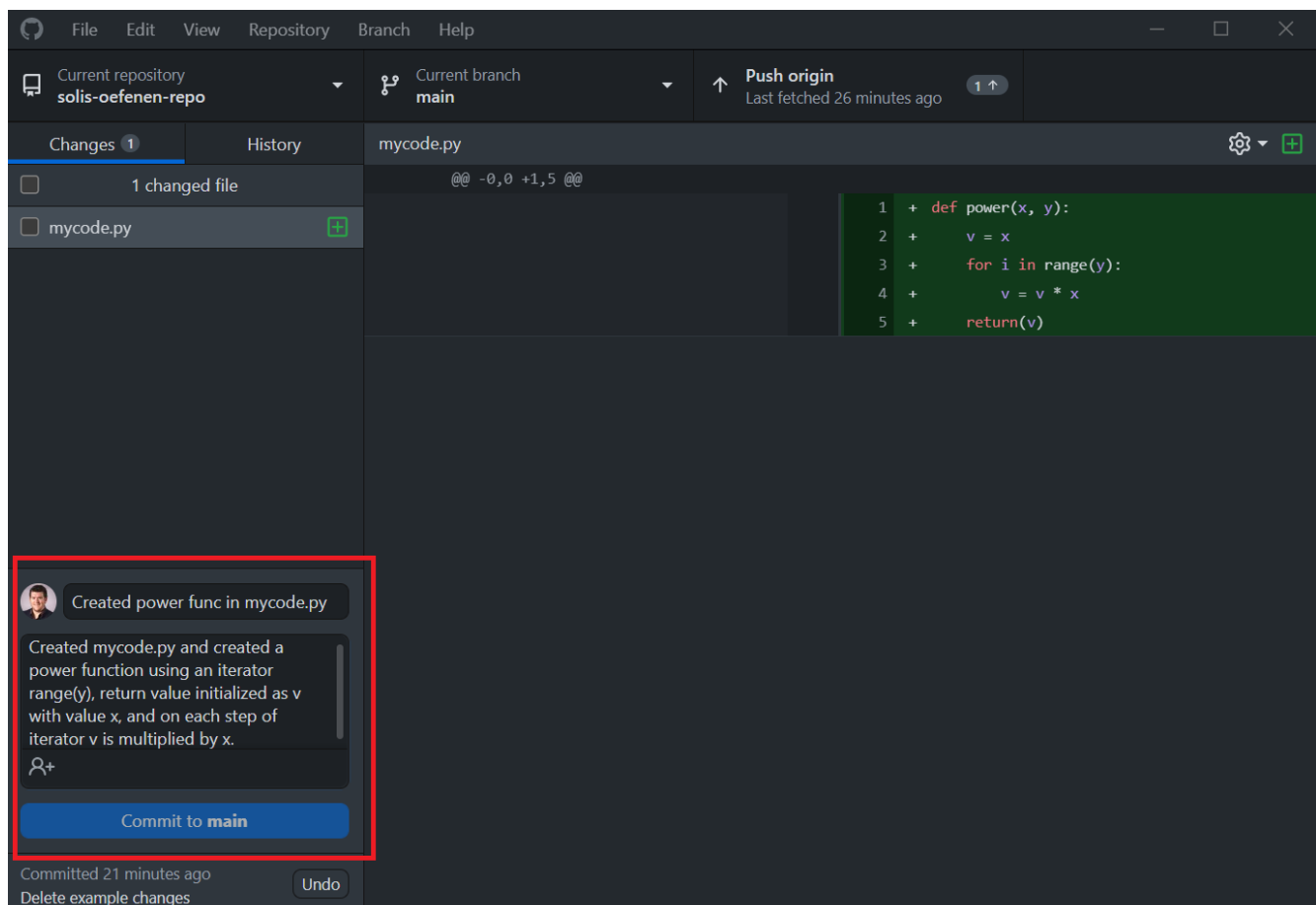


The screenshot shows the Visual Studio Code editor interface. The Explorer sidebar on the left shows the project structure for 'SOLIS-OEFENEN-REPO', including 'LICENSE', 'mycode.py', and 'README.md'. The main editor window displays the 'mycode.py' file with the following Python code:

```
1 def power(x, y):
2     v = x
3     for i in range(y):
4         v = v * x
5     return(v)
```

The code is highlighted with a red rectangular box.

I would now commit my change to Git with a message “Created power func in mycode.py” along with a possible description of “Created power function with a range iterator initializing v from x, then on each step of the iterator multiplying v with x”.

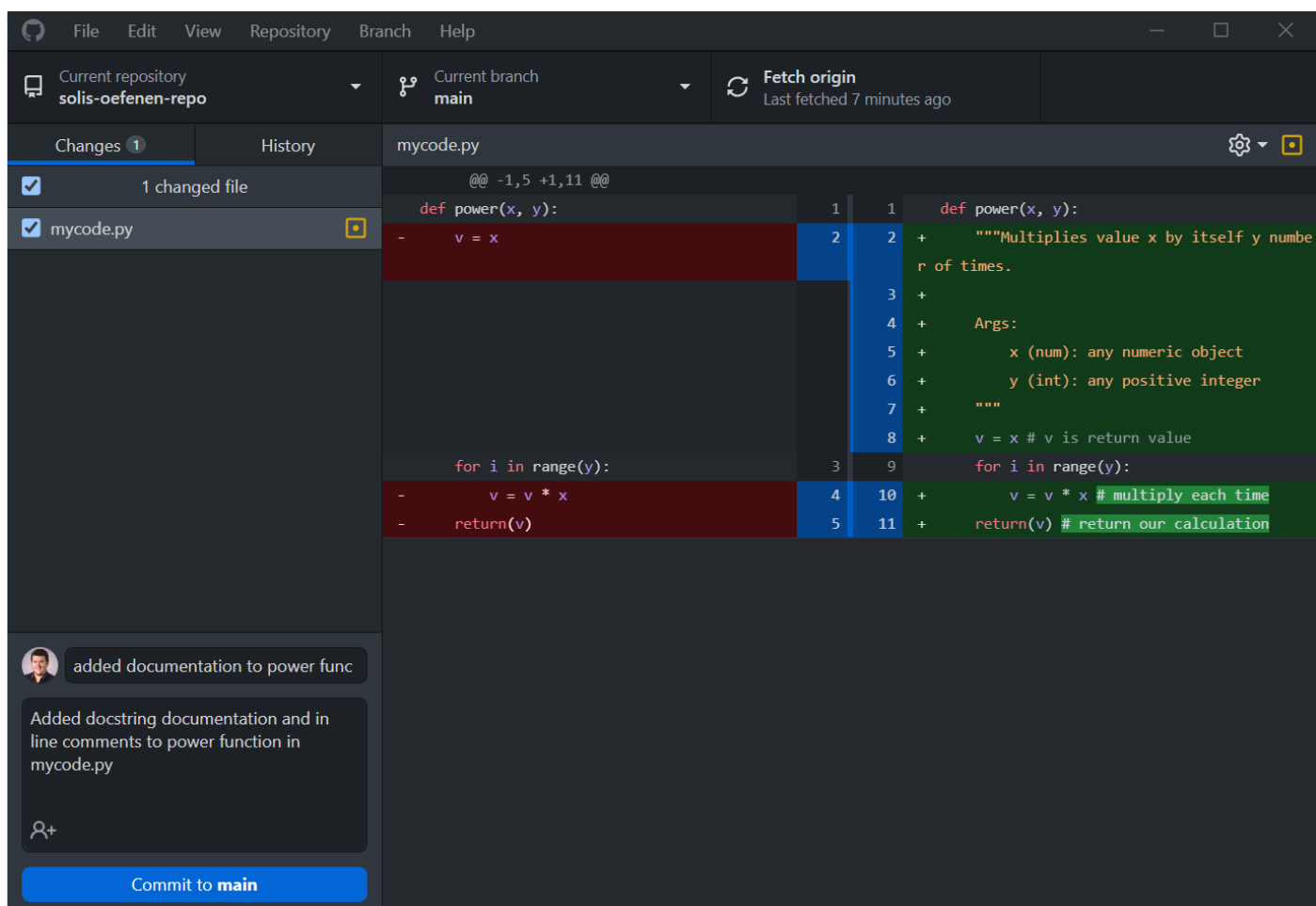


Documenting Code

After I've gotten my first function working, I want to add some documentation to that code through docstring convention and inline comments. Consider the documentation added here:

```
mycode.py > ...
1  def power(x, y):
2      """Multiplies value x by itself y number of times.
3
4      Args:
5          x (num): any numeric object
6          y (int): any positive integer
7      """
8      v = x # v is return value
9      for i in range(y):
10         v = v * x # multiply each time
11     return(v) # return our calculation
12
```

I would make my commit to Git when I make all of my documentation changes to my code, not at each step of making the docstring or the inline comments. Look at my commit message and description in GitHub Desktop here:



Fixing and/or Enhancing Existing Functions

This section may be considered opinionated based on the approach of the authors to programming.

The keen eyed Pythonistas might have noticed my power function is quite inefficient and limits actual functionality.

I have now realized the problems with my code, and I have also come up with some restraints I want to make on my code:

- Replace calculation method with built-in `x**y` notation
- Update documentation
- Raise error if `y` is not a positive number

I'm actually going to make these changes in two steps, not one.

First I am going to update the calculation method to use the built in, and update the documentation accordingly:

```
mycode.py > ...
1  def power(x, y):
2      """Multiplies value x by itself y number of times.
3
4      Args:
5          x (num): any numeric object
6          y (num): any numeric object
7      """
8      v = x ** y # v is return value
9
10     return(v) # return our calculation
11
```

And the commit itself:

Changes 1 | History | mycode.py

1 changed file

mycode.py

File	Line	Old	Diff	New
mycode.py	3	def power(x, y):		def power(x, y):
	4			Args:
	5	x (num): any numeric object		x (num): any numeric object
	6	y (int): any positive integer	-	y (num): any numeric integer
	7	"""		"""
	8	v = x # v is return value	-	v = x ** y # v is return value
	9	for i in range(y):	-	
	10	v = v * x # multiply each time	-	
	11	return(v) # return our calculation	-	return(v) # return our calculation

Update power with built-in

Update power func in mycode.py with built-in power calculation.

Commit to main

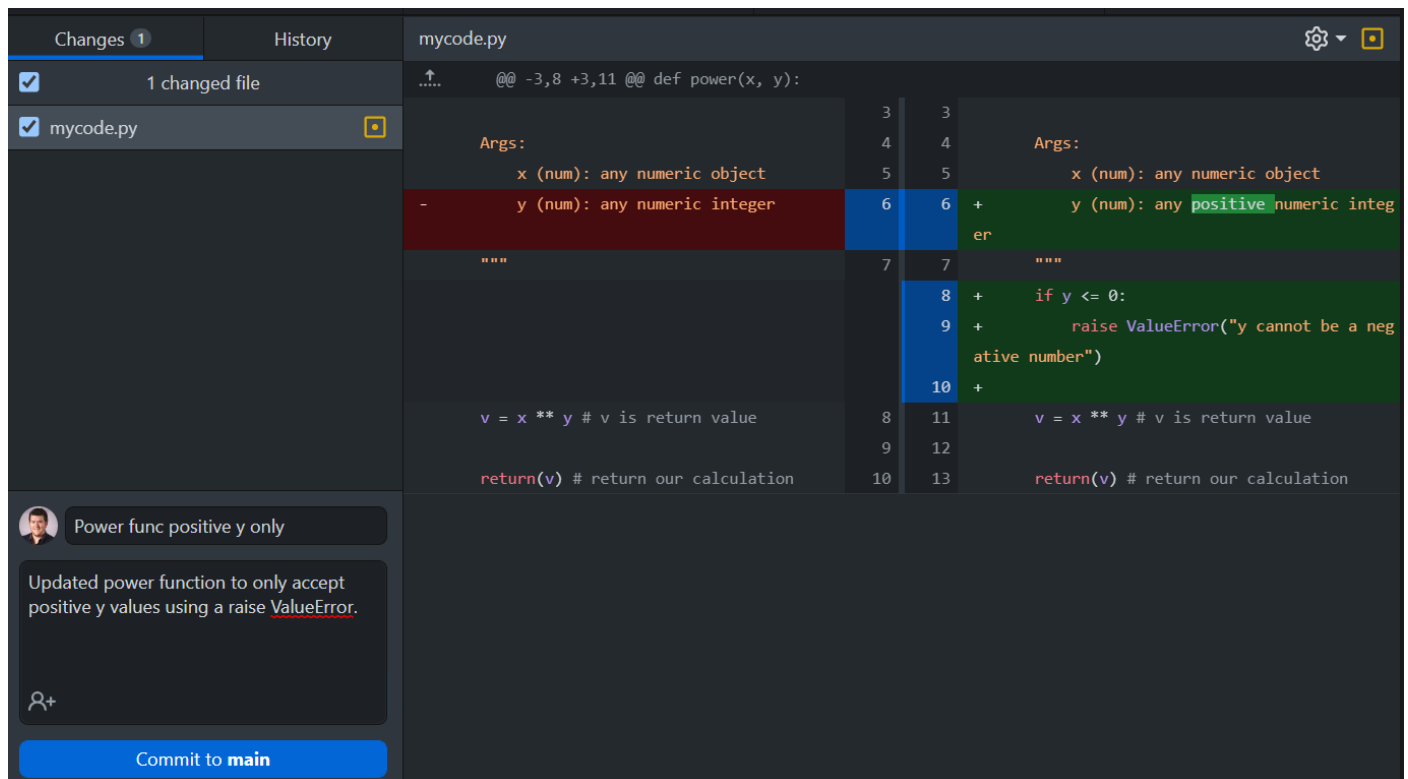
Committed 2 minutes ago
added documentation to power func

Undo

Since I have committed the first change, I now want to implement a change to force the function to only accept positive y values.

```
mycode.py > power
1 def power(x, y):
2     """Multiplies value x by itself y number of times.
3
4     Args:
5         x (num): any numeric object
6         y (num): any positive numeric integer
7     """
8     if y <= 0:
9         raise ValueError("y cannot be a negative number")
10
11     v = x ** y # v is return value
12
13     return(v) # return our calculation
14
```

And the commit message, description, and changes in GitHub Desktop. You can see in the change section that I kept accurate documentation at each point with my changes. The built in function can have negative or positive x or y values, so in the function update previously I documented that any numeric object is possible. Only now that I have added the `ValueError`, y can no longer be any numeric object, it must be a positive numeric object, and I reflect that in my documentation.



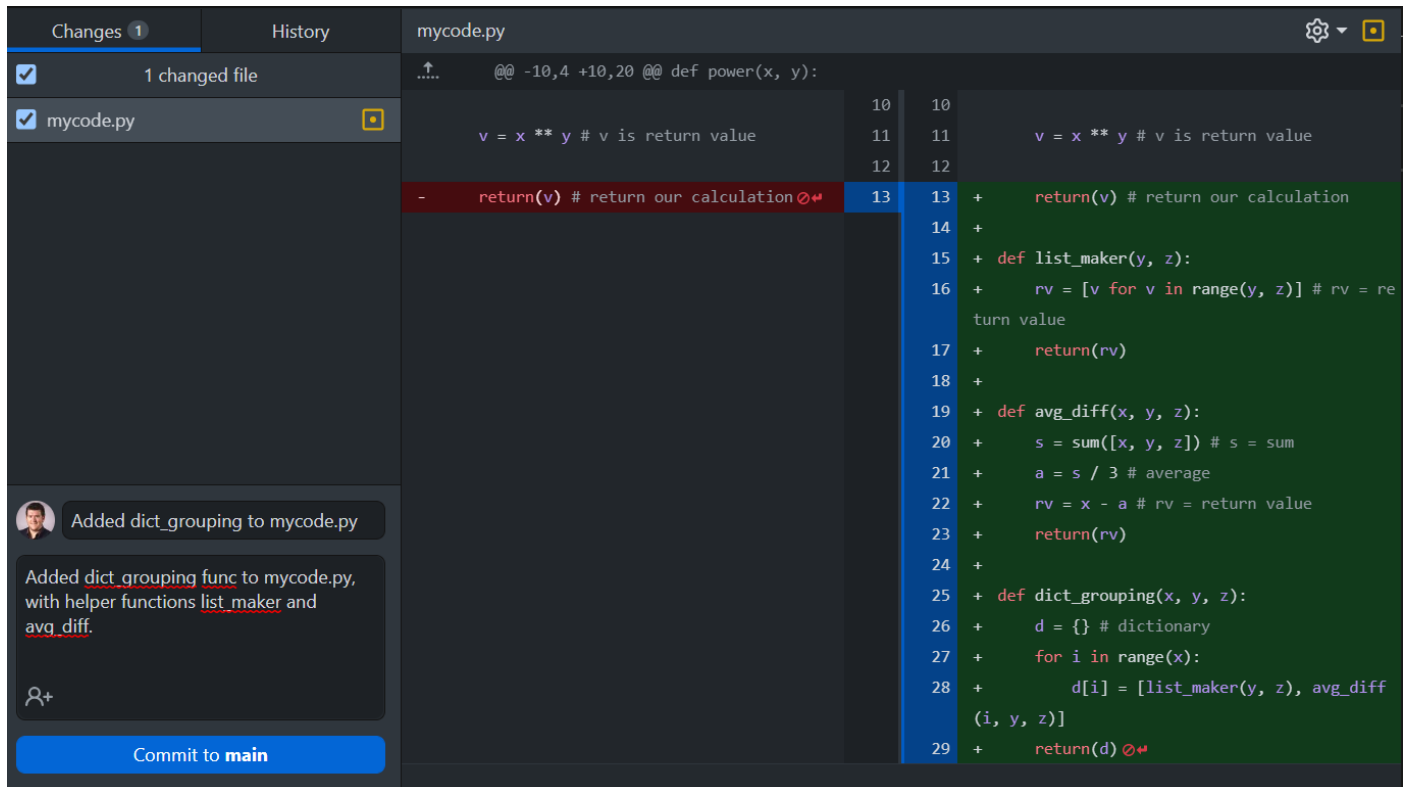
Making Multiple Related Changes for a Group of Functions

You might not always want to commit every function when developing code as you may be making many small functions to help in a larger process.

In the following example, I made three new functions, `dict_grouping` relies on the functions `list_maker` and `avg_diff`. The small functions, `list_maker` and `avg_diff` are quick to write and don't have a lot of functionality but could be useful later. The main process is `dict_grouping`, and is the focus of the current development effort.

```
mycode.py > dict_grouping
15 def list_maker(y, z):
16     rv = [v for v in range(y, z)] # rv = return value
17     return(rv)
18
19 def avg_diff(x, y, z):
20     s = sum([x, y, z]) # s = sum
21     a = s / 3 # average
22     rv = x - a # rv = return value
23     return(rv)
24
25 def dict_grouping(x, y, z):
26     d = {} # dictionary
27     for i in range(x):
28         d[i] = [list_maker(y, z), avg_diff(i, y, z)]
29     return(d)
```

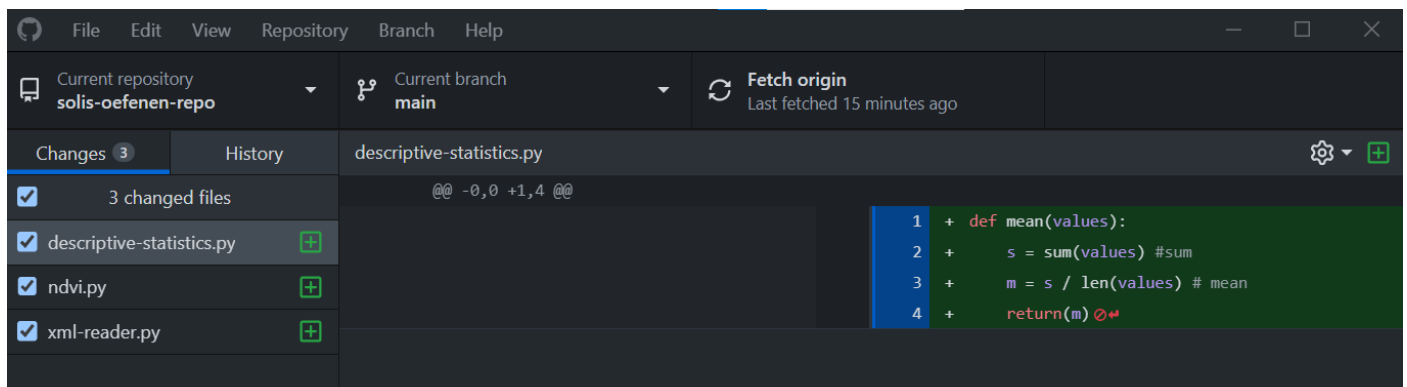
In my commit message, I am going to mention creating the `dict_grouping` function, and in my commit description I will mention the helper `avg_diff` and `list_maker` functions.



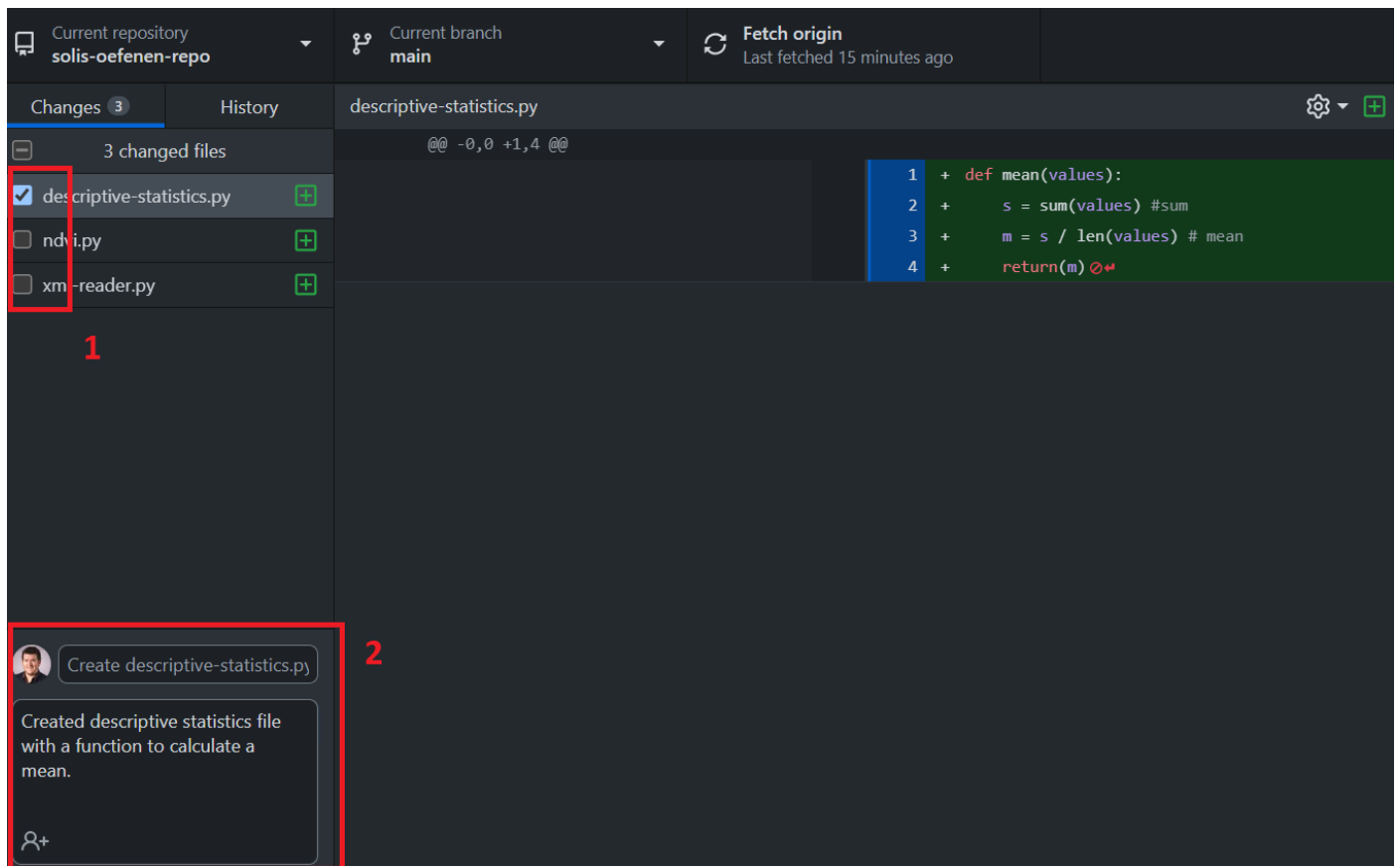
Making Multiple Unrelated Changes for a Group of Functions

Sometimes you're editing multiple files and you have a bunch of unrelated changes in each one. As long as the changes are in different files you can use the file change panel to commit changes made to each file separately.

We'll take a look at another example with some Python code. In this example, I created three basic python files, one for an NDVI ([Normalized Difference Vegetation Index](#)) function for remote sensing, one for calculating a statistical mean, and another for opening an XML file. Now my GitHub Desktop looks like this:



In order to track my changes one file at a time, I will deselect the other files one at a time in the file changes area of GitHub Desktop (1). Then I can write my commit message and description for the single file I am committing changes in.

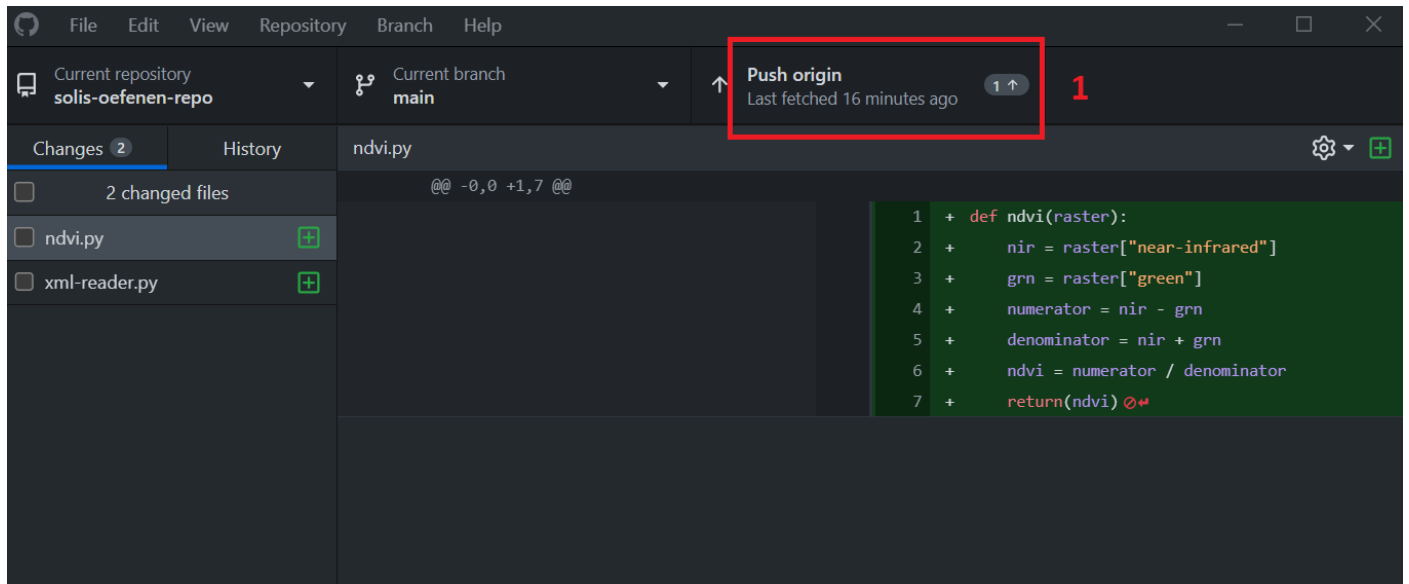


After I click commit, my descriptive-statistics.py file has disappeared from the changed file section, but the other two files remain.

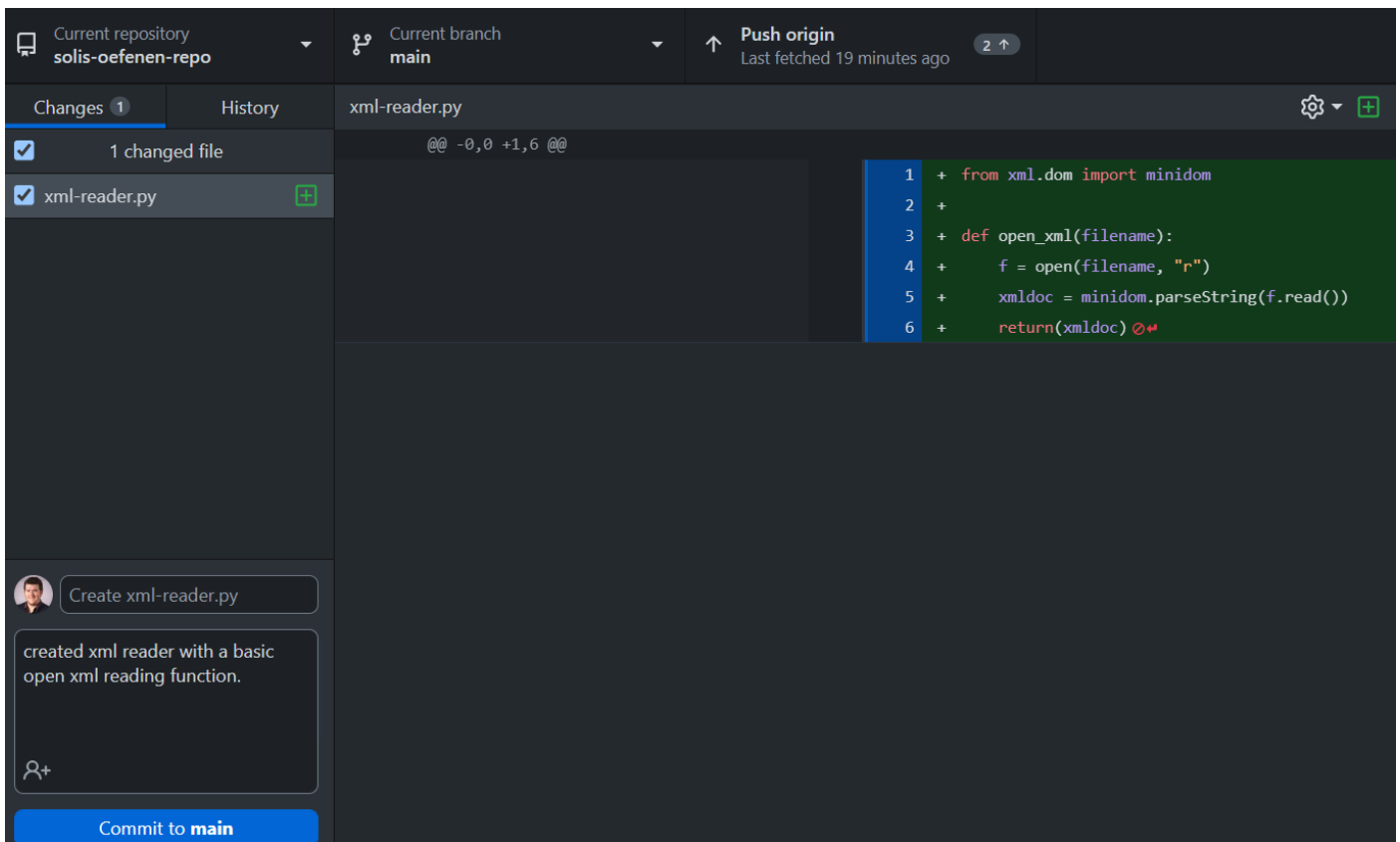
Note, that the files that remain do not automatically get re-checked for my next commit, so I will need to select the next file for committing.

Also note that GitHub Desktop did not revert to the unchanged screen where it will ask you to push the changes to origin/remote (GitHub.com). If you wish to push to origin/remote between each commit, you can click the Push origin button (1).

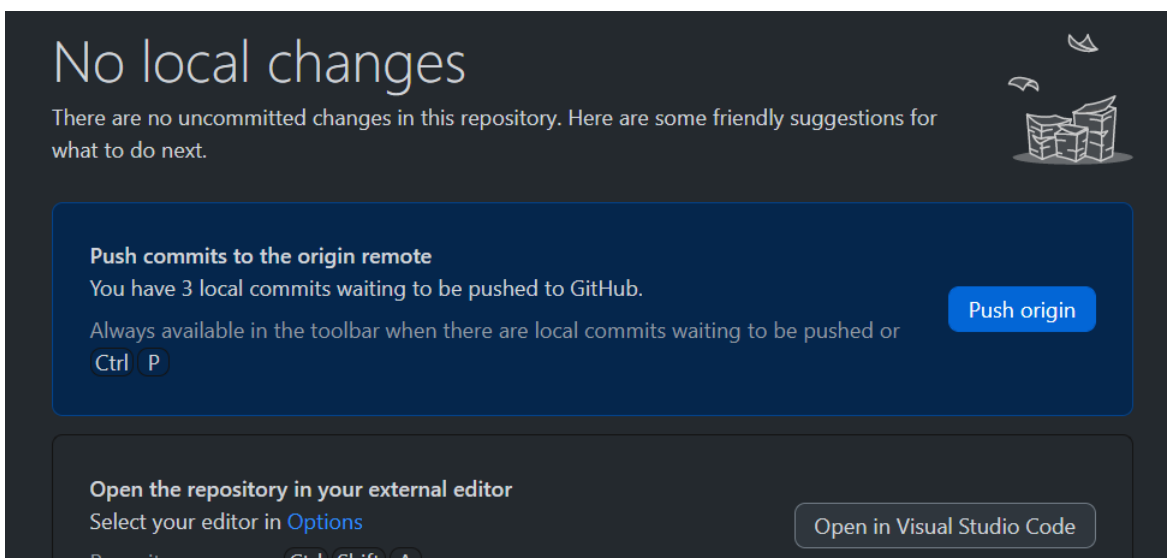
Again, add the commit message and description, and click commit.



After committing, you can then enable the last file, add the commit message and description, and commit.



Now that there are no more pending changes, GitHub Desktop will return to the no-pending changes view and have the blue box for Pushing origin.



Other Times to Commit Changes

In case of sudden incidents, instead of just leaving your code changes on your local machines, it is always advisable to commit the changes as they are and push them to the remote repository. Especially if an unplanned event occurs that could have consequences for device loss, consider the examples below:

- Power outage, mobile hotspot your computer to your phones internet connection, and commit "power outage, safety push".
- Fire alarm goes off, write a commit message "Fire Alarm, safety push"
- Train about to arrive and you need to alight, "Safety push, getting off train"

7

.gitignore: How to Avoid Leaking Sensitive Data

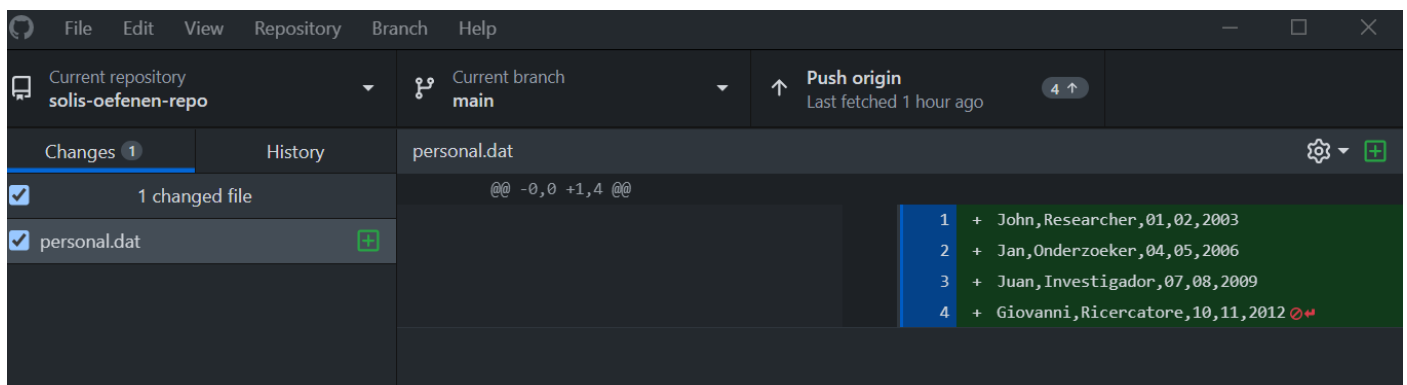
When developing code, you might need to avoid some data being tracked by Git and from possibly being posted on a web repository (e.g. GitHub). For example, you may not want to put sensitive data, passwords/access tokens, or large files (>50 MB) on GitHub. Generally speaking, it is advised to use GitHub for code and other repositories for Research Data. Small unsensitive data, such as a fake dataset for testing or demonstrating purposes, may be added to GitHub. To avoid certain files that are in your local repository to be added to GitHub is by using .gitignore files; a special file that tells Git to ignore files or entire folders with set names or name patterns. This way, anything listed in the .gitignore file will not be tracked by git, and thus not be version-controlled or uploaded to GitHub.

The structure of the file is very simple, it is a text file with a list of file names or name patterns to avoid.

First .gitignore

Let's consider a situation where I am working with sensitive personal data, in a file called "personal.dat". This file contains names and birthdays, and it would be a leak of personal data if it got published on GitHub with my research code.

After adding this file, we see personal.dat in GitHub Desktop.

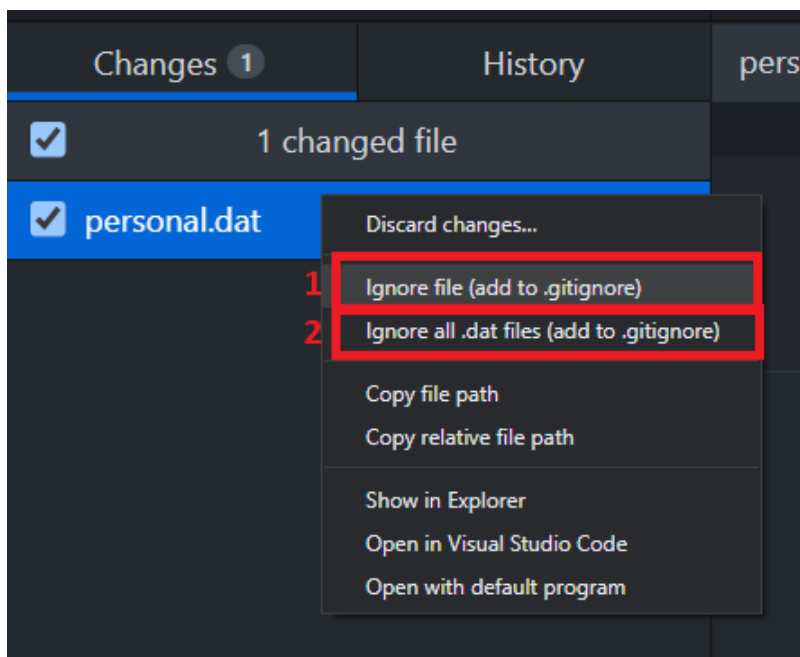


GitHub Desktop makes it easy to ignore this file, you can right click on the file and select either "Ignore File" (1) which will ignore this specific file by its filename.

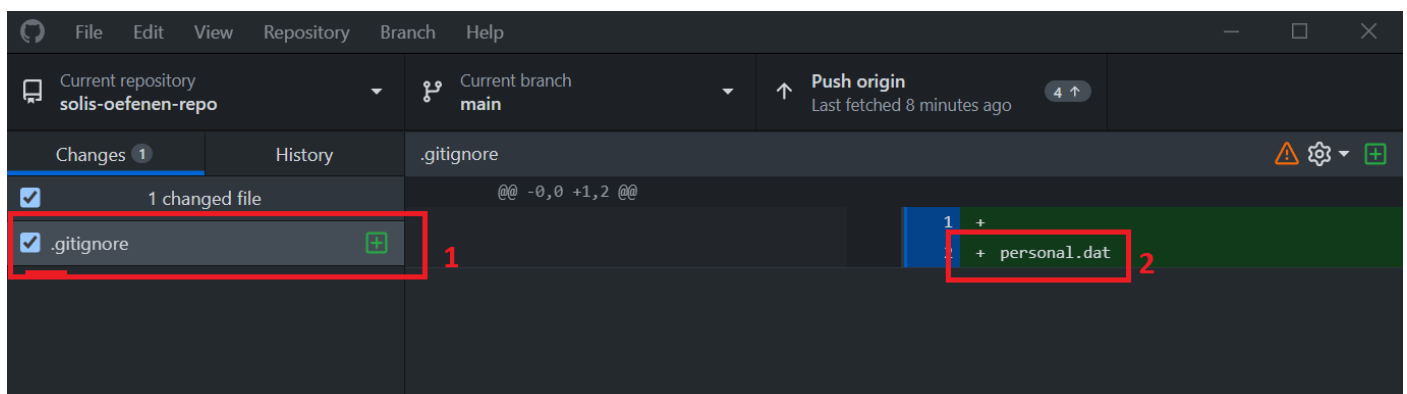
You can also select "Ignore all .dat files" (2) which will tell Git to ignore all .dat files. If this were another file extension, such as .csv, GitHub Desktop would say "Ignore all .csv files".

In chapter 1, during the repository creation process, we mentioned initializing your repository with a .gitignore file, and that some languages have templates available, and some do not. These templates will ignore many common items that are generated when running and developing code, but are only useful or relevant to that local computer; such as logs, checkpoints, and local compiler caches. You can read more about .gitignore templates in the GitHub .gitignore repository.

Your repository may or may not already have a .gitignore file, but if it does not, GitHub Desktop will create one.

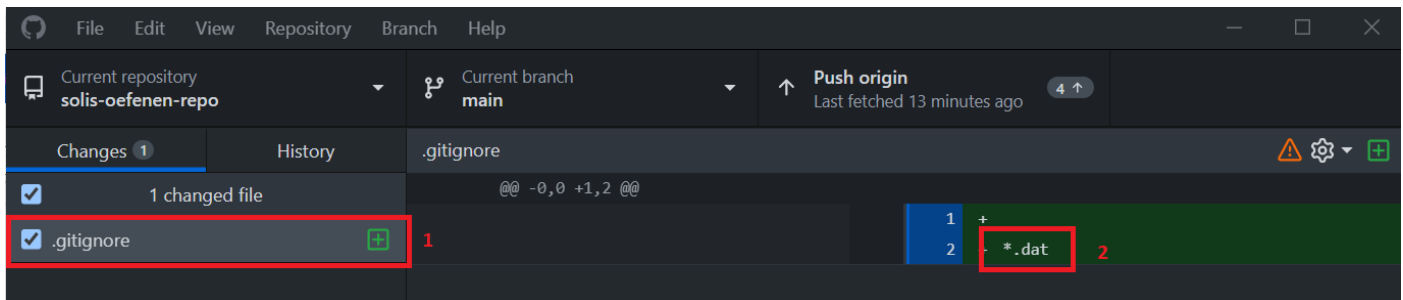


Let's see the changes if we select Ignore File. In the changed files section (1) our personal.dat file has disappeared, and on the right (2) we see there is a new line with personal.dat listed.



What happened? GitHub Desktop created a new file `.gitignore`, and added a new line with the value `personal.dat`, since this repository did not have an existing `.gitignore` file. The Git part of GitHub Desktop scanned the files in the repository, found the `.gitignore` file, and ignored any file with that exact name.

Now let's take a look at what happens if we had selected the Ignore all `.dat` files (and hadn't selected the other option before). Again, GitHub Desktop creates a `.gitignore` file (1), but now adds a line `*.dat` (2). The symbol `*` is known as a wildcard, anything that matches a pattern will get added. We will discuss wildcarding more later in this chapter. In the background, the Git part of GitHub desktop scans the repository, finds the `.gitignore` file, sees that it is being told to ignore anything that matches `*.dat`. The file `personal.dat` matches, and as such is ignored and not available to commit.



Adding to `.gitignore`

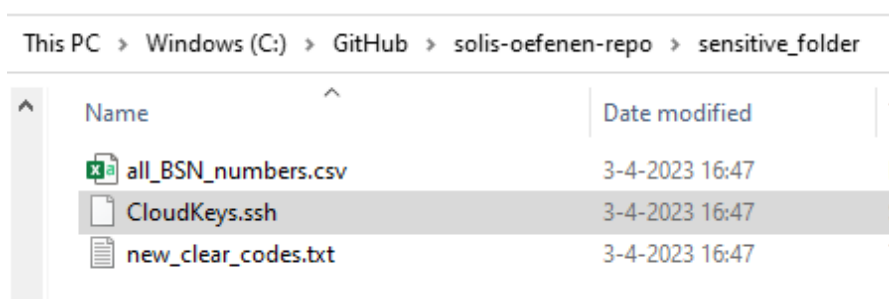
You can continue to use the above methods within GitHub Desktop to expand the `.gitignore` file as you go, or you can edit the `.gitignore` file in your code editor.

To edit, simply open it as you would any code file, and add each filename or pattern as individual lines for them to be ignored.

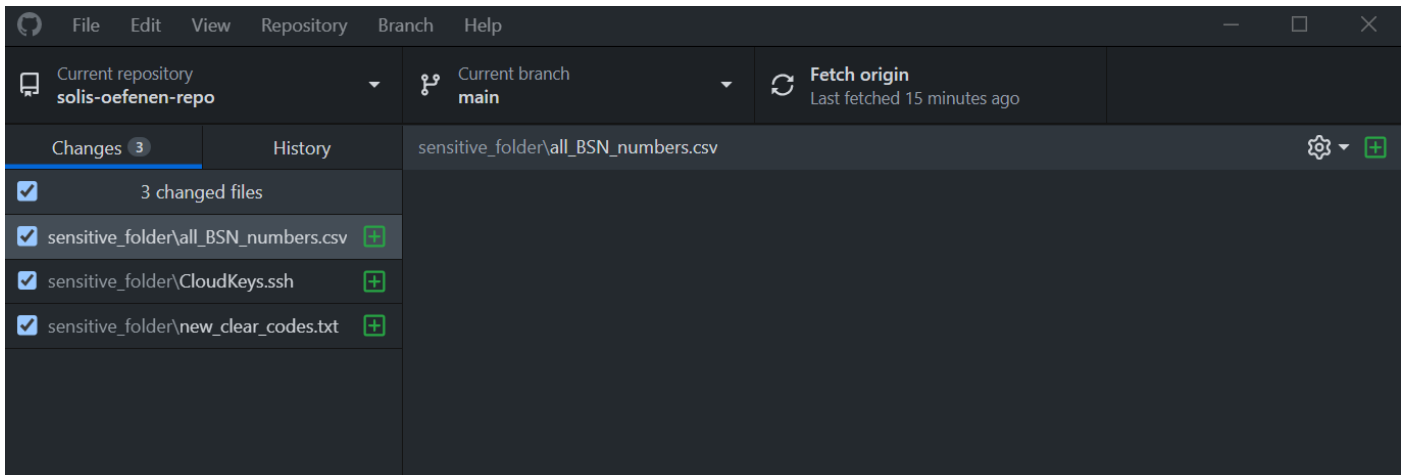
Ignoring a Folder

In some cases, you might have an entire folder of files that you want to ignore. Unfortunately, GitHub Desktop does not have a right click menu to ignore an entire folder. To do so, we need to manually edit our `.gitignore` file in the code editor.

Let's consider a scenario where I have this folder in my repository, and anything else that gets added to this folder is also sensitive and I do not want to be tracked by Git:

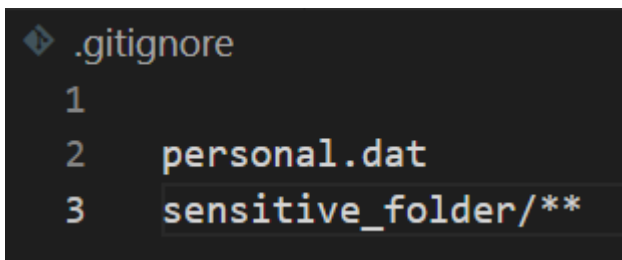


In GitHub Desktop, just after adding them, you should see each sub-file added to the new file list.



I could add each file name one by one to the .gitignore file, but there is a better way.

I can add a simple line to my .gitignore file, "sensitive_folder/**". With this line, Git will scan the repository, find the .gitignore file, see that anything that is in the sensitive folder directory matches, and will subsequently ignore those files. Git also ignores folders entirely if they are empty, or everything contained is ignored.



Now if we check GitHub Desktop, each of those files disappear from the file changes view, and we see only the change to the .gitignore file.

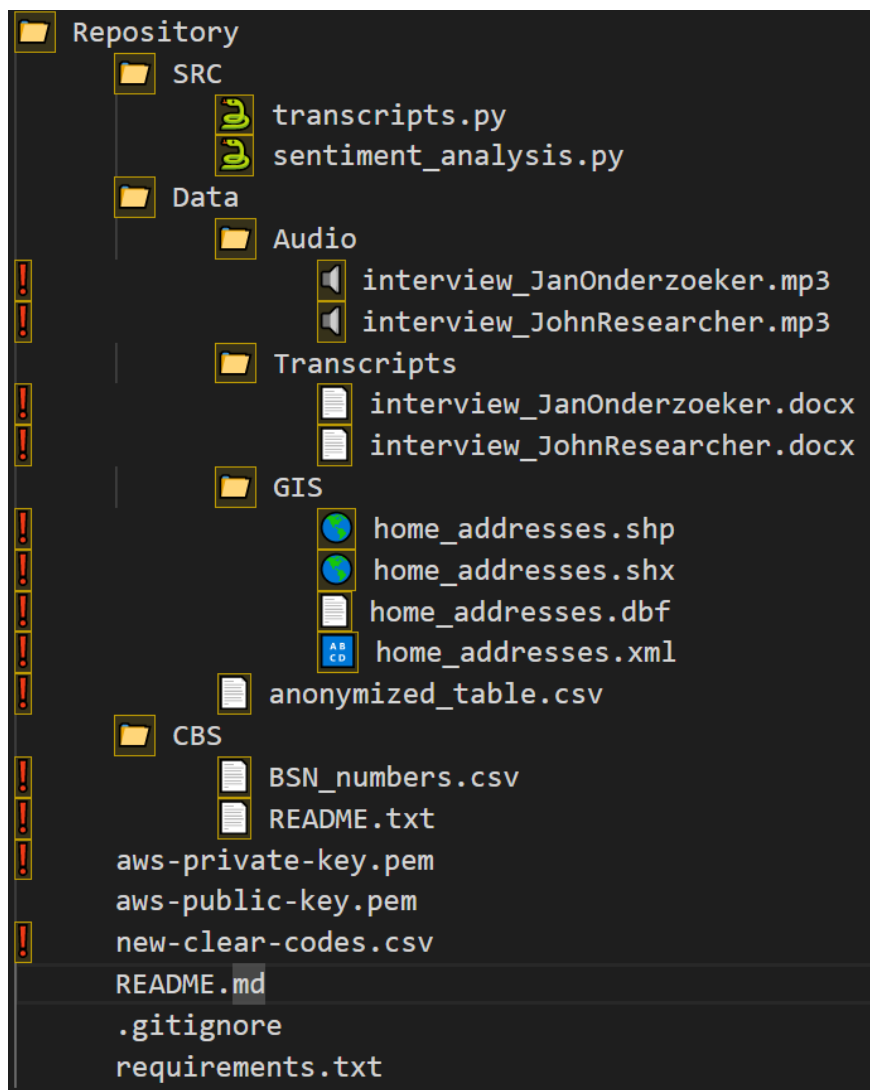
If the files have previously been committed, their data will still show up in the Git history. See chapter 7 on Reverting Changes in GitHub Desktop.

Wildcard (*) Patterns

For this section, you should have file name extensions turned on in your file system, please view the next section for how to turn on file name extensions in your operating system file system.

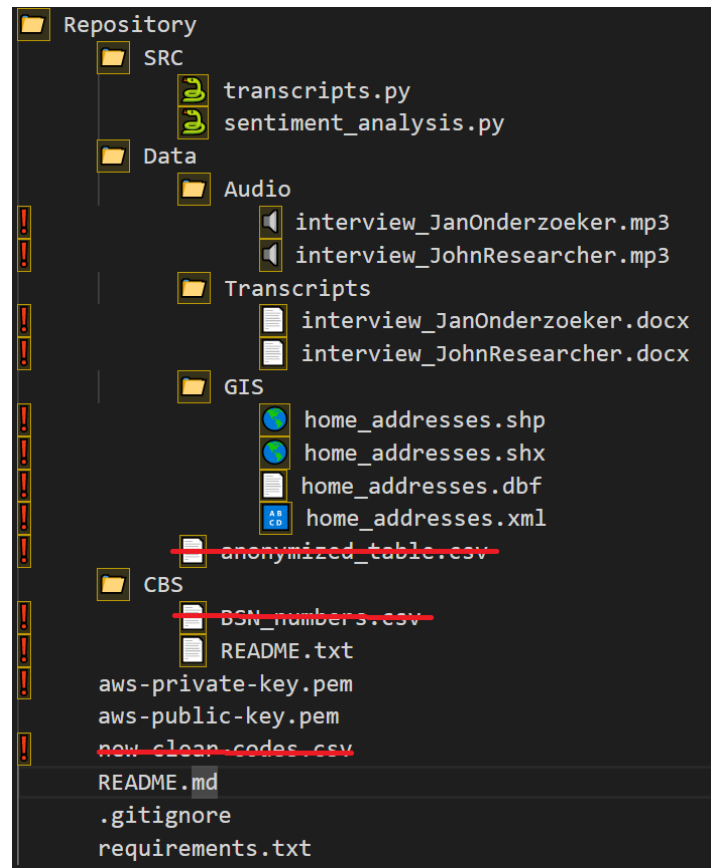
Wildcards are helpful for pattern matching in file names and paths, and can represent any character in a file name (letters, numbers, dots ".", hyphens "-", and underscores "_"). By creating .gitignore records with patterns with wildcards you can avoid listing every individual file to be ignored.

To demonstrate the power of wildcards for the .gitignore file, we're going to look through a couple of common patterns on a folder structure as seen below. The sensitive files in this repository are annotated on the left with a red exclamation mark, in the end we will want to ignore all of these files.

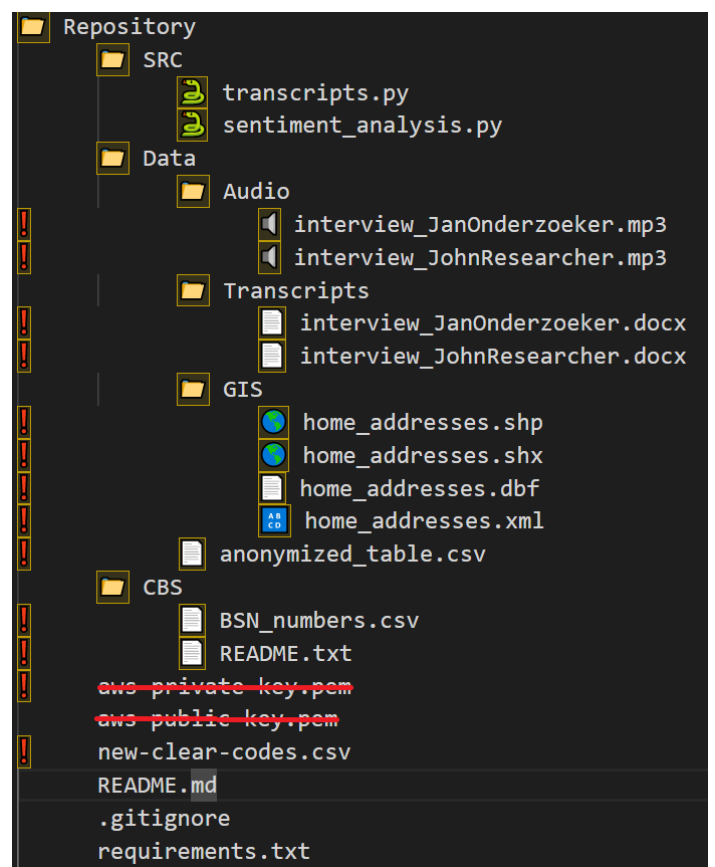


Basic Filetype ignore

To ignore all files of a given file extension, the pattern is simple `*.{file extension}`, so to ignore all of the CSV files we would add a line `*.csv`. See right for which files would then be ignored:



If we wanted to ignore all of the pem files (a cryptographic key file) from our repository, we would write a `*.pem` to our gitignore file, see the results right:

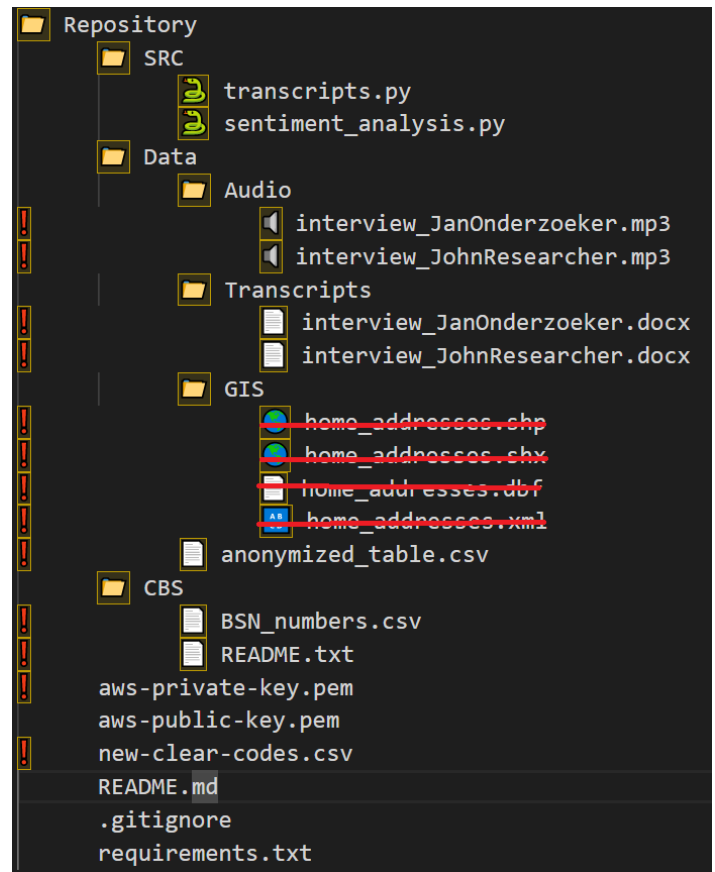


For both of these examples, the dot (.) before the file name extensions is optional, and writing `*csv` and `*pem` would also work. Adding the dot will ensure that only file types are ignored and not filenames that do not have extensions.

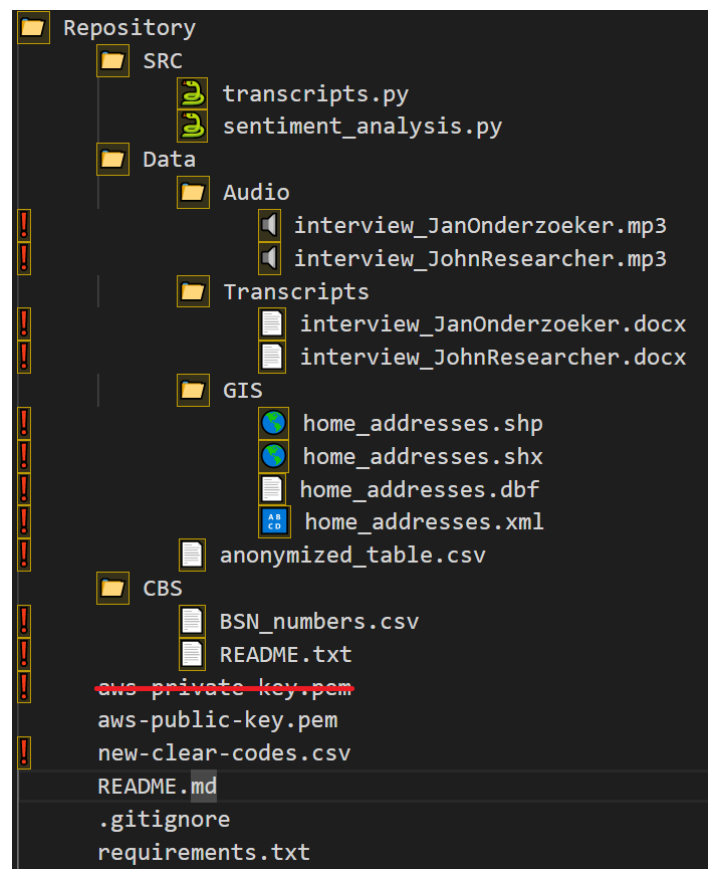
Ignore any file type with a common name

Some datasets or files have a single file name and different file extensions, this is called having side-car files. This is common with metadata files, they will take the same name of the file they represent and use a different extension such as .xml or .json. The basic pattern for this use case is {filename}.*.

In the following example, we want to ignore the shapefiles in the data folder. These files all have the name "home_addresses.{extension}", so we will write our .gitignore wildcard patterns as "home_addresses.*":



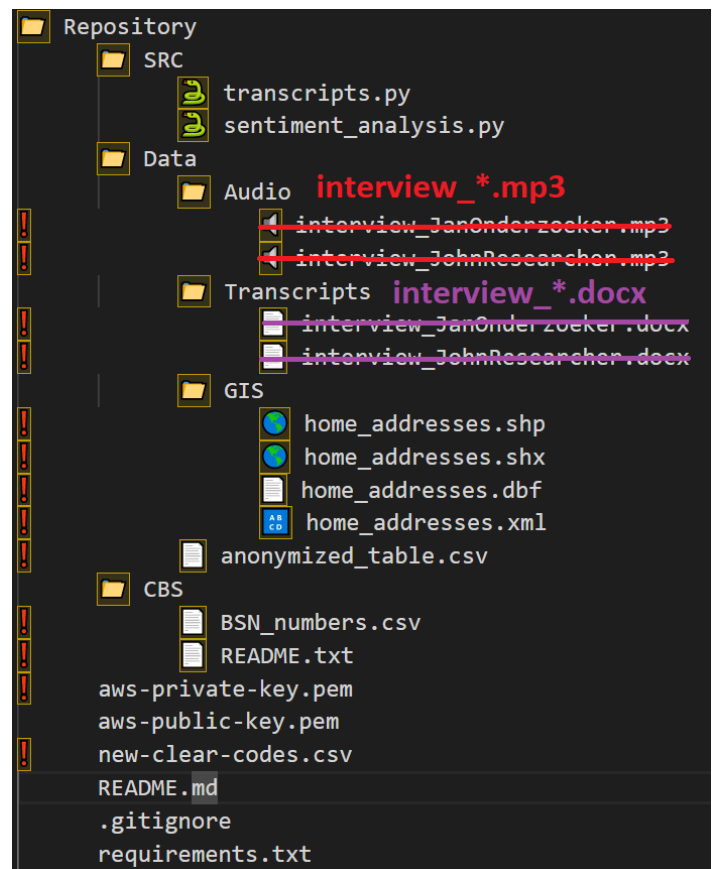
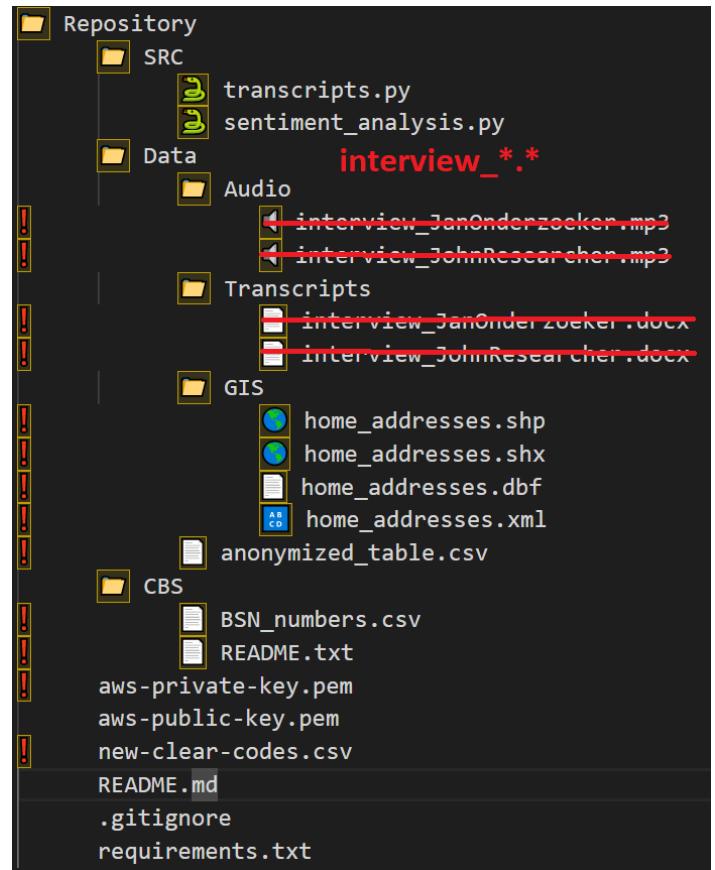
Let's revisit the *.pem example from the first section, if you looked closely, one of the files we did not want to ignore was ignored; aws-public-key.pem. In order to correct this, we can add the pem file we do want to ignore to the list with the line "aws-private-key.pem", or we can use the structure of the naming to avoid any future leak of a private key with "*private*.pem":



File names with variables

Many programmers and researchers use structured file names with space for variables in the file names to make it easier to different datasets. We can also place wildcards in the middle of file names to identify files that match a filename pattern.

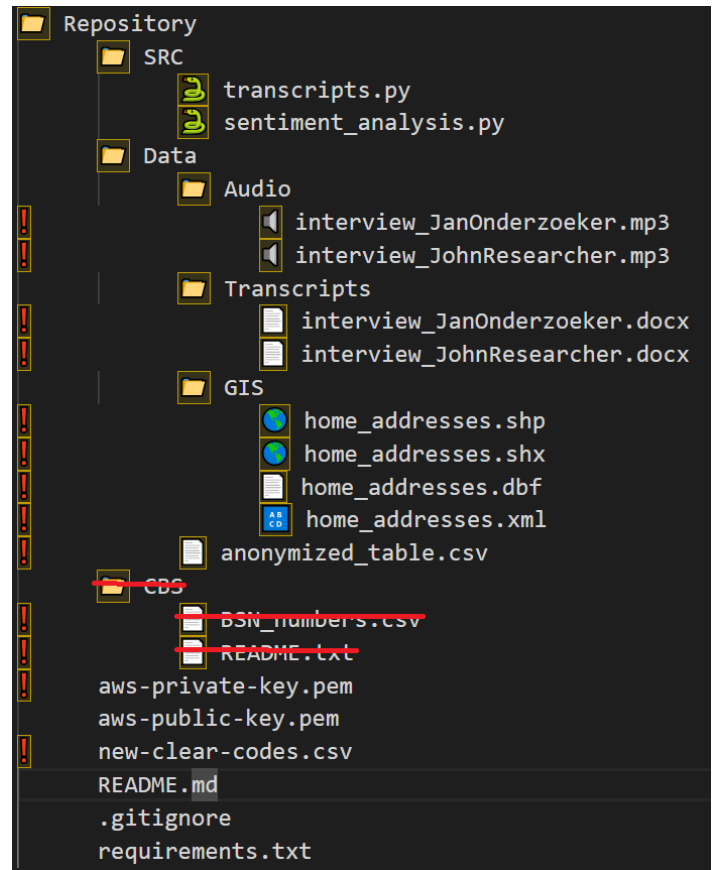
In the following example we have interview data following the structure `interview_{PersonName}.{file extension}`. In order to be sure our Jans and Johns data does not get included in the Git history, we can write a single `.gitignore` line `"interview_*.*`", or two lines to specify each file format; `interview_*.docx` and `interview_*.mp3`:



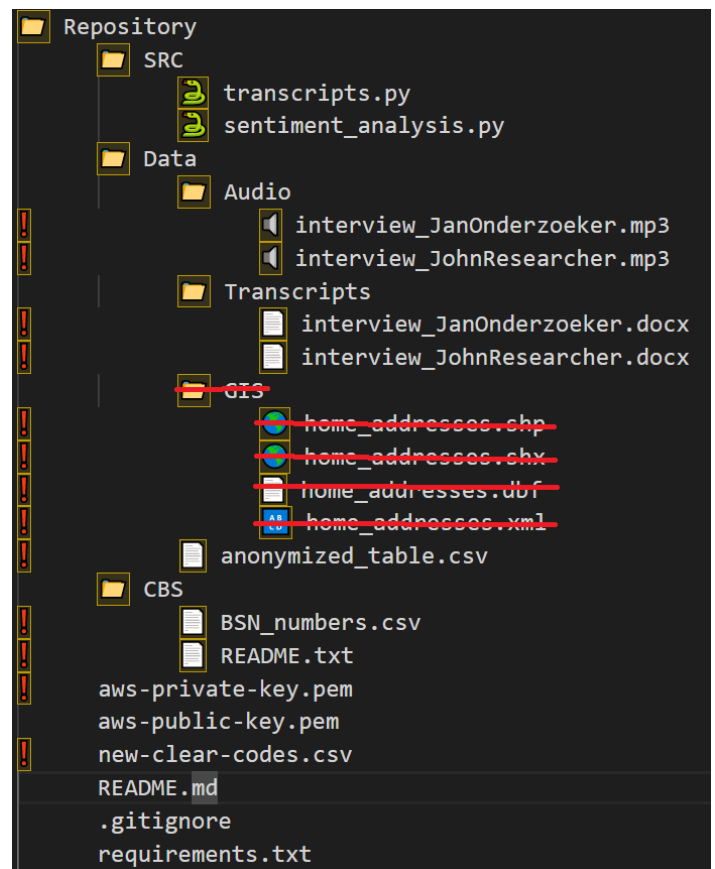
Ignoring Folders

To ignore a whole folder, the pattern is very simple {folder name}/**. On Windows, while normally you use the “\” character in paths, with Git and GitHub Desktop you can use the Linux/Mac “/” character for folder paths.

If you wanted to ignore the entire CBS folder we would add a line reading “CBS/**”.



For sub-folders, we need to give the path to the folder that will be ignored in relation to where the .gitignore file is. In most cases your .gitignore file will be at the base of your repository. In order to ignore the whole GIS folder, relationally to the .gitignore file, given that the GIS folder is in the Data folder we need to write the line “Data/GIS/**”.



Full .gitignore for our example

To demonstrate our full gitignore, we have prepared an annotated screenshot with the folder structure on the left with annotations for which files are ignored by Git, and which lines of the .gitignore file caused them to be ignored. The .gitignore file is on the right. In our example, some files are ignored by multiple lines. There is no harm in using multiple lines to ignore files, and in the interest of sensitive data protection, it is encouraged.

The screenshot shows a file explorer on the left and a .gitignore file on the right. The file explorer displays a directory structure with files and folders. Red annotations (numbers 1-7) are placed next to specific files and folders, indicating which lines of the .gitignore file cause them to be ignored. The .gitignore file on the right shows the corresponding ignore rules.

File Explorer Structure:

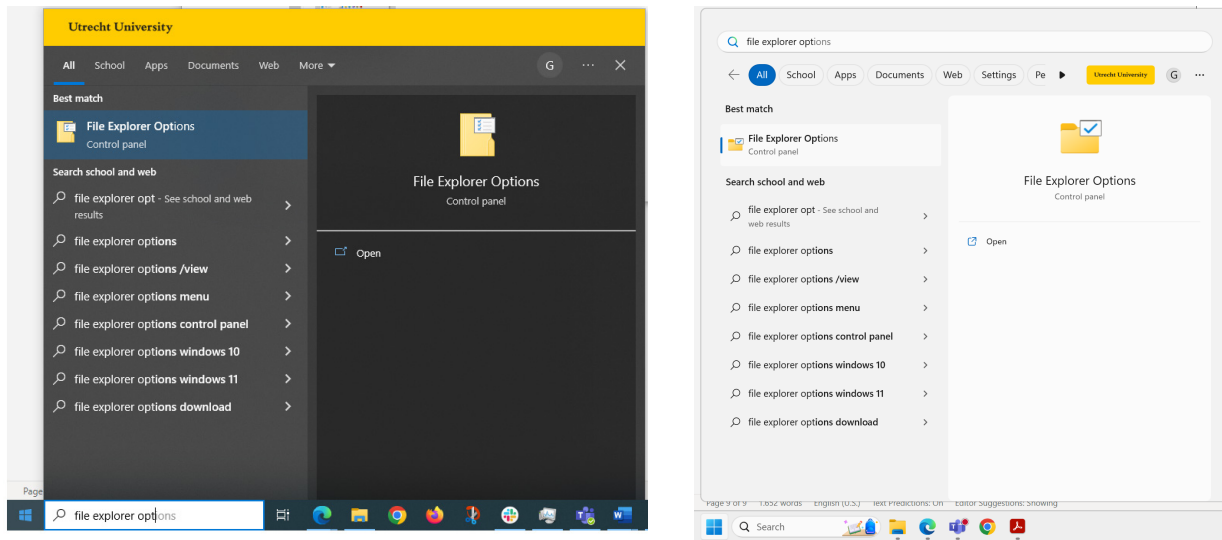
- Repository
 - SRC
 - transcripts.py
 - sentiment_analysis.py
 - Data
 - 6 Audio
 - 1,6 interview_JanOnderzoeker.mp3
 - 1,6 interview_JohnResearcher.mp3
 - 7 Transcripts
 - 1,7 interview_JanOnderzoeker.docx
 - 1,7 interview_JohnResearcher.docx
 - GIS
 - home_addresses.shp
 - home_addresses.shx
 - home_addresses.dbf
 - home_addresses.xml
 - anonymized_table.csv
 - 5 CBS
 - BSN_numbers.csv
 - README.txt
 - aws-private-key.pem
 - aws-public-key.pem
 - new-clear-codes.csv
 - README.md
 - .gitignore
 - requirements.txt

.gitignore File:

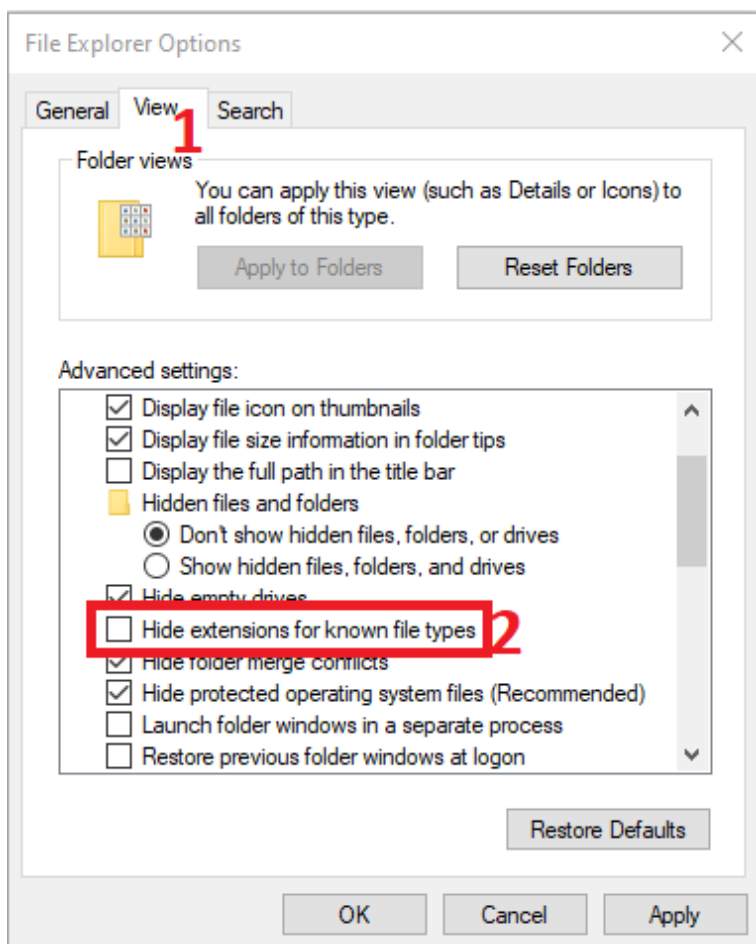
```
1 interview_*.  
2 home_addresses.*  
3 *.csv  
4 *private*pem  
5 CBS/**  
6 Data/Audio/**  
7 Data/Transcripts/**
```

Turning on File Name Extensions

From the start menu, search for File Explorer Options:



In the File Explorer Options window, go to the View tab (1), and uncheck the option "Hide extensions for known file types" (2).



Git History and Reverting Changes

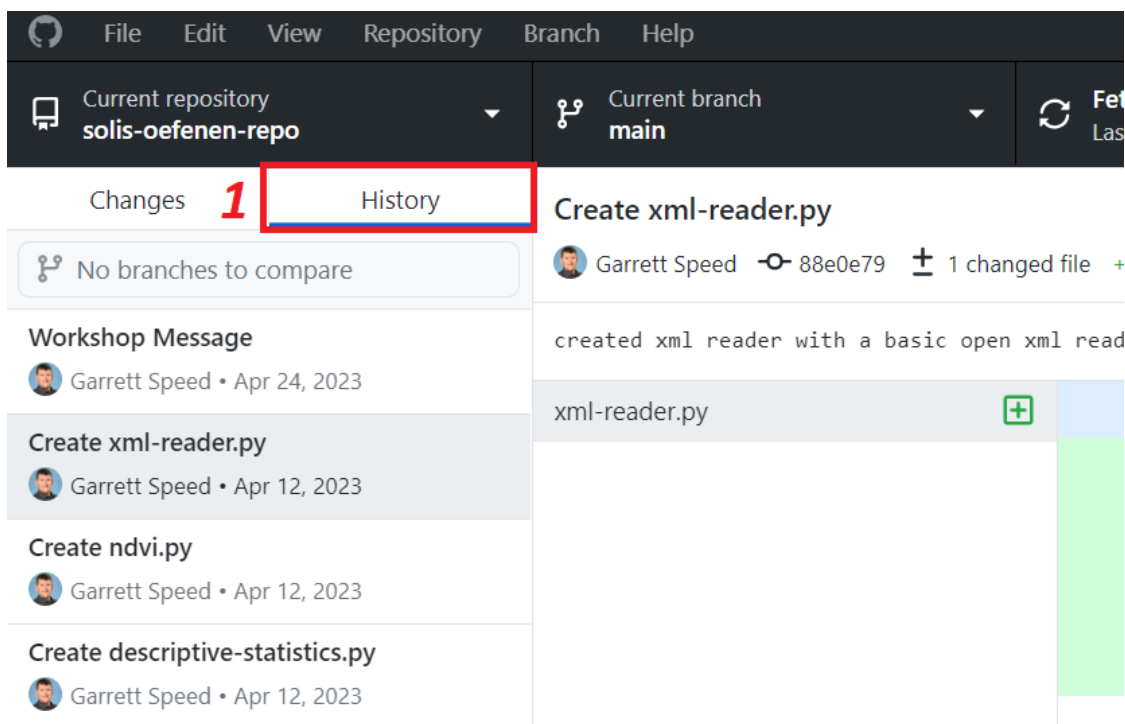
8

Basic History Navigation

GitHub Desktop provides its users with the History functionality, so they can review the details of any commit made in a local repository, including the file changes that this commit included. You can also clone a remote repository locally and inspect the full Git commit history.

Each commit in a GitHub repository contains the following key information; the committer's username on GitHub, the commit's unique ID, the time the commit was made and the commit message and description.

The History of your GitHub repository can be seen, by clicking on the History tab (1) on the left sidebar of GitHub Desktop.



On the History tab click on the commit you would like to review. The selection of multiple commits (2) simultaneously is also possible, by holding Ctrl or Shift buttons on your keyboard.

The screenshot shows the Git GUI interface. The top bar includes 'File', 'Edit', 'View', 'Repository', 'Branch', and 'Help'. Below this, the 'Current repository' is 'solis-oefenen-repo' and the 'Current branch' is 'main'. The 'Fetch origin' button shows 'Last fetched just now'. The 'History' tab is selected, showing 'Showing changes from 3 commits' with '3 changed files' (+17, -0). The commit list on the left includes 'Workshop Message', 'Create xml-reader.py' (highlighted with a red box and labeled '2'), 'Create ndvi.py', 'Create descriptive-statistics.py', and 'Update .gitignore'. The right pane shows the diff for 'descriptive-statistics.py', which includes a function to calculate a mean.

```

Created descriptive statistics file with a function to calculate a mean.

descriptive-statistics.py  @@ -0,0 +1,4 @@
ndvi.py                   1 +def mean(values):
xml-reader.py              2 +     s = sum(values) #sum
                             3 +     m = s / len(values) # mean
                             4 +     return(m)
  
```

To review the changes made to each file of the commit or of multiple commits, just click on the individual file (3). On the right side (4) of the files' list, you can review the changes made to that file.

This screenshot shows the same Git GUI interface, but with 'xml-reader.py' selected in the file list (labeled with a red '3'). The right pane now shows the diff for 'xml-reader.py', which includes a function to parse XML files (labeled with a red '4').

```

Showing changes from 3 commits

xml-reader.py  @@ -0,0 +1,6 @@
               1 +from xml.dom import minidom
               2 +
               3 +def open_xml(filename):
               4 +     f = open(filename, "r")
               5 +     xmldoc = minidom.parseString(f.read())
               6 +     return(xmldoc)
  
```

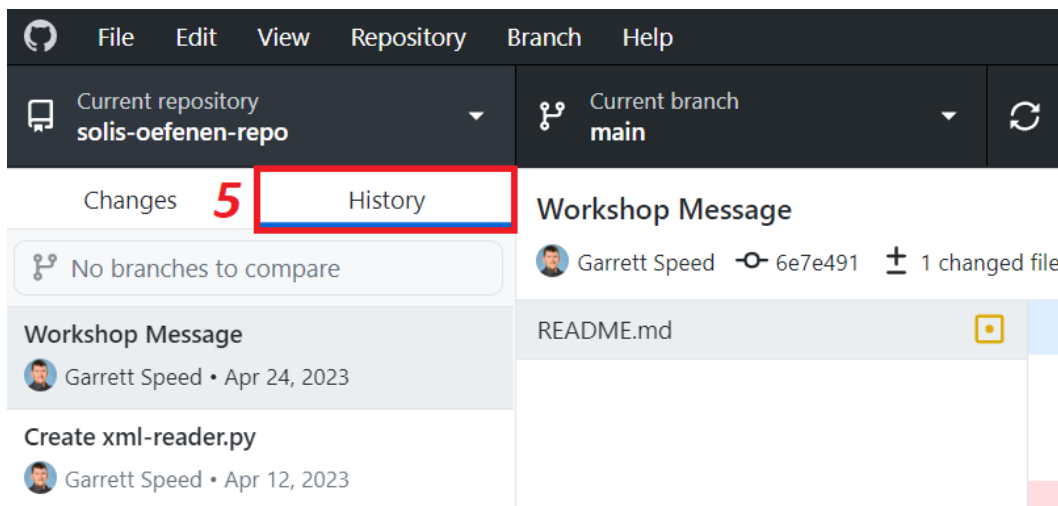
Reverting to a Previous Commit

Reverting to a previous commit allows you to remove any changes made with this commit, from your repository. Bear in mind that the revert action is also a commit to your repository. Based on that the initial commit also remains in your repository's history.

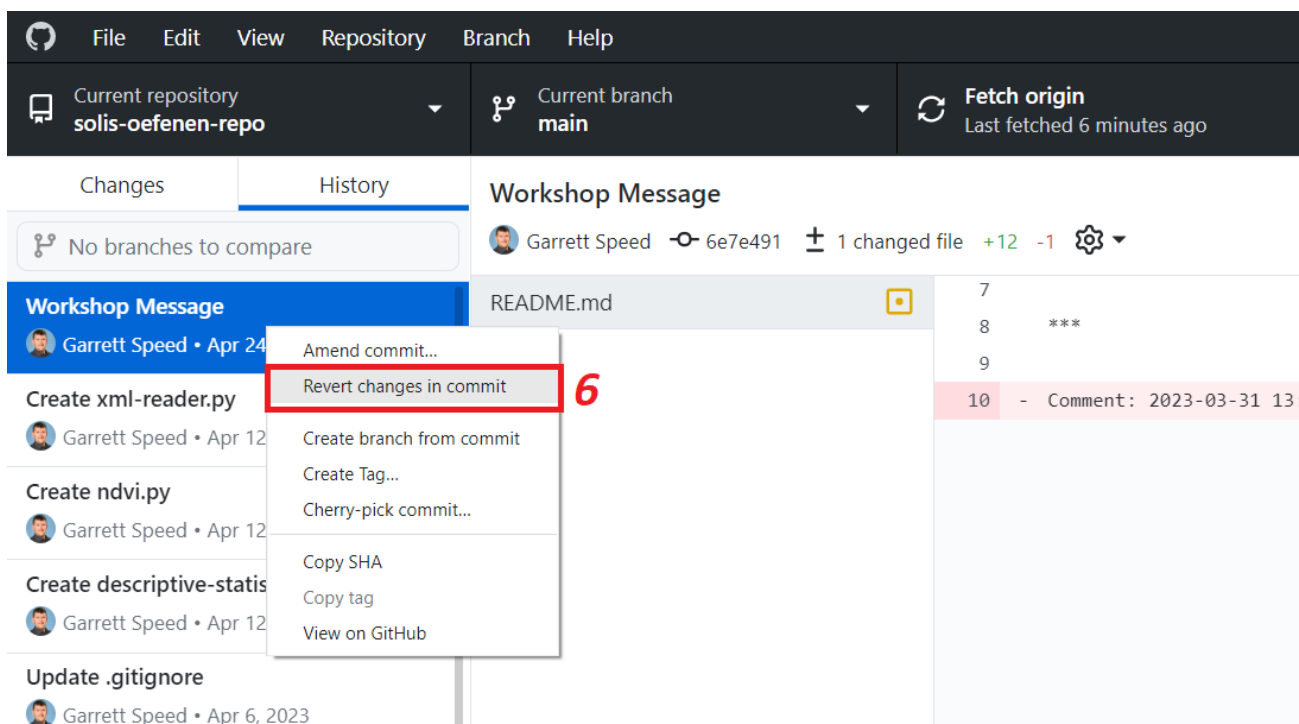
The reverting functionality is available only through the GitHub Desktop application and the command line of Git and not through the GitHub online interface. However, for the sake of this manual we will present only the reverting steps of a commit on GitHub Desktop.

To revert a commit through GitHub Desktop, click on History (5) in the left sidebar.

If you like to revert multiple commits, it is advisable to revert from the newest to the oldest one. Reverting commits in different order might cause merge conflicts in your repository.



Then right click on the commit you would like to revert and select Revert Changes in Commit (6).



After reverting the changes in your commit, you can see the previous status of your commit (7).

File Edit View Repository Branch Help

Current repository: solis-oefenen-repo

Current branch: main

Fetch origin: Last fetched just now

Changes History

No branches to compare

Revert "Workshop Message"

AristotleKandylas • 86ecac3 • 1 changed file • +1 -12

This reverts commit 6e7e491dac6d0d524710278c8daed89fb1fb8cd4.

7

README.md

@@ -7,15 +7,4 @@ Comment: 2023-03-31 11:19 UTC

7 ***

8 ***

9 ***

10 - Comment: 2023-03-31 13:11 UTC

11 -

12 - ## What is this repository?

13 -

14 - This repository is used in my Getting Started with Git and GitHub (GSGG) workshop at the Utrecht University faculty of Geosciences. The workshop introduces Git through GitHub.com and GitHub Desktop, demonstrating how to get started versioning and sharing code.

15 -

16 - If you want to see our schedule of workshops in addition to this one, [visit our workshops page](https://geo-data-support.sites.uu.nl/workshops/)

17 -

18 - If you want to see our workshop materials, find them at one of these links (*Check back often, we're updating and creating new materials!*)

10 + Comment: 2023-03-31 13:11 UTC

Adding Collaborators to a Repository

9

Collaborators can help you manage your repository and contribute code without performing merge/pull requests (merge/pull requests will be discussed in our next manual).

Collaborators with administrative rights on the repository can do most management functions discussed in the next chapter (Repository Administration), and can carry out most procedures within the scope of this manual without prior approval from the repository owner. These procedures could also include committing code changes that conflict with your code changes, which may break the functionality of your code. In our next workshop, Collaboratively Writing Code with Git and GitHub we will demonstrate how to avoid these issues with pull/merge requests and procedures.

You should only add your co-workers or other members of a consortium that will be helping you directly. Adding collaborators gives them permissions to manage your repository, and adding a random person can be a security risk.

The official documentation for this process can be found here: [Inviting collaborators to a personal repository](#)

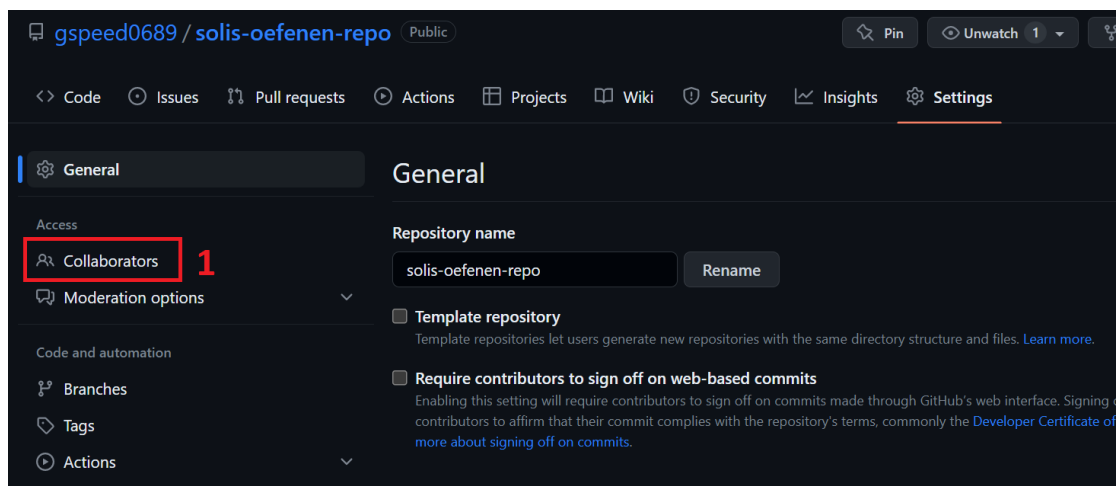
Collaborators can be added from the repository page on GitHub.com through the settings page (1).

The screenshot shows the GitHub interface for a repository named 'solis-oefenen-repo' by user 'gspeed0689'. The 'Settings' tab is highlighted with a red box and a red number '1'. The repository is public and has 1 branch (main) and 0 tags. The file list shows the following files and their commit history:

File	Commit Message	Time Ago
.gitignore	Create .gitignore	2 days ago
LICENSE	Initial commit	last week
README.md	Updated README with another comment	5 days ago
mycode.py	Added dict_grouping to mycode.py	2 days ago

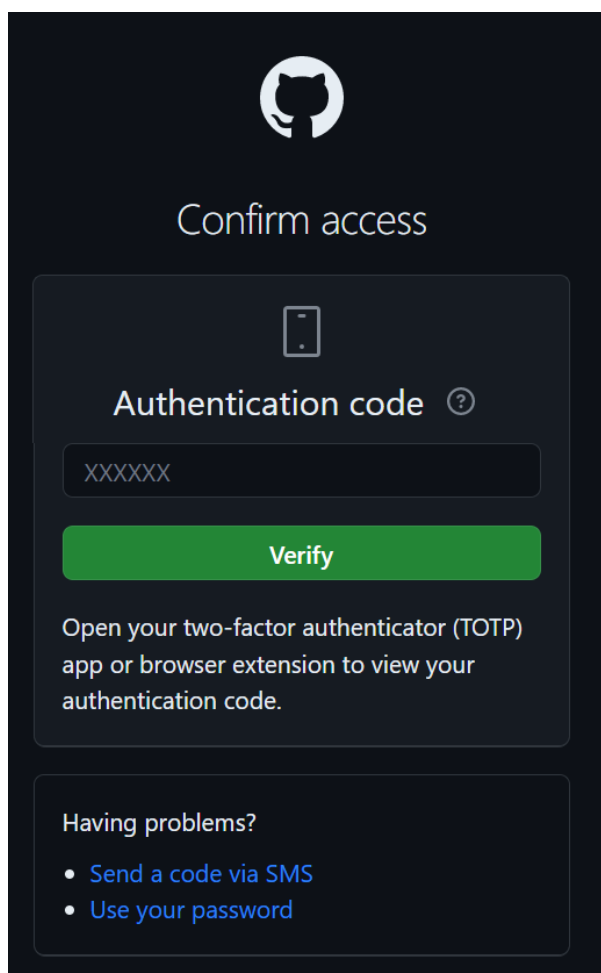
The 'About' section on the right shows the repository is a test repository for the 'G' with Git and GitHub Wo. It has 0 stars, 1 watching, and 0 forks.

From the settings page, then click on the collaborators button on the left (1).

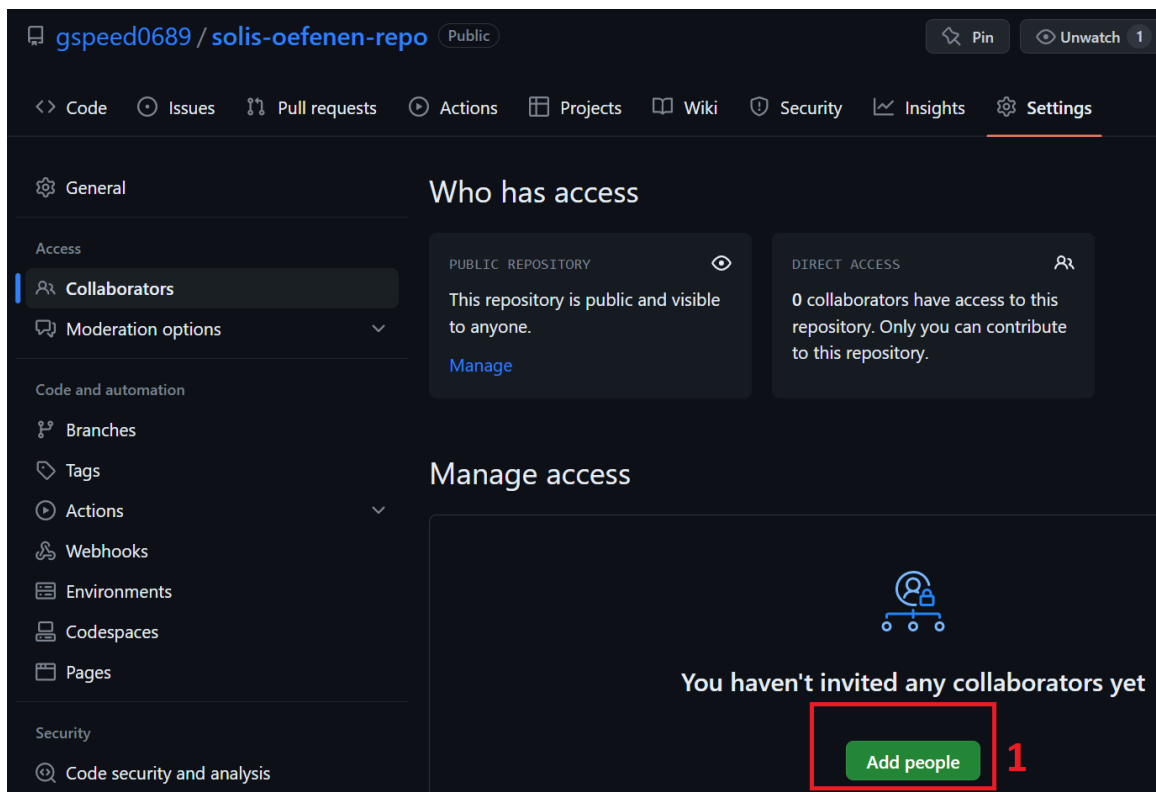


If you have multi-factor authentication set up, GitHub.com will ask for another verification method.

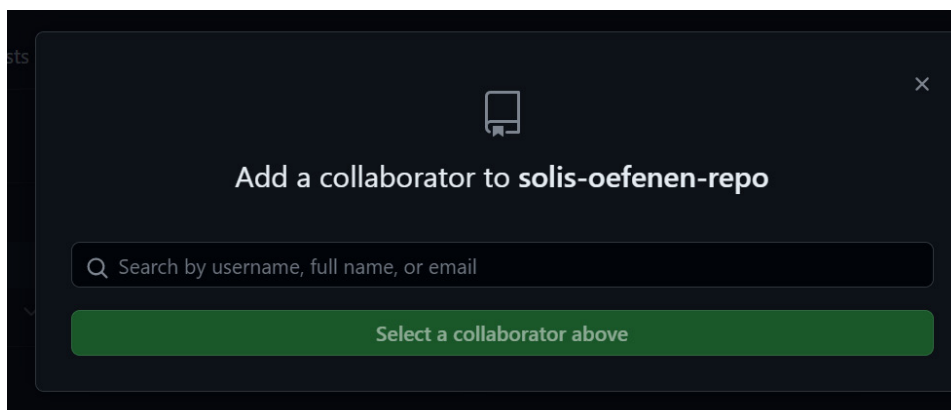
If you have not set up multi-factor authentication, you should do so, it is also required for connecting to the [GitHub organizational account for Utrecht University](#).



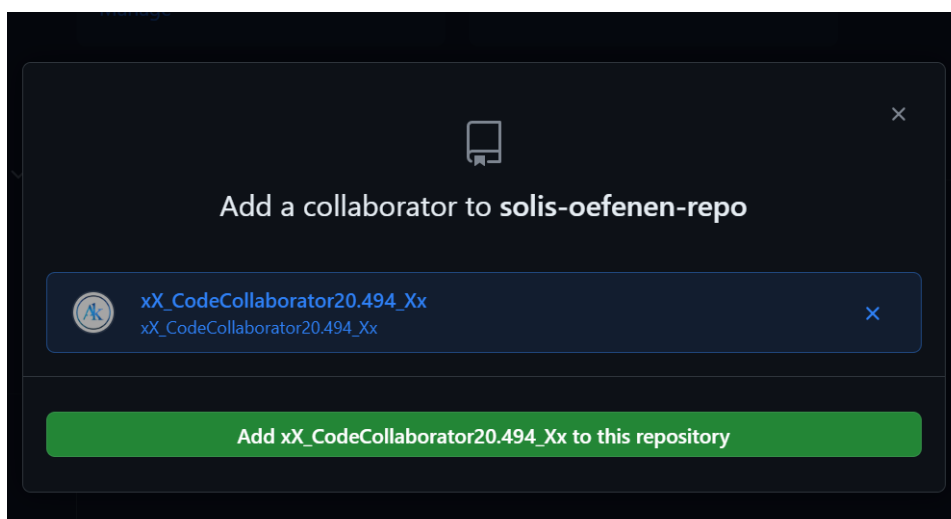
After authenticating, you can now add collaborators with the add people button (1).



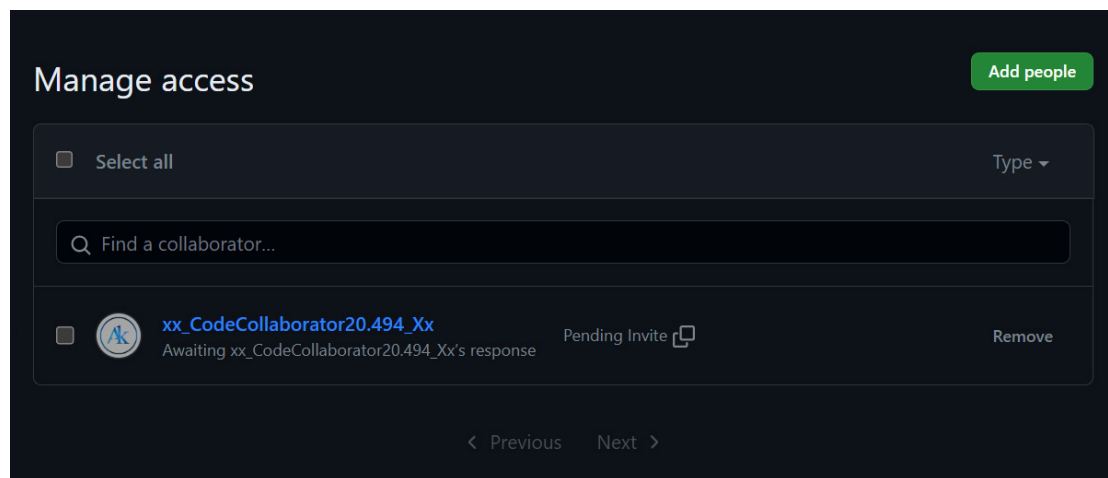
The add people pop-up will now show up.



While GitHub.com says that you can add people with their name or email address, this is actually not so easy. It is best to find the GitHub username of the person you would like to add to your repository.



When you have found your collaborators, you can click the add button and they will then show up in the collaborator list, with a pending message next to their name. Your collaborator will need to accept the invitation to be formally added to the repository, they also need to accept this within 7 days of the addition to the repository. If they do not accept in this time frame, you can simply remove them and repeat this process.



Utrecht University GitHub Organization

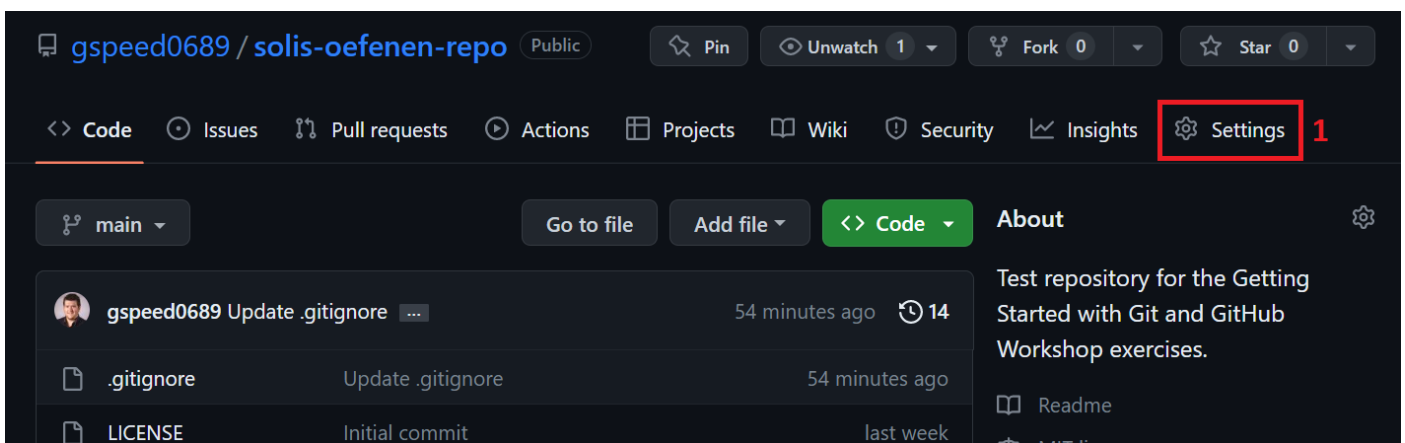
The Utrecht University GitHub organizational account grants Enterprise benefits for repository administration. One of the extra features is that you can choose between 5 different permission levels that you want to give to collaborators in a Repository that you create in the organization (<https://docs.github.com/en/organizations/managing-user-access-to-your-organizations-repositories/repository-roles-for-an-organization>).

In the UU organization you can also create teams and subteams. You can invite teams to your repositories. Teams get a dedicated page in the organization and their own repository overview.

Other Repository Administration Procedures

10

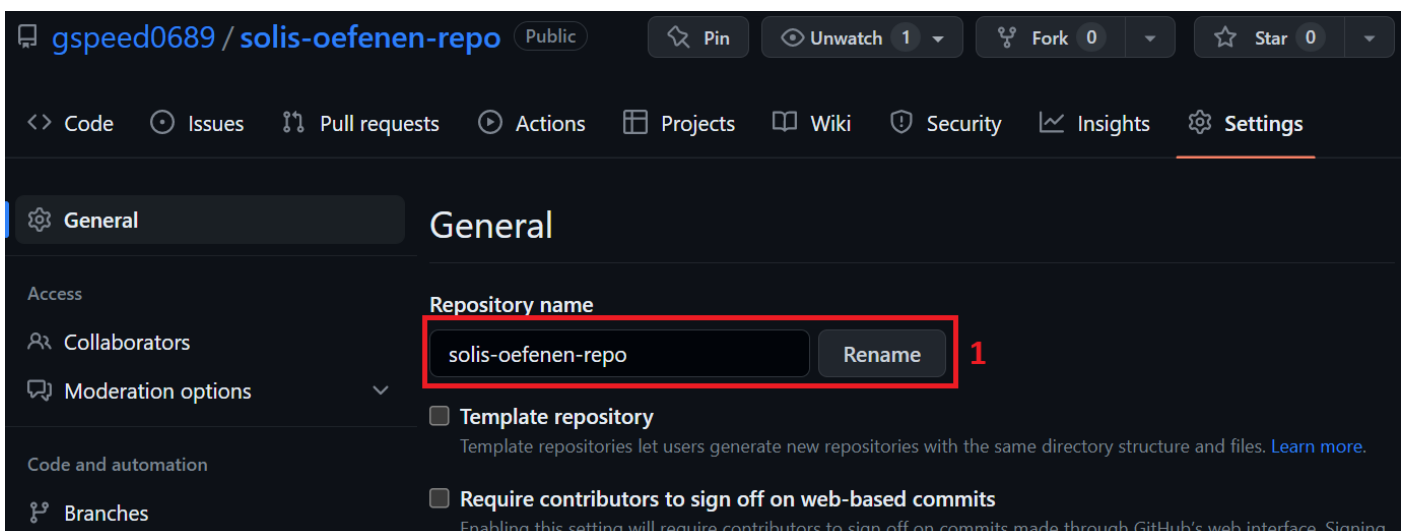
All of the procedures in this chapter will start from the settings page of the GitHub.com repository. To find the settings page, go to the main page of your repository, and find settings at the top on the right (1).



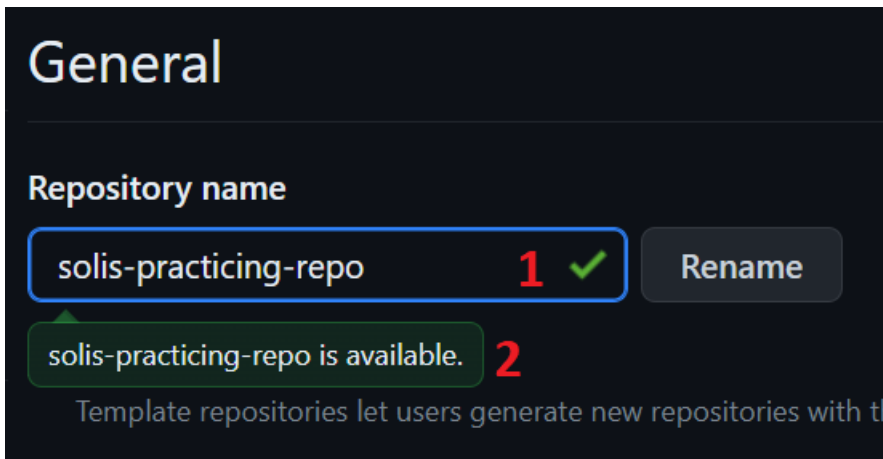
Changing the Repository Name

The official documentation for this process can be found here: [Renaming a repository](#)

Changing the repository name is the first item on the settings page, and will be pre-filled with your current repository name (1).

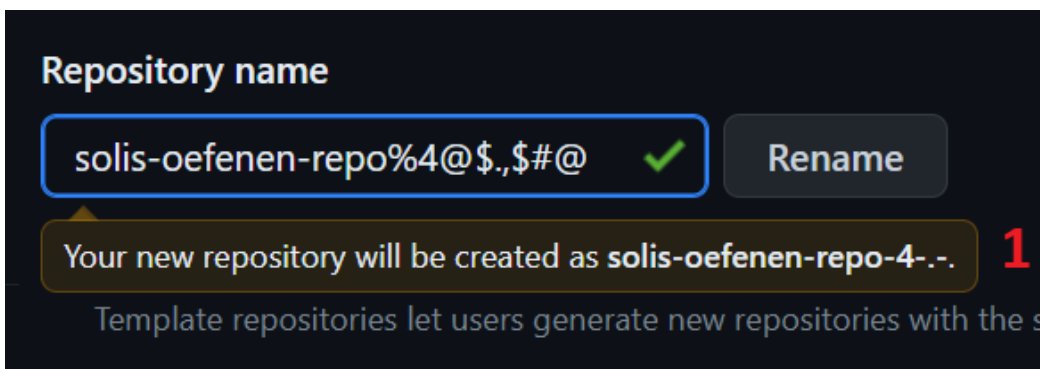


To change, write a new name into the box, if everything is okay with the new name, GitHub will tell you the name is available (2).



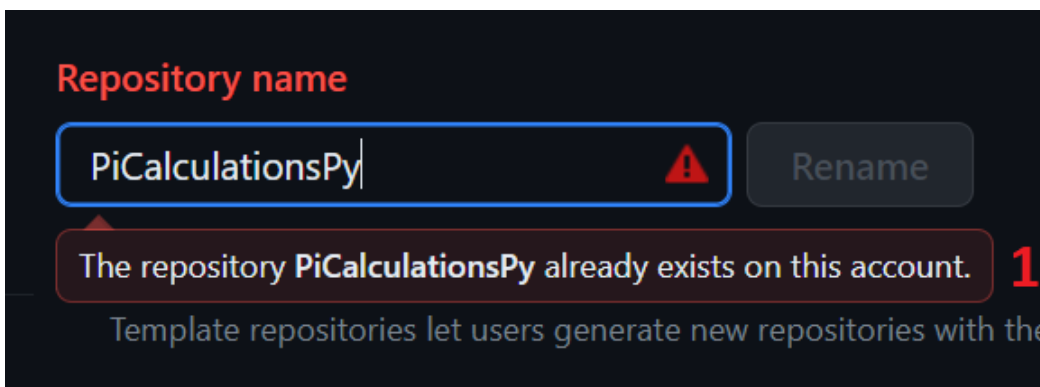
The screenshot shows the 'General' tab of a repository. The 'Repository name' field contains 'solis-practicing-repo'. To the right of the field is a green checkmark and a red number '1'. A 'Rename' button is visible. Below the field, a green message box says 'solis-practicing-repo is available.' with a red number '2'.

Names can only contain letters, numbers, dashes (streepjes), underscores (liggenstreepje), and periods (dot, full stop, punt). GitHub will let you rename the repository with other characters, but will disclose what name it will actually use (1).



The screenshot shows the 'Repository name' field with the text 'solis-oefenen-repo%4@\$.,\$#@'. To the right of the field is a green checkmark and a red number '1'. A 'Rename' button is visible. Below the field, a yellow message box says 'Your new repository will be created as solis-oefenen-repo-4-.-.' with a red number '1'.

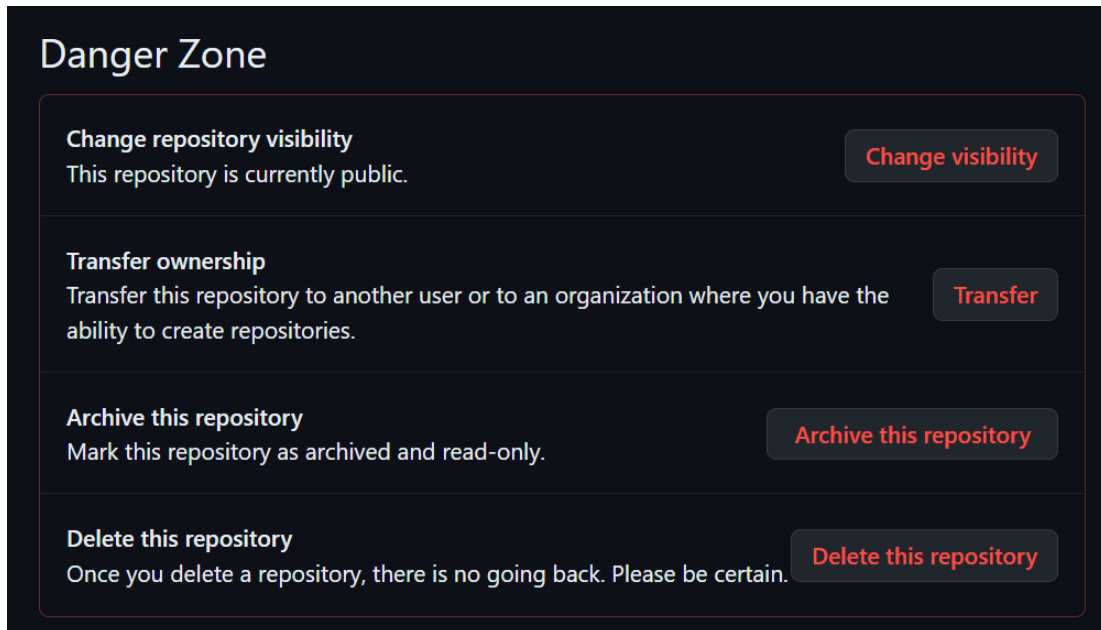
Names must also be unique to the account they are owned by, if there is repository with the same name, GitHub will prevent you from renaming the repository (1).



The screenshot shows the 'Repository name' field with the text 'PiCalculationsPy'. To the right of the field is a red warning triangle and a red number '1'. A 'Rename' button is visible. Below the field, a red message box says 'The repository PiCalculationsPy already exists on this account.' with a red number '1'.

Danger Zone!

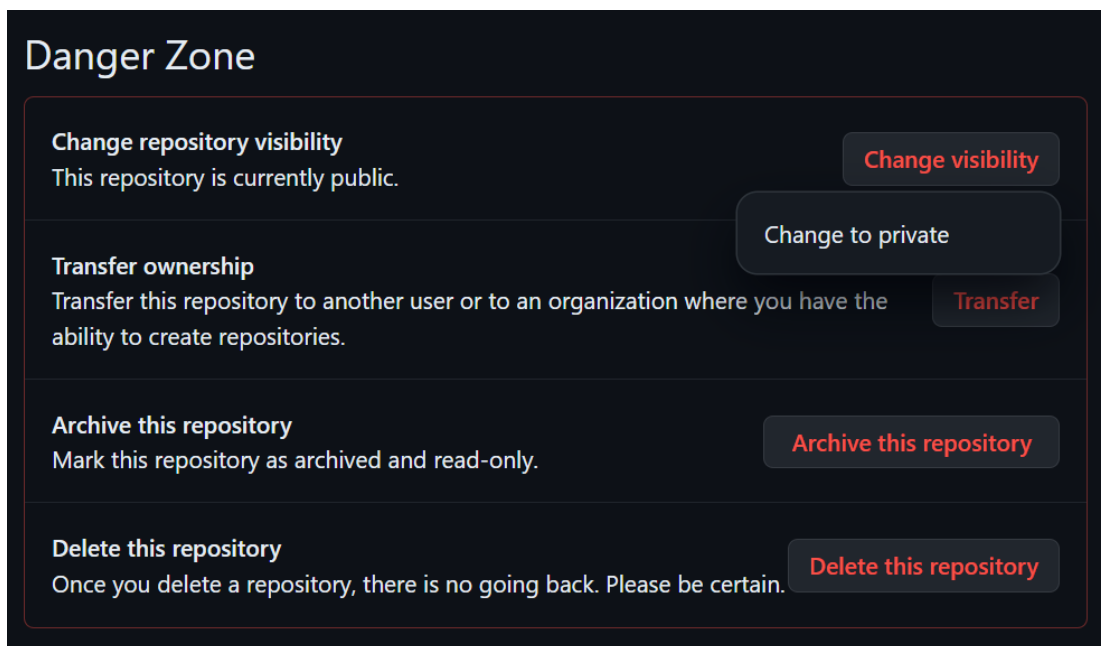
At the bottom of the settings page is the Danger Zone! Here you will find tools that when run will severely alter your ability to administer a repository, such as making it public, private, transferring it to someone else, or deleting it.



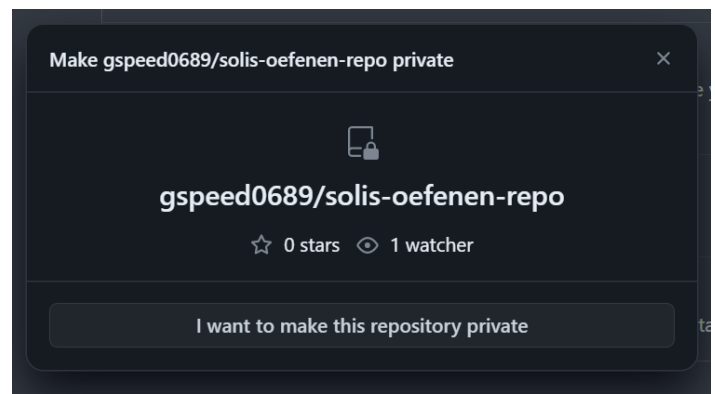
Changing the Visibility of the Repository: Public to Private

The official documentation for this process can be found here:
[Setting repository visibility](#)

If you wish to change the visibility of your repository, click the "Change visibility" button. GitHub will ask to confirm in several ways you want to change to a new status, and the first is to confirm "Change to private".

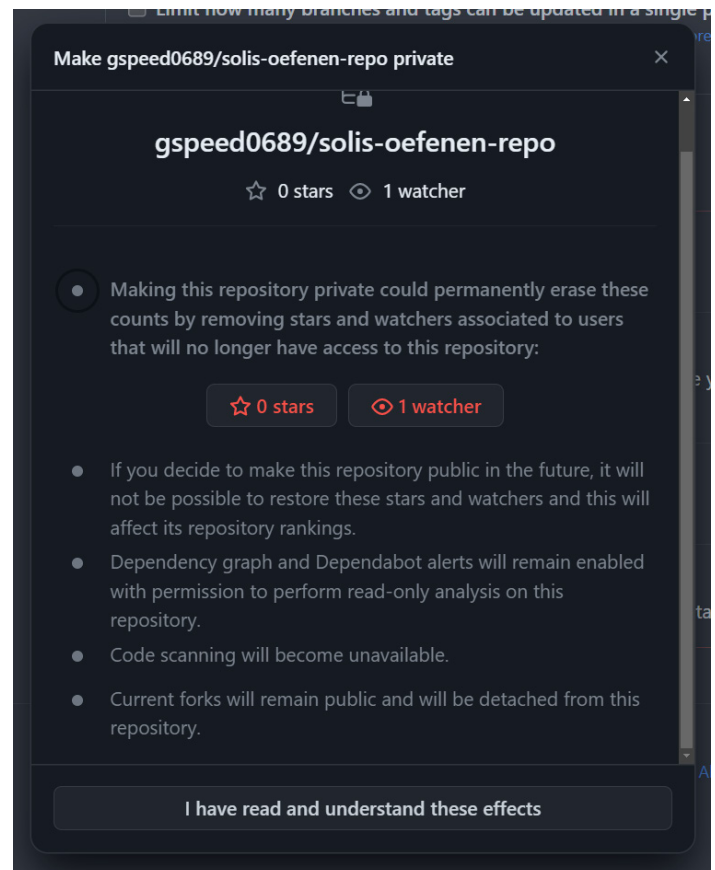


It will again give you an overview of the repository status, and ask you to confirm if you want to make it private.

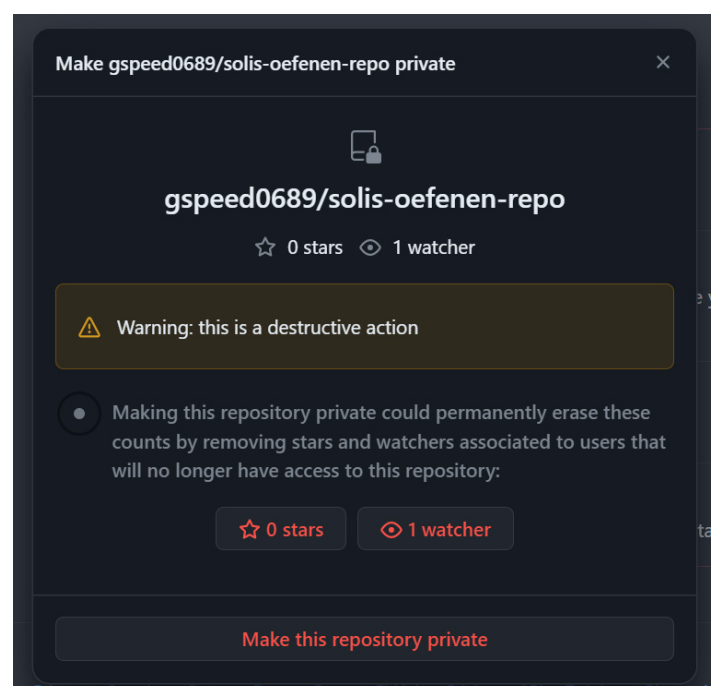


It will then inform you of the ramifications of making the repository private, which are:

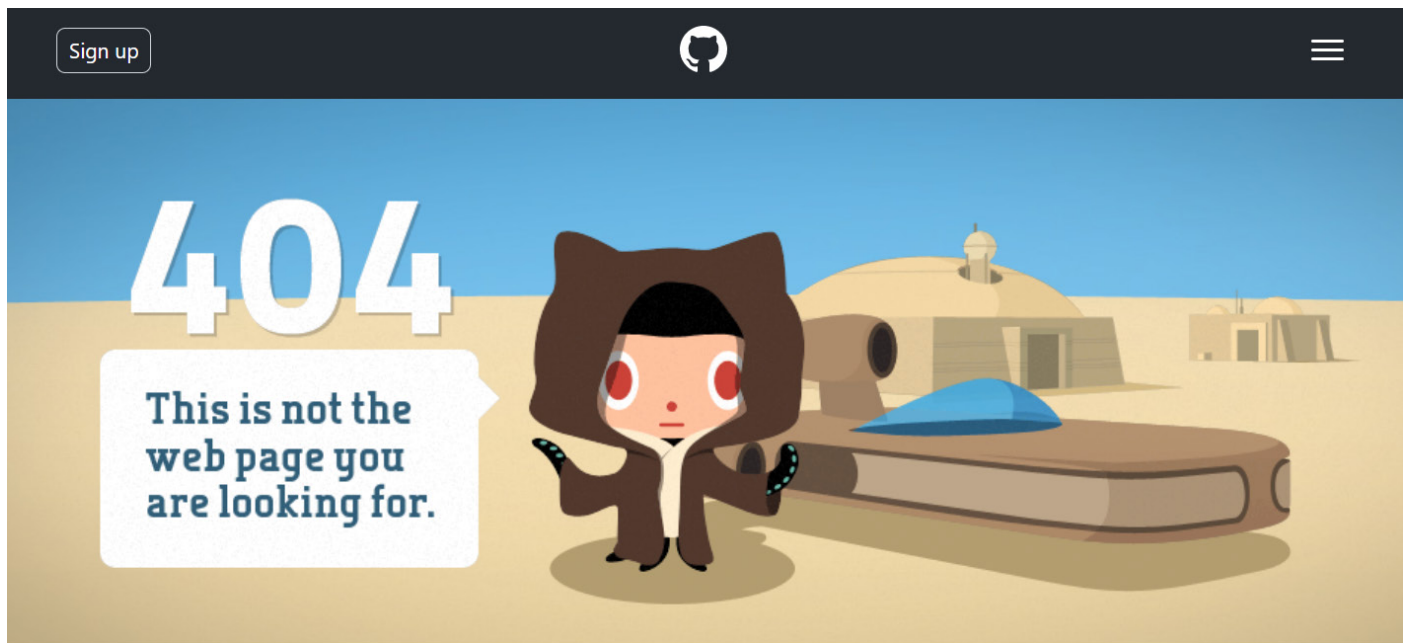
- Links to the repository will no longer work,
- If there are people following the repository they will no longer be able to follow,
- Forks of your repository will no longer be linked to yours (we discuss forks in our Collaborating with Git and GitHub workshop).



GitHub will then remind you yet again that if you make the repository private, the social media aspect of GitHub will be reset to 0. This is the final step to make it private.



After clicking this button, if you try to navigate to the page (while not logged in), you will find an Octocat (the GitHub mascot) making a pop-culture reference on a 404 page.



Find code, projects, and people on GitHub:

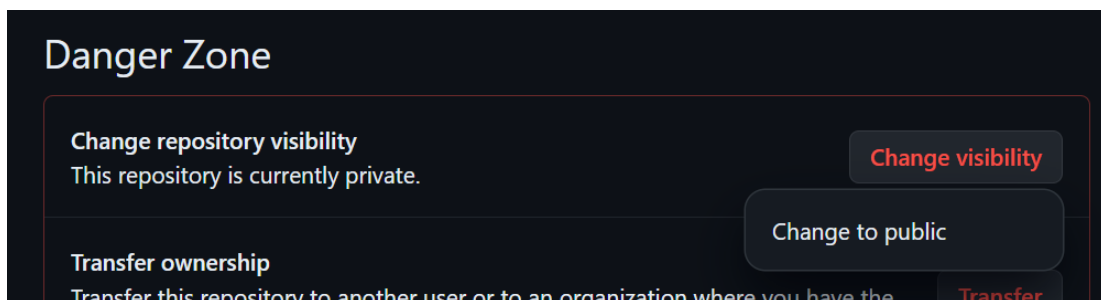
Changing the Visibility of the Repository: Private to Public

Making a repository public from private is very similar to making a public repository private.

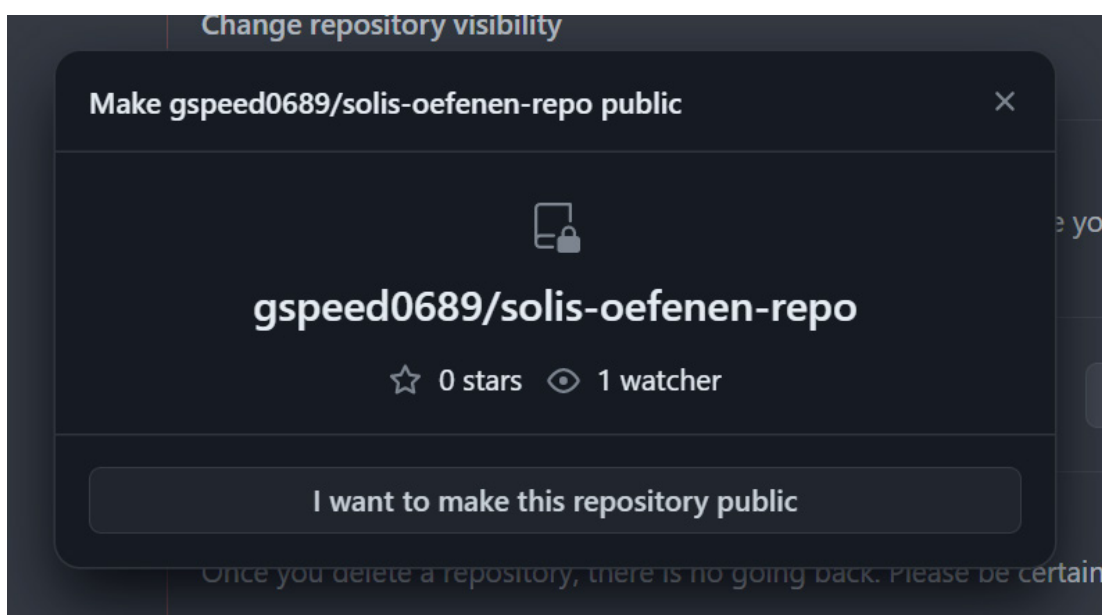
The official documentation for this process can be found here: [Setting repository visibility](#)

If you work with personal data, be sure that no personal data has ever been included in the Git repository's history. When you make a repository public, the Git history also becomes public. If there was inclusion of personal data at any time, you should create a new repository, and copy your code into the new repository. This will delete your versioning history, but prevents any (history of) personal data from being published.

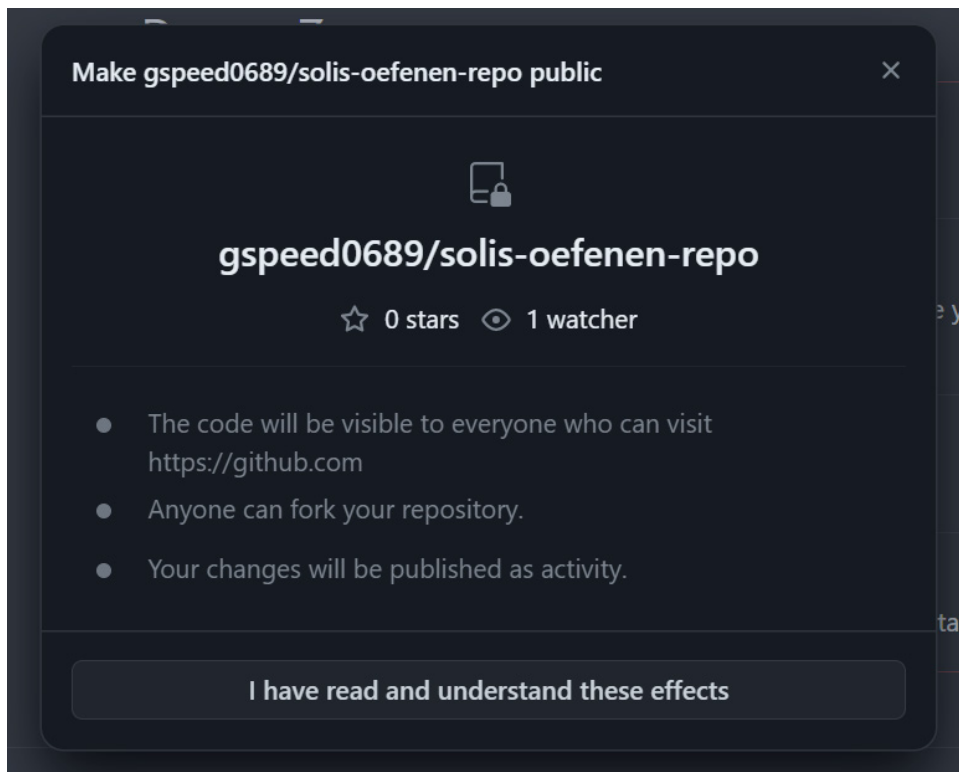
To get started with making the repository public, click the change visibility button, then click on change to public.



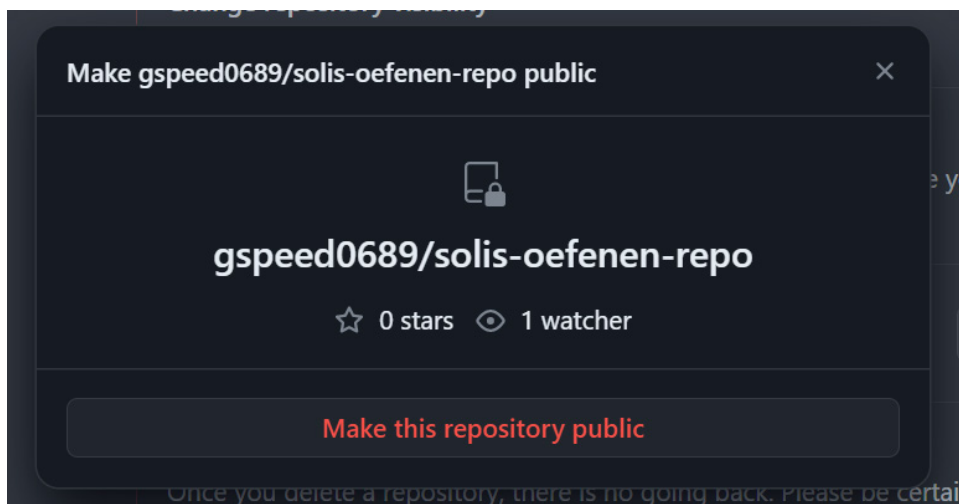
GitHub will ask you to confirm the repository to change to make public.



GitHub will explain the ramifications of making the repository public.



Finally, it will ask you to confirm "Make the repository public".



Now others will be able to find your repository on GitHub.com.

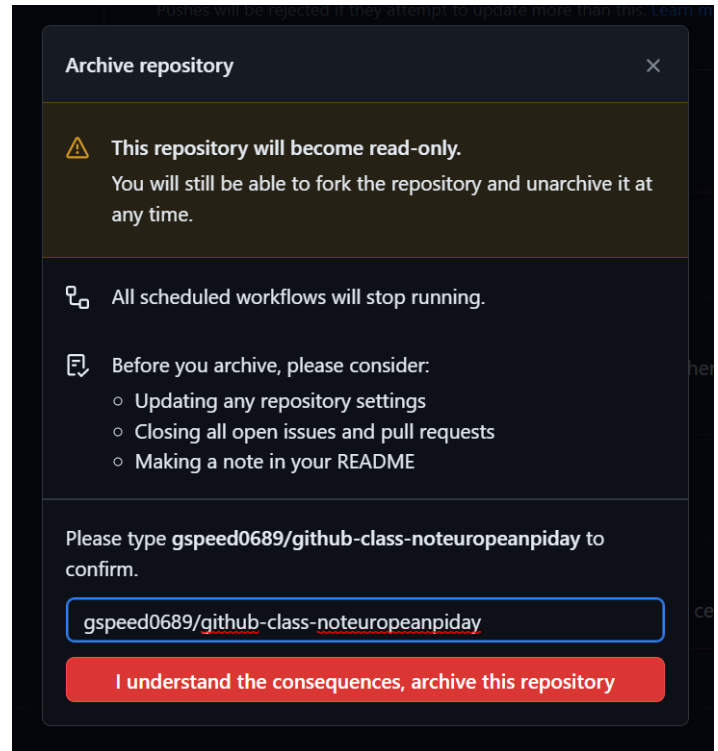
Archiving a Repository

The official documentation for this process can be found here: [Archiving repositories](#)

In the transfer ownership repository section, we will mention that you should not transfer ownership to most other persons if you are no longer working on the code in your repository. If development has ended and you will no longer make updates to the code, you should consider archiving your repository. Archiving a repository puts it into a read-only state, so no changes can be made to it.

To archive a repository, click on the “Archive this repository” button in the Danger Zone.

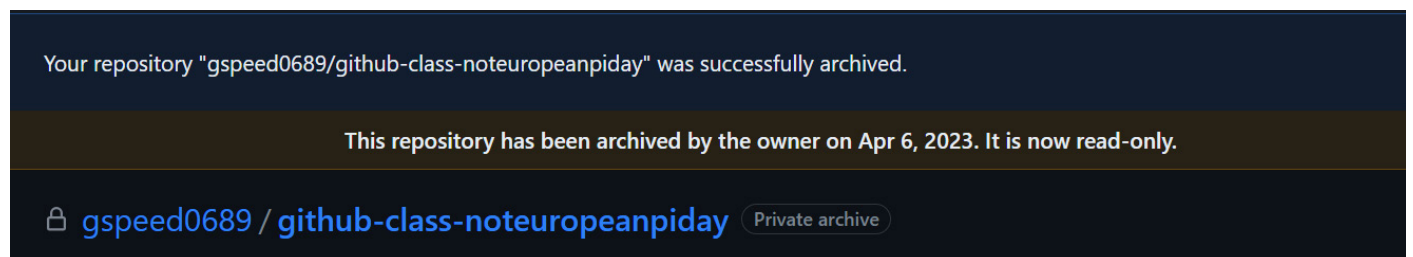
The archive pop-up will then appear, read about what happens to archived repositories, and if the effects are okay with you, type in your username and repository name to confirm.



The screenshot shows the 'Archive repository' dialog box with a close button (X) in the top right corner. It contains the following information:

- Warning:** This repository will become read-only. You will still be able to fork the repository and unarchive it at any time.
- Workflow Status:** All scheduled workflows will stop running.
- Considerations:** Before you archive, please consider:
 - Updating any repository settings
 - Closing all open issues and pull requests
 - Making a note in your README
- Confirmation:** Please type `gspeed0689/github-class-noteuropeanpiday` to confirm.
- Input Field:** A text box containing `gspeed0689/github-class-noteuropeanpiday`.
- Action Button:** A red button labeled 'I understand the consequences, archive this repository'.

When you confirm, GitHub will bring you back to the settings page, and display two messages, first that the repository was archived, and second that the repository was archived on a certain day and is in a read only state.



The screenshot shows the repository settings page with the following elements:

- Success Message:** Your repository "gspeed0689/github-class-noteuropeanpiday" was successfully archived.
- Archive Details:** This repository has been archived by the owner on Apr 6, 2023. It is now read-only.
- Repository Header:** `gspeed0689 / github-class-noteuropeanpiday` with a 'Private archive' badge.

Deleting a Repository

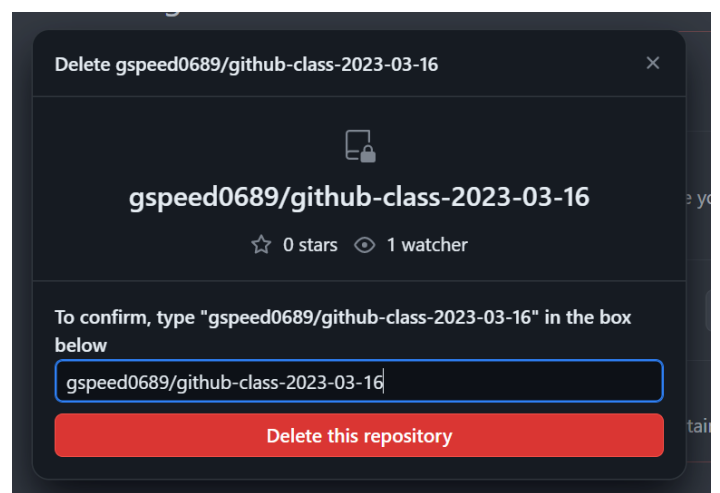
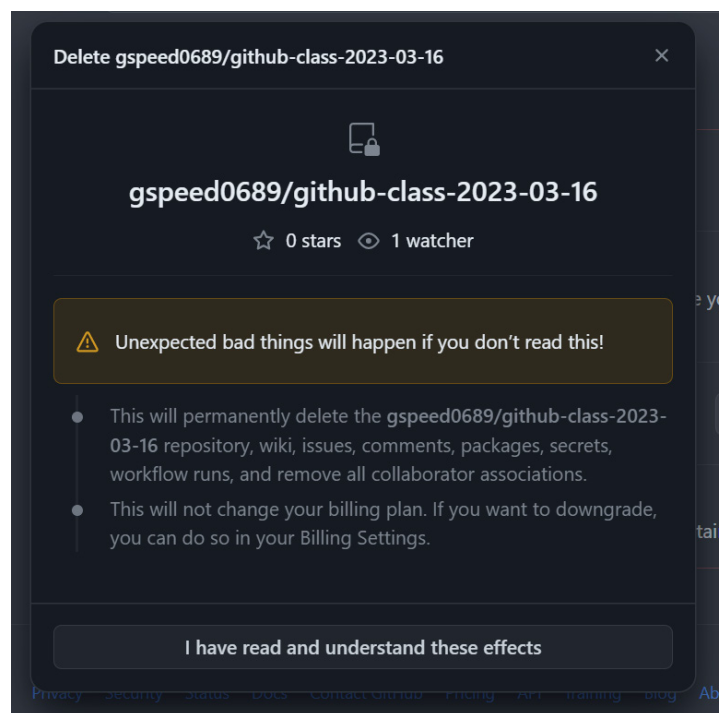
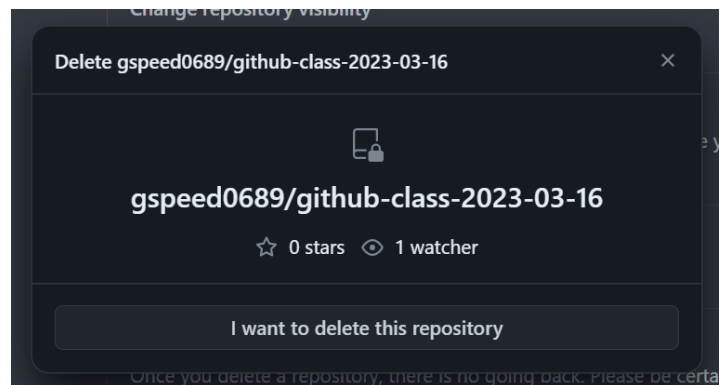
The official documentation for this process can be found here: [Deleting a repository](#)

Deleting a repository will remove the repository from GitHub.com, it will not delete the repository from your local computer if you still have a copy there. You cannot un-delete a repository, so if you may need the code again in the future, keep a local copy, or make the code private on GitHub.com.

From the danger zone, click on the "Delete this repository" button, it will immediately ask you to confirm the repository you wish to delete.

Next it will explain the ramifications of deleting the repository.

To confirm deletion, you will need to type out your username and the repository name.



Then GitHub will take you to your profile page, and show a banner confirming the deletion of the repository.

Your repository "gspeed0689/github-class-2023-03-16" was successfully deleted.

Transfer Ownership

The official documentation for this process can be found here:
[Transferring a repository](#)

Transferring ownership is almost never recommended, and this manual will not demonstrate the process.

There are several reasons not to transfer ownership:

- Transferring ownership to organizational accounts (such as Utrecht University) will remove your administrative access to the repository, and you would then have to go through UU ICT admins to get back your admin rights
- If you have stopped updating the code, you should still keep ownership of the project and let others fork the project from your repository. You can add documentation to your repository pointing people to the new repository with new updates.
- Never transfer ownership to someone you do not know even if they promise to update your code, they may change it to become a vehicle for malware.

To list your repository under the Utrecht University organizational account, use the repository mirroring capability in GitHub. The Geo Data Team will create a manual on how to do this in the future, however this is not included currently as it requires using the command line Git interface.

Mirror

The official documentation for this process can be found here:
[Duplicating a repository](#)

Unfortunately, this requires moving to the command line, but the command is very simple.

The command line is out of scope of this manual, the Geo Data Team will produce a short manual in the future for setting up the Git command line and performing a mirror to the Utrecht University organization.

The authors opened a GitHub Feature Request Issue on the repository for GitHub Desktop on GitHub.com. This request was denied as GitHub Desktop does not want to add all of the flag options to the interface.

[git push --mirror Integration #16471](#)

