GitHub: How to navigate and contribute to Git-based projects

A LIBER Digital Scholarship & Data Science Topic Guide for Library Professionals

Nora McGregor Liam Green-Hughes Liam O'Dwyer

2025-05-20

A practical introduction to navigating the GitHub interface with tips on how to use it for contributing and creating projects relevant to library work.

Introduction

GitHub is a collaborative software development platform. Like a kind of Google Docs for coders, it began as a tool for them to collaboratively work on software development projects, ie, allowing different coders to contribute to parts of an overall project, asynchronously and across teams and even international borders, keeping track of changes all the while to make sure the end result is one cohesive software.

GitHub today however is used for many more different purposes by people in all different fields, not just software developers and software projects, to:

- record and share information on a collaborative project
- work asynchronously and internationally
- test and store scripts and technical documentation
- build a public website for a project

GitHub allows people to collaboratively work on and share projects (called 'repositories') so that others can copy them, either to adapt or to contribute back to the original project. That's because any 'public' repository of files on GitHub is accessible via the web. (Note that public repositories are created by default when you start a project, but it is also possible to make your repositories private if you like in the settings).

Repositories may be filled with code, but they may also be filled with pages of text written in markdown and this markdown can be rendered by static site generators such as Jekyll or Hugo into HTML that is understood by the browser. This very website you're reading from right now is being hosted on GitHub!

This guide will help you to understand how to practically navigate and contribute to projects like this one hosted via GitHub, and will demystify some of the basic actions and jargon around using GitHub along the way.

Git, GitHub and GitHub Pages

Before we begin, it can be helpful to first grasp what Git, GitHub and GitHub Pages each are and how they relate to each other:

- **Git** is a distributed open source version control system. Version control allows you to track changes in a set of files. It does this by taking snapshots of repositories at each stage of development - in Git these snapshots are known as 'commits'. In software development this means things can be tested and rolled back, and enables sharing of stages of development. Git can be used on its own but when used with GitHub has more potential for collaborative projects. You can download and install Git on your own machine. It runs in the command line and there are desktop GUIs available. Git for Humans (Alice Bartlett talk at UX Brighton 2016) is a nicely accessible introduction to the purpose and uses of Git.
- **GitHub** is a web-based software development platform. It has many of Git's features and can host Git repositories but also provides a web interface and additional functionalities. You can view millions of repositories hosted there. Many are openly licensed, meaning you can freely copy (or *fork*) them, either to adapt for yourself or to contribute back to the original project. By default the files and folders in a repository you create are public, and with a little bit of extra code, a repository can also be turned into a more public-facing project website, blog or wiki using a feature called GitHub Pages.
- **GitHub Pages** is a feature within GitHub to turn any repository or project into a website. GitHub can also be used to host a site that has been created using other software packages. One example of such a software is **RStudio**, a visual editor for **Quarto** Markdown which is what we use for this *DS Topic Guides* project!

The language of GitHub

A barrier to GitHub for beginners is that it has its own terminology for common tasks and actions. While off-putting at first, they do serve to outline core concepts of both Git and GitHub. When using Git in particular, these are important to understand as they relate to git commands such as *fork, pull, commit*, *clone*, or *branch*. There are many quick guides and cheat sheets to this terminology on the web which can be useful to use as a reference while familiarising yourself:

- GitHub Docs Glossary
- Git for Librarians Glossary

In the next section, we will demonstrate the actions behind these keywords as we walk through three types of ways librarians regularly interact with GitHub based projects:

- Writing an Issue
- Contributing code/content to an existing project
- Re-using existing code/content for a new project

Relevance to the Library Sector (Case Studies/Use Cases)

Both Git and GitHub have many potential applications for librarians and libraries. Library systems, digital collections and digital preservation are key areas where many resources and scripts are community-led and open source, made available via GitHub and other code repositories. For those working with open source platforms such as Omeka, a basic understanding of GitHub provides access to a host of plugins developed by user communities that can be cloned and applied to your local instance where you may not have resources to develop them yourself. In less technical fields of library work there also is the opportunity to use it to develop or share documentation of working groups and special projects. In all of these, there is the facility to raise an *issue* or *fork* a repository and make a *pull request* if you are looking to query or contribute to a project.

Here are some examples of library activity on GitHub from Week one reading for "Git and GitHub for Librarians" course and Library Carpentry Library Carpentry: Introduction to Git: Summary and Setup, with some personal additions of our own:

- Sharing documentation and code for library-related projects and platforms:
  - Living with Machines
  - British Library Repositories
  - Omeka exhibit platform and related plugins
  - Digital projects of Library of Congress
  - Collection Builder
  - 350+ repositories tagged with #code4lib
- Distributing OCR'd text extracted from digitised materials for collections-as-data analysis [example] [example] [example]
- Storing scripts for executing metadata ingests and transformations [example, example, [example]
- Archiving the source texts of open educational resources [example]
- Websites for library workshops, to share course material or sample datasets [Library Carpentry example, NCSU Digital Scholarship Workshops example, DCU Library example],
- Collaborating on and contributing to a project Digital Scholarship & Data Science Essentials for Library Professionals

Let's look more closely at some practical ways you can get started navigating GitHub and contributing and making use of GitHub based projects for your library work!

1. Writing Issues

Issues are a great tool in GitHub to let the people know that something is possibly broken or something new is needed in a project. They help everyone by providing a place for everyone to discuss the concerns raised by the issue and providing a way to manage a response. If you have ever raised a ticket with the helpdesk of your IT department, you will notice a lot of similarities with Issues.

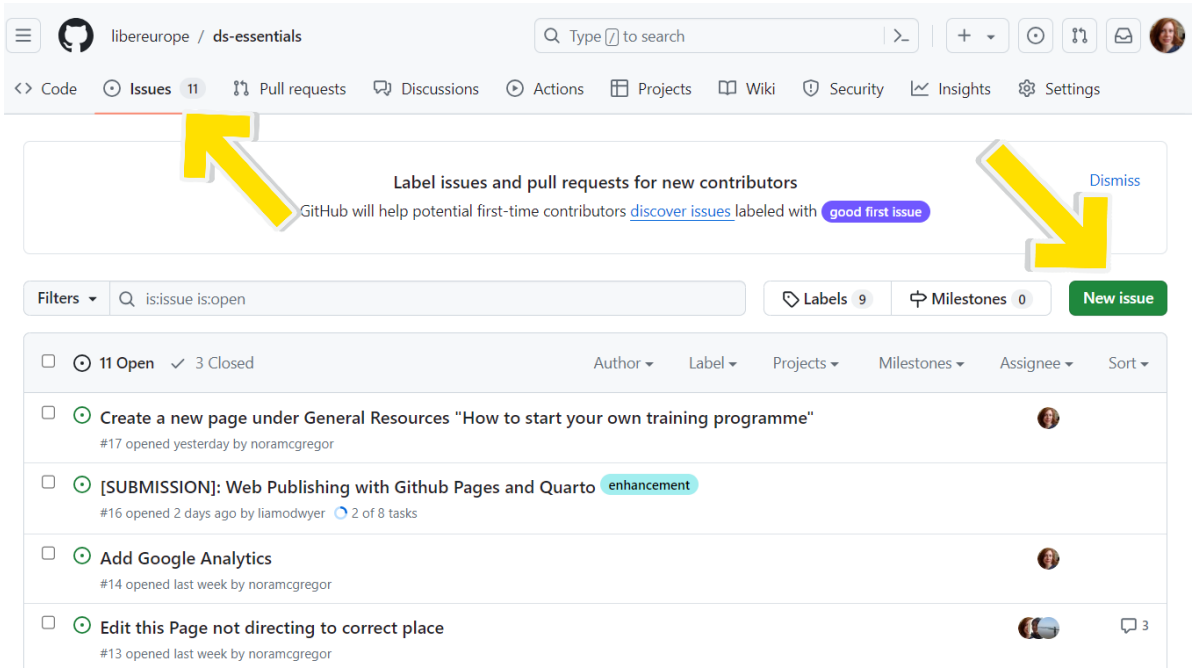

Figure 1: Screenshot of a github project with arrows showing where Issues can be found and created.

Preparing to raise an issue

Issues are a human centred tool in GitHub, so it is up to everybody involved to get the best out of them. Before raising an issue, consider these steps:

- Read the documentation carefully to make sure what you are experiencing is not down to any misunderstandings about the functionality or content provided.
- Have a look through the previous issues to make sure that what you are experiencing has not been raised before. If it has then consider adding details to that ticket and clicking on "Subscribe" under Notifications to be kept up to date with any fixes or changes.
- You can raise issues for a variety of reasons, not just faults. You can also ask for new functionality or make a suggestion. Be clear, if you can, on the type of request you are making as the project maintainers may have different processes for different request types.

- Remember that what is obvious to you might not be obvious to a project maintainer, so explain fully and be patient with those trying to address the issue.
- Some repositories apply labels to categorize the issue (e.g. bug, feature request, documentation, backend, frontend etc.). If you are enable to label the issue you created please consider the available categories.

Reporting Bugs

Raising an issue is a great opportunity to help to get something fixed. Providing good quality and complete information can cut down the time it takes to resolve an issue. A good write up should contain these elements:

- A full description of what happened, including:
  - Any error messages or codes
  - Where you were in the project (for examples, for a web application the URL of the page you were on, or if it is not available, the stream of actions to get there)
  - What you were trying to do
- What did you expect to happen? Including:
  - What normally happens
  - What should have happened
  - If the behaviour is new
- Any other relevant information: the project maintainer will be looking into possible causes of the issue you are experiencing, so extra information can help enormously. For example of a web application this might include:
  - The time of day the error occurred
  - Which browser you use
  - If you have experienced any similar errors with other sites
  - Which operating system you use
  - Provide screenshot (if applicable)
  - Provide version number of the software (if applicable)

Some projects provide issue templates that ask specific questions relevant for the project (for example see the bug report template for Dataverse).

Requesting new features

Issues are not only for reporting bugs. They can be used for requesting new features or further support. In this case explain why it would make sense to add your new feature to the project. Think about who might be affected by your change, both in a positive and negative way. Lastly, consider having a go at the change yourself and submitting a pull request. If this is not possible explain why here.

The life of an issue

Once you submit an issue you will see that it will have the status of "Open". Usually, the issue will then be assigned to a specific member of the project team to examine further. During this process you may see comments added discussing the issue and possible implementations, fixes or feedback. The issue may even have some labels or categories applied to it to help classify the issue type. You may be asked for more information or input to help resolve the issue.

When work on the issue stops, it will be "closed". In GitHub there are two types of issue close:

- Close as completed: This means the issue has been worked on and that work is complete. E.g. this might mean a bug has been fixed or a feature added.
- Close as not planned: This means that the project maintainer has chosen not to work on the issue. This can be for a variety of reasons such as:
  - Won't fix: Adding a fix for this feature is outside the current scope of the project, beyond its resources or deemed unnecessary.
  - Can't reproduce: In software, developers must usually reproduce the error in order to fix it. Sometimes this is not possible to do, this might happen for intermittent errors or errors caused by particular hardware configurations.
  - Duplicate: The issue has been reported elsewhere. There will usually be a reference to where this is.
  - Stale: Sometimes issues stay in the system for a long time. This might be because the initial reporter has moved on and is unable to answer queries about it. In this case the issue will be removed.

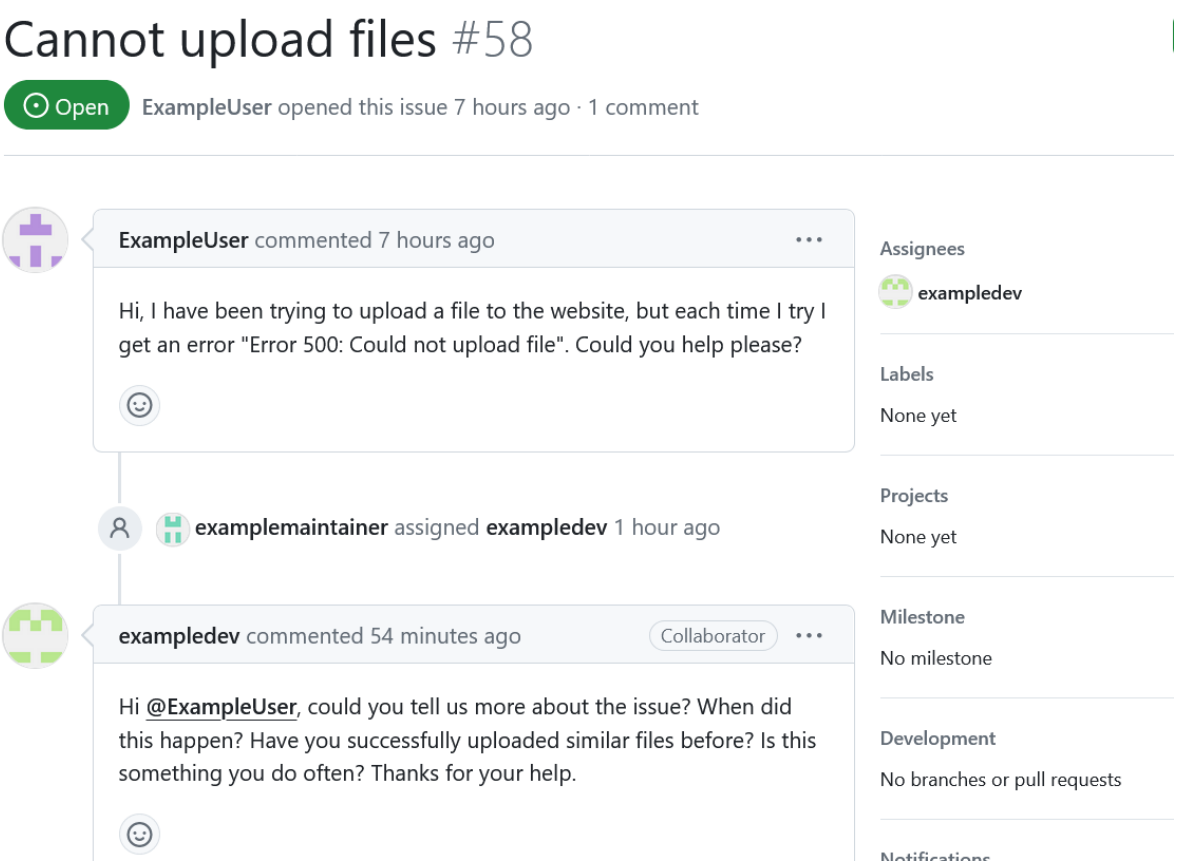

Figure 2: Screenshot of a conversation being held about an issue on a github project.

With an issue, there is no expectation that you will provide a solution to the problem you are facing or the new functionality that you would like. At some point you might contribute a fix or something new to a project, maybe as a result of an Issue. When this happens you will raise a "pull request". This is a proposal to make a group of changes to files in a git repository. A maintainer can then decide whether to accept the pull request and "merge" it with the project, reject it, or send it back for further work.

2. Contributing content/code to an existing project

As you now know, GitHub hosts many open-licensed projects and by clicking the *fork* button, any GitHub user can instantaneously create their own fully independent copy of that project to work on without affecting the original. This copy or *forked* project can then be used to work out new features in a piece of software (or in our case, writing new content for a project website) that can eventually be merged back into the original project. Let's walk through a

simplified example using the example of contributing substantial edits to one of the pages of *DS Topic Guides*. Let's say you'd like to write up a new use case on one of the DS Topic Guides. That process may look a little like this:

1. Find the GitHub repository/source code behind the website

On the web version of *DS Topic Guides*, look for the GitHub logo in order to be taken to the GitHub repository behind it.

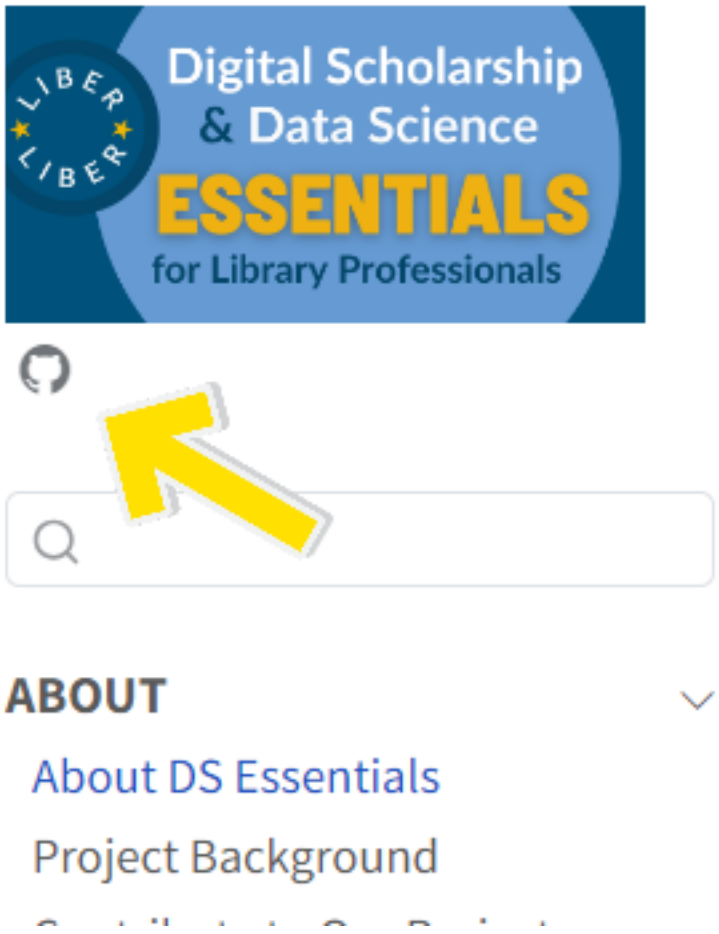

Figure 3: Screenshot showing where to find the github icon on a project website.

2. Make your own copy of the project (repository) to work on

Once on the project GitHub page you can fork your own copy of the repository so that you have a full copy of the project in your own GitHub account. You will now be owner of this new version and can make edits and changes as you please to the files without it affecting the main project. Just remember that eventually if you want your contributions to be merged with the original project those changes will need to be reviewed and approved by that maintainer.

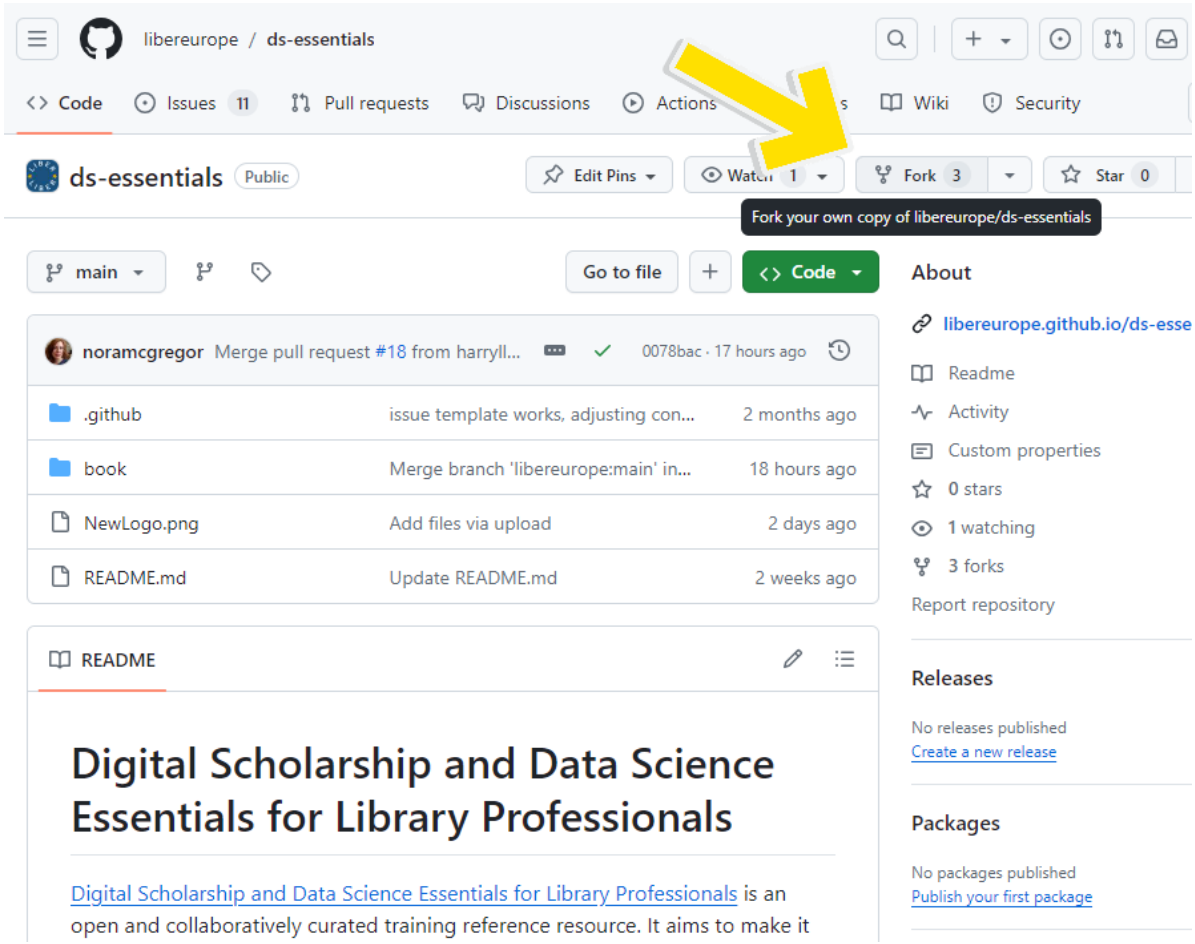

Figure 4: Screenshot of a github project with arrows showing where the Fork this repository button can be found.

3. Find the file you would like to change and make your edits

Navigate to the file you would like to edit in your own copy of the DSEssentials project and click the pencil icon to edit. (Note that one way you'll know you're working in your own copy is by seeing your personal GitHub username ahead of the repository name). Let's say the Topic Guide on AI & ML in Libraries Literacies needs an update. To find the right file in the directory you can do a search or look for a markdown file in the list which corresponds to the

URL on the website version and has the file extension .md. Note, because this website was built using Quarto, the file extension in this example is .qmd. For instance if ml-ai.html is the web version, you would look for ml-ai.qmd in the file directory on GitHub.

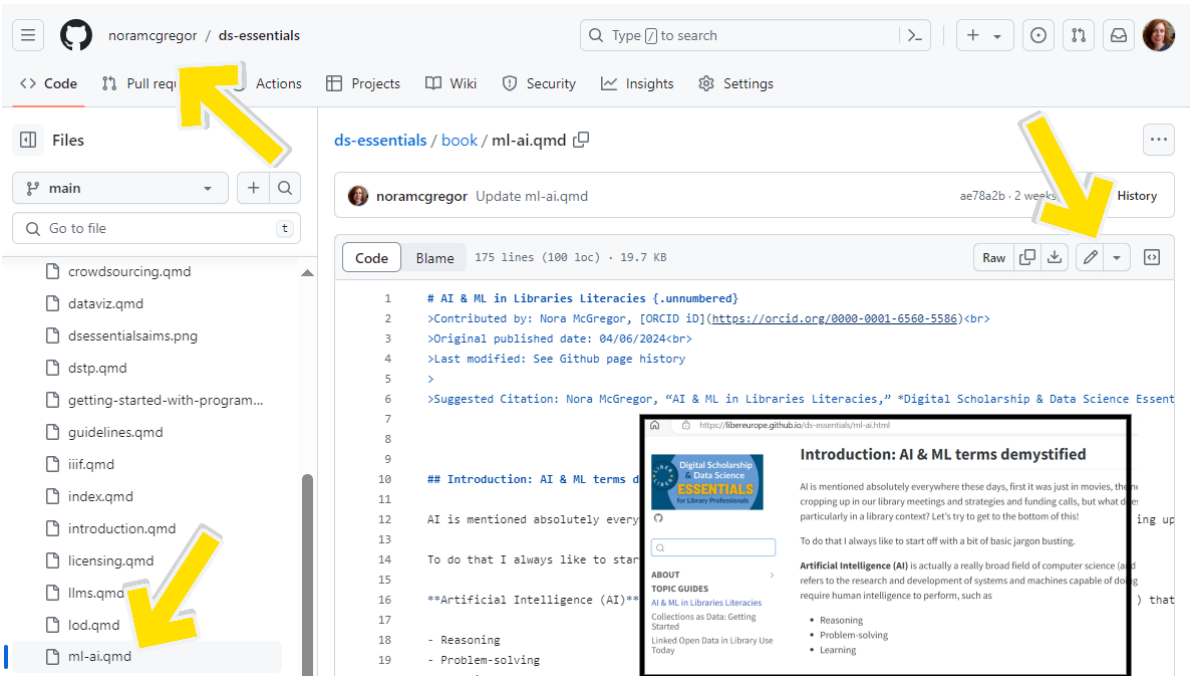

Figure 5: Screenshot of a forked github project with arrows pointing to how to find the edit button for a file within it.

Click on the little pencil icon to edit this file in markdown. Markdown is a simple system for marking sections of text that should be stylised in a certain way, e.g. made bold or appear in a list of bullet points. Markdown files have the file extension ".md" and can be edited in GitHub or with a text editor like Notepad. One common file in GitHub repositories that you will find in the markdown format is the "README.md" file which usually covers the purpose of the repository and how to get up and running.

Markdown works by using certain text characters to indicate styles. For example, placing a "#" in front of a line will make it a first level header. Placing "##" before the line makes it a second level header. Putting an asterisk around a piece of text, like this "*my text", will cause that text to be italicised as *my text*. You can see more markdown commands and experiment with it on the site Markdown Live Preview.

4. Commit your changes

Whenever you are working in a repository that you have forked, you are the Owner of this copy of the original project and can *commit* (save) any changes and edits you make to files directly to this copy of the project without needing any approval. Committing builds up a history of changes that you can roll back as needed along the way. Each time you commit you have an option of writing a comment as to what has changed which is a useful tool to use and habit to get into for going back in time if you need to reverse something. You can even refer to the related issue with `#<issue number>` in the commit message, and Github will cross reference the issue and the commit on the web interface.

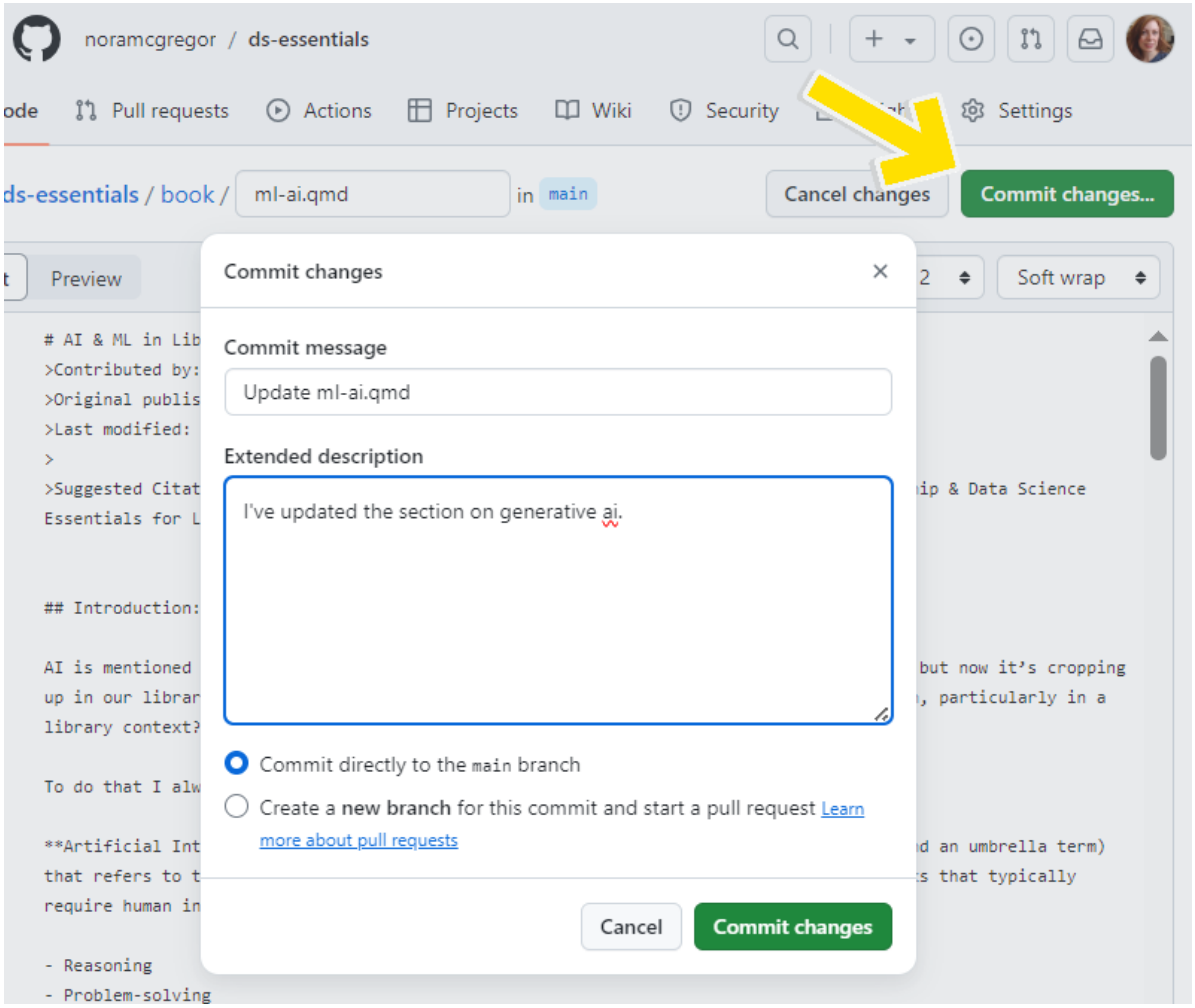

Figure 6: Screenshot of a file being edited within a github project with arrows showing where to commit the changes made.

5. Contribute your changes back to the original/main project

When you are finished with your changes and would like to share your new updated copy with the original project (in this example, incorporating your new text on Generative AI into the main *DS Topic Guide* on AI & ML in Libraries) this is when you will start a *pull request*. Navigate to the Pull Requests section of your forked project (repository) and click on "New pull request" to start that process.

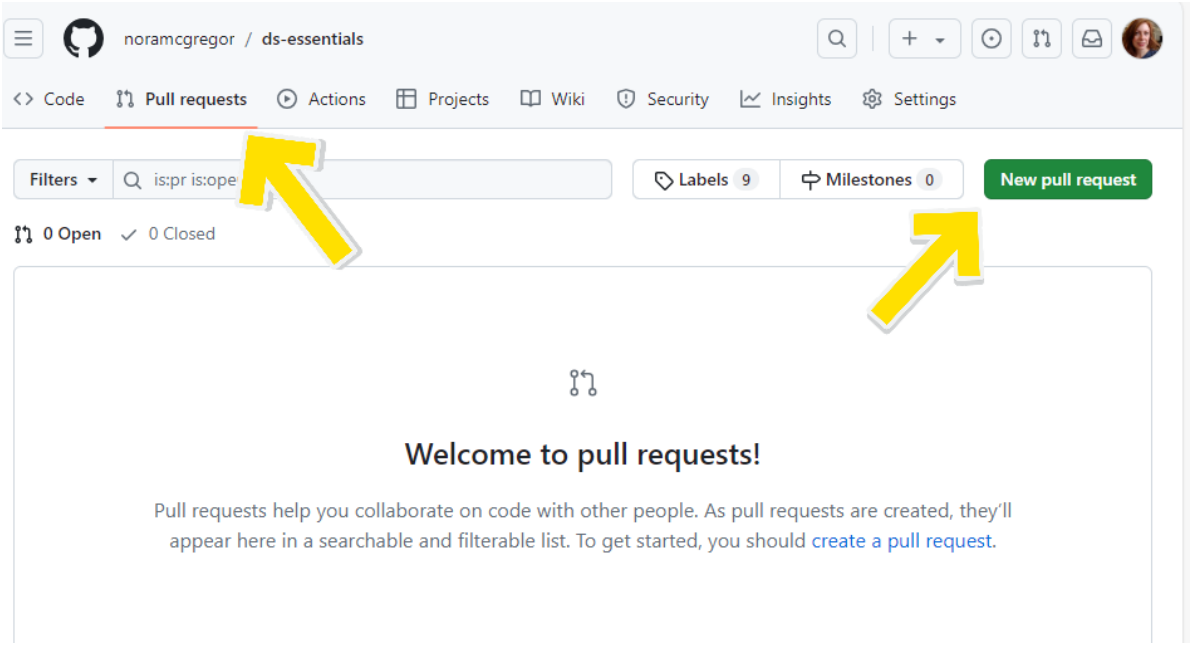

Figure 7: Screenshot of a github project with arrows showing where to start your Pull Request.

This will take you to back to the original project you forked from, where you can then formally "create a pull request". Once this has been made your pull request notifies the maintainers of the original project that you have made changes to their project that they may like to consider *merging* into theirs. Your pull request will show up under their project where you can discuss and review the changes you're suggesting. Once the review has been completed and everyone is happy with the changes suggested, the maintainer of the project will merge your pull request and you will have officially contributed (and your profile's icon will be listed among the contributors)!

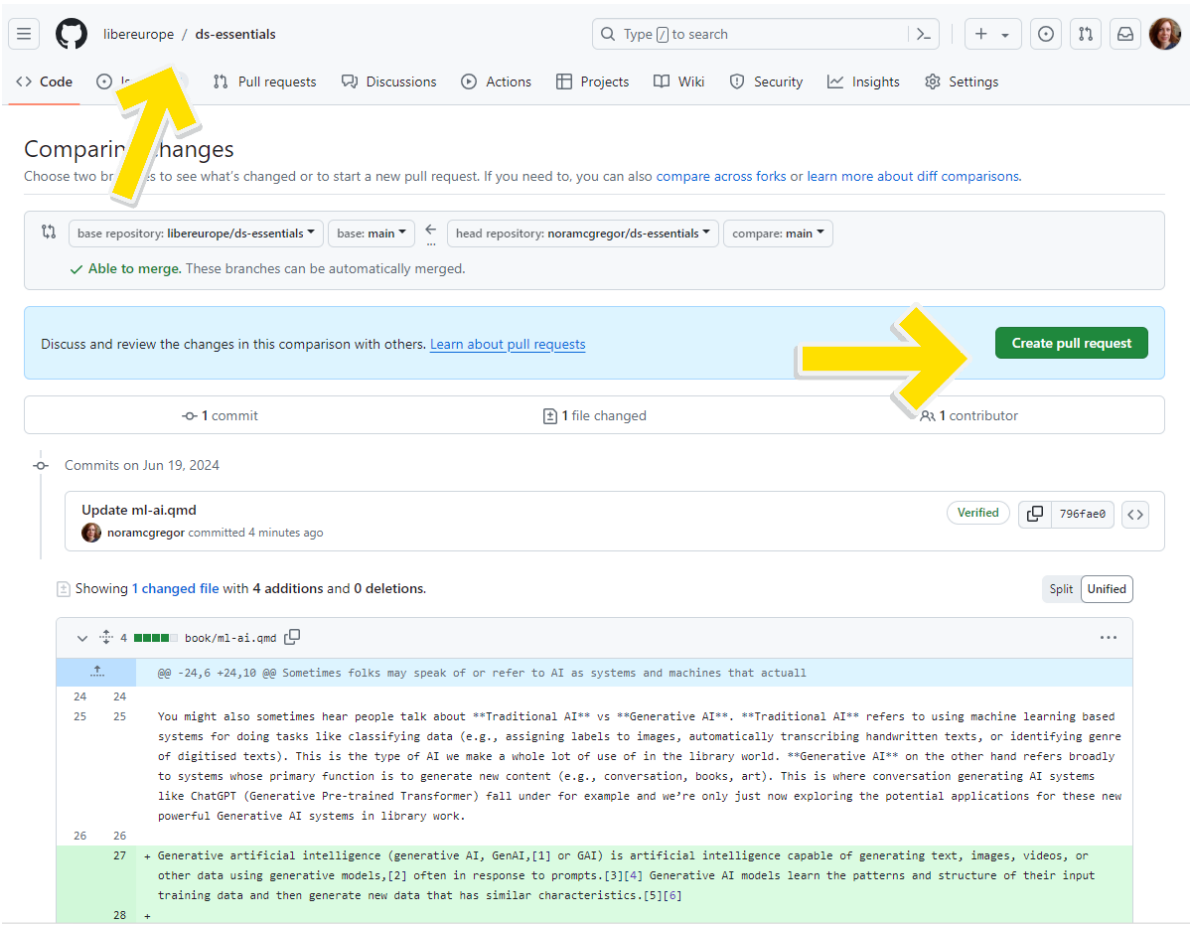

Figure 8: Screenshot of the original github project with arrows showing where to officially execute your Pull Request.

3. Re-using existing code/content for a new project

Let's say that instead of just contributing code/content to an existing project you want to make use of the whole project, such as a piece of open source software shared there. This is quite common for instance when looking to reuse software that someone has created and shared on GitHub. And it's a similar process to the above although in this case you might want to *clone* the repository rather than *fork* it as cloning allows you to save the whole repository directly to your local machine, rather than within the GitHub platform itself in order to implement it. Developers will find this much more practical than working in GitHub as they go through the process of installing, editing, adapting the code as needed to implement it. See for instance the example mentioned earlier of a variety of open source Omeka plugins being available for install from the Omeka GitHub repository.

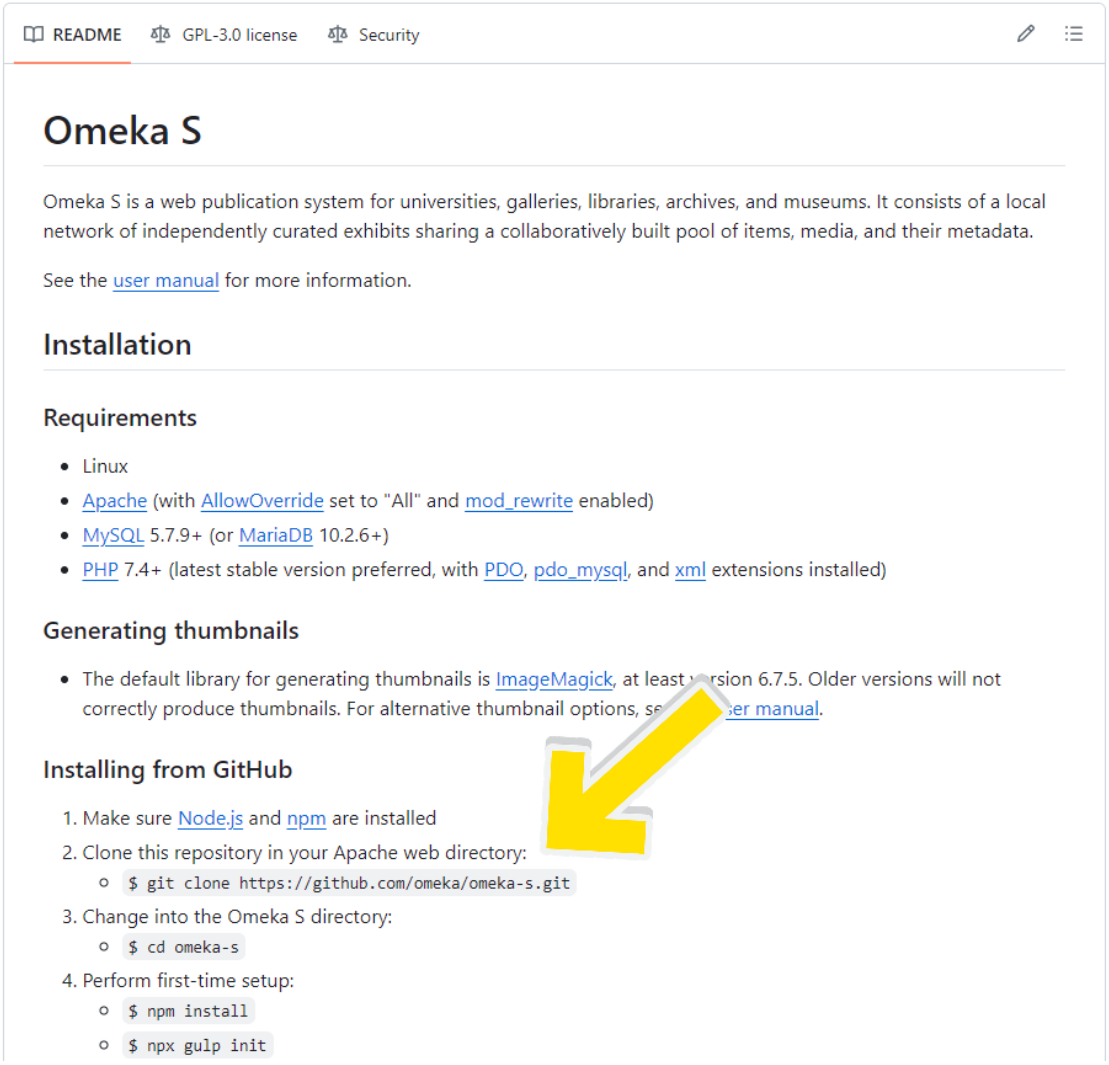

Figure 9: Screenshot of a github project showing where one might clone the project to re-use its software.

Hands-on activity and other self-guided tutorial(s)

Like many technologies, the best way to learn git or GitHub is to use it. Luckily there are lots of tutorials out there with ready-made exercises and walkthroughs if you don't have a clear purpose or aim to start with. Depending on your learning style, there are written lesson plans like the Carpentries' ones, or plenty of walkthrough videos covering everything from basic terminology to full workflow development. Many of these resources begin with a grounding in git, the locally-installed version control software. While not essential to using GitHub, this can

be useful to gain a broader understanding of the workings of both git and GitHub and maybe make sense of some of their additional functionalities. There are also specific lessons on various topics such as developing a static website using just GitHub and GitHub pages.

Our list below is not intended to be comprehensive but is a selection of what is out there and all have been used by library staff like ourselves to get to grips with Git!:

Learn the basics

- This official GitHub tutorial teaches you GitHub essentials like repositories, branches, commits, and pull requests.
- Another great tutorial over on GitHub uses the Spoon-Knife project, a test repository that's hosted on GitHub.com that lets you test the fork and pull request workflow.
- Git-for-librarians exercise on branching and one on forking

Making a website using GitHub

- Making a website with GitHub & GitHub Pages (lesson plan with video)
- Making a website with GitHub & Quarto / RStudio (video)
- Programming Historian - Building a static website with Jekyll and GitHub Pages
- Programming Historian - Running a collaborative research website and blog with GitHub Pages

Writing in Markdown (tools to try)

- Markdown Live Preview is a tiny web tool to preview Markdown formatted text.
- Word to MD is a useful tool for uploading word files and transforming it into markdown

As with most platforms, specific functions or tasks may change occasionally on GitHub. So you may find a lesson which refers to an old term or function that is no longer called what it used to be. GitHub documentation should provide up-to-date information on the current practice in most cases.

Recommended Reading/Viewing

- Library Carpentry Git & Version Control / Software Carpentry
- Push, Pull, Fork: GitHub for Academics (hybridpedagogy.org)
- A Comparative Analysis of the Use of GitHub by Librarians and Non-Librarians
- A reading list for librarians learning about Git and GitHub
- Week one reading for "Git and GitHub for Librarians" course.

Finding Communities of Practice

The best way to understand GitHub is to have a play yourself and if you get into trouble we've found that asking questions over on the GitHub Community Hub is a great way to get connected with other users. You can also join networks like AI4LAM and ask questions of members of the very engaged and helpful community on their slack channel.