The Congruence Engine - Training and Mutual Learning Working Group

Intro to Git-GitHub

Dr Anna-Maria Sichani
20 April 2023

TOWARDS A NATIONAL COLLECTION
UKRI
Arts and Humanities Research Council
DISCOVERY PROJECTS

SCIENCE MUSEUM GROUP

SCHOOL OF ADVANCED STUDY | UNIVERSITY OF LONDON

What is this training about?

Git and **GitHub** are powerful tools for collaborative and individual projects.

Git is a version control software that aids with tracking changes made to a set of files over time.

GitHub is a web-based platform for storing and sharing project files online.
This session begins with a conceptual overview of both tools, including an introduction to fundamental concepts such as version control. This session then covers initializing Git repositories, committing changes, pushing to GitHub, cloning repositories to your local machine, and forking repositories from other accounts on GitHub.

Resources this sessions has been using:

https://librarycarpentry.org/lc-git/
https://swcarpentry.github.io/git-novice/

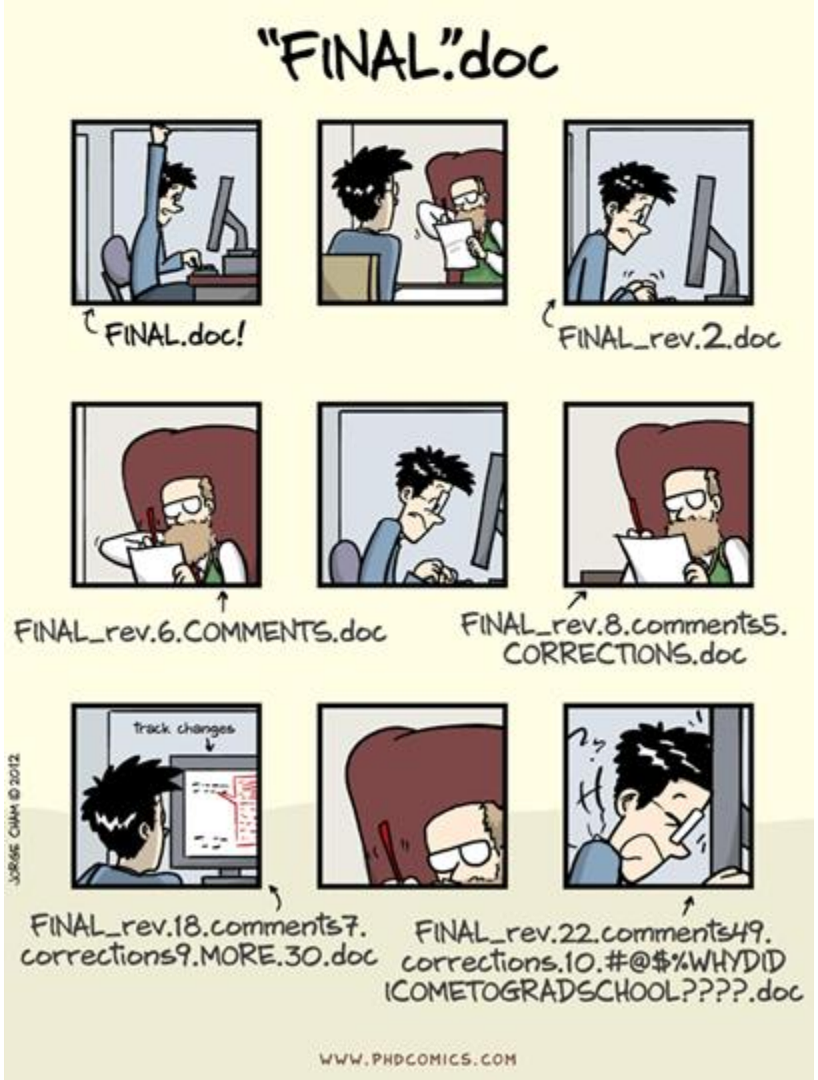
What is Version Control?

Version control is a name used for software which can help you record changes you make to the files in a directory on your computer. Version control software and tools (such as Git and Subversion/SVN) are often associated with software development, and increasingly, they are being used to **collaborate** in research and academic environments.
Version control systems work best with **plain text files** such as documents or computer code, but modern version control systems can be used to track changes in any type of file.

At its most basic level, version control software helps us **register and track sets of changes** made to files on our computer. We can then reason about and **share** those changes with others.

Benefits of using version control?

- **Collaboration** - Version control allows us to define formalized ways we can work together and share writing and code. For example merging together sets of changes from different parties enables co-creation of documents and software across distributed teams.
- **Versioning** - Having a robust and rigorous log of changes to a file, without renaming files (v1, v2, *final_copy*)
- **Rolling Back** - Version control allows us to quickly undo a set of changes. This can be useful when new writing or new additions to code introduce problems.
- **Understanding** - Version control can help you understand how the code or writing came to be, who wrote or contributed particular parts, and who you might ask to help understand it better.
- **Backup** - While not meant to be a backup solution, using version control systems mean that your code and writing can be stored on multiple other computers.



"Piled Higher and Deeper"
by Jorge Cham,
http://www.phdcomics.com

The Long History of Version Control Systems

- Automated version control systems are nothing new. Tools like RCS, CVS, or Subversion have been around since the early 1980s and are used by many large companies. However, many of these are now considered legacy systems (i.e., outdated) due to various limitations in their capabilities. More modern systems, such as Git and Mercurial, are *distributed*, meaning that they do not need a centralized server to host the repository. These modern systems also include powerful merging tools that make it possible for multiple authors to work on the same files concurrently.
- Version control systems start with a **base version of the document** and then record changes you make each step of the way. You can think of it as a **recording of your progress**: you can rewind to start at the base document and play back each change you made, eventually arriving at your more recent version.
- Once you think of changes as separate from the document itself, you can then think about "**playing back" different sets of changes on the base document,** ultimately resulting in different versions of that document. For example, two users can make independent sets of changes on the same document. Unless multiple users make changes to the same section of the document - a conflict - you can incorporate two sets of changes into the same base document.

- **Version control is like an unlimited 'undo'.**
- **Version control also allows many people to work in parallel.**

What are Git and GitHub?

We often hear the terms ***Git*** and ***GitHub*** used interchangeably but they are slightly different things.

Git is one of the most widely used **version control systems** in the world. It is a **free, open source tool that can be downloaded to your local computer/machine and used for logging all changes made to a group of designated computer files** (referred to as a "git repository" or "repo" for short) over time. The repository with your project's files is stored on your hard drive. You also edit the text files on your local machine using a plain text editor, which is another software on your local computer. It can be used to control file versions locally by you alone on your computer, but is perhaps most powerful when employed to coordinate simultaneous work on a group of files shared among distributed groups of people.

Git can be enabled in a folder, and then used to save the state of the contents in that folder at different points in the future, as designated by you. In the language of Git, a folder is called a ***repository*.** Using Git, you can view a log of the changes you've made to the files in a repository and compare changes over time. You can also revert back to previous versions, and create **"branches"** of a project to explore different futures.

- Git can be used for managing revisions to **any file type** on a computer system, including text documents and spreadsheets
- **But** can **not** version control for Word documents, as they contain special formatting, nor PDFs, though both file types can be stored in Git repositories.
- Once installed, interaction with Git is done through the **Command Prompt in Windows, or the Terminal on Mac/Linux.**

What are Git and GitHub?

GitHub on the other hand is a popular a cloud-based platform/**website for hosting and sharing Git repositories remotely** that you access through your internet browser. Even though you physically are still in the same place, working on your laptop, **you are no longer working on your local machine, you are on the Internet**. This is a fundamentally different location than when you're working with your Git repository and editing and creating files in your plain text editor. With GitHub, you are uploading your repository from your local machine to this platform on the Internet to be shared more broadly. You can also create private repositories if you want to use GitHub to backup a project.

It offers a **web interface** and provides **functionality and a mixture of both free and paid services for working with such repositories**. The majority of the content that GitHub hosts is open source software, blogs, and regularly updated text books. In addition to GitHub, there are **other Git hosting services** that offer many similar features such as GitLab, Bitbucket and Gitee. You can also use the GitHub Desktop app to work on your remote repositories **offline.**

It functions for some, predominantly programmers, as a **social network** for sharing and collaborating on code-based projects. Users can share their own projects, as well as search for others, which they can then often work on and contribute to.

What is Git?

Git is a distributed, open-source version control system. It enables users to track code, sync changes with a remote server, merge changes and revert to older versions. Due to its flexibility and popularity, Git has become an **industry standard** as it supports almost all development environments, command-line tools, and operating systems.

How does Git work?

Commits
Git stores your files and their development history in a local repository. Whenever you save changes you have made, Git creates a commit. **A commit is a snapshot of current files**. These commits are linked with each other, forming **a development history graph**. It allows us to revert back to the previous commit, compare changes, and view the progress of the development project. The commits are identified by a unique hash which is used to compare and revert the changes made.

Branches
The branches are copies of the source code that works parallel to the main version. To save the changes made, merge the branch into the main version. This feature promotes conflict-free teamwork. Each developer has his/her task, and by using branches, they can work on the new feature without the interference of other teammates. Once the task is finished, you can merge new features with the main version (master branch).

There are three states of files in Git: **modified, staged, and commit**. When you make changes in a file, the changes are saved in the local directory. To create a commit, you need to first stage changed files. You can add or remove changes in the staging area and then package these changes as a commit with a message describing the changes

What are the benefits of Git?

- **Track changes**: It allows developers to view historical changes. Development history makes it easy to identify and fix bugs.
- **IDE Integration**: Due to its popularity, Git integration is available in all development environments, for example VSCode and JupyterLab.
- **Team collaboration:** A developer team can view their progress, and by using branches, they can work individually on a task and merge changes with the main version. Pull requests, resolving merge conflicts, and code review promote team collaboration.
- **Distributed VSC:** In a distributed system, there is no centralized file storage. There are multiple backups for the same project. This approach allows developers to work offline and commit changes.

THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

Randall Munroe, xkcd (H/T Eric Kansa)

Before we start :

- Check you have install Git in your machine
- Check you have opened an editor (e.g. Notepad on Windows or TextEdit on Mac OSX).
- Check you have opened your **Command Prompt in Windows, or the Terminal on Mac/Linux.**

Ready?

git verb options

One of the main barriers to getting started with Git is understanding the terminology necessary to executing commands. Although some of the language used in Git aligns with common-use words in English, other terms are not so clear. The best way to learn Git terminology - which consists of a number of verbs such as add, commit and push (preceded by the word ‘git’) - is **to use it**, which is what we will be doing during this lesson. We will explain these commands as we proceed.

On a command line interface, Git commands are written as `git verb options`, where `verb` is what we actually want to do and `options` is additional optional information which may be needed for the `verb`. So let’s get started with our setup.

```
$ git config --global user.name "Your Name"
```

VERB OPTIONS

Note that whenever we use git via the command line, we need to preface each command (or verb) with `git`, so that the computer knows we are trying to get git to do something, rather than some other program.

Text editors

Let’s also set our **default text editor.** A text editor is necessary with some of your Git work, and the default from Git is Vim, which is a text editor that is hard to learn at first. Therefore, we recommend setting a simpler text editor in your Git configuration for this lesson (e.g. Notepad on Windows or TextEdit on Mac OSX).

Any text editor can be made default by adding the correct file path and command line options (see GitHub help). However, the simplest `core.editor` values are `"notepad"` on Windows, `"nano -w"` on Mac, and `"nano -w"` on Linux.

Setting up Git I & II

When we use Git on a new computer for the first time, we need to configure a few things. The basic elements of a configuration for Git are:

- your name and email address,
- what your preferred text editor is,
- and that we want to use these settings globally (i.e. for every project)

Check Your Installation: First, let's make sure Git has been successfully installed. In your terminal, type the following command:

```
$ git --version
```

If you see a version number, you're all set!

Configuring Git on Your Computer: First, we will tell Git our user name and email. For this lesson, we will be interacting with GitHub, and therefore we want to use the same email address we used when we set up our GitHub account. If you are concerned about privacy, please review GitHub's instructions for keeping your email address private.

Type these two commands into your shell, replacing `Your Name` and the email address with your own:

```
$ git config --global user.name "Your Name"
$ git config --global user.email "yourname@domain.name"
```

To check your set-up, type the following command into your terminal:

```
git config --list
```

Default text editor: The default from Git is Vim, which is a text editor that is hard to learn at first. Therefore, we recommend setting a simpler text editor in your Git configuration for this lesson.

```
$ git config --global core.editor "notepad"
```

Creating a repository & Using Git

A Git **repository** is a data structure used to track changes to a set of project files over time. Repositories are stored within the same directory as these project files, in a hidden directory called `.git`. Let's use the command line to create a git repository for the experiments that we're going to do today.

First, we will create a new directory for our project and enter that directory. We will explain commands as we go along.

```
$ mkdir hello-world
$ cd hello-world
```

We will now create an empty git repository to track changes to our project. To do this we will use the git **init** command, which is simply short for *initialise*.

```
$ git init
```

The `hello-world` directory is now a git repository.
If we use `ls` to show the directory's contents, it appears that nothing has changed:

```
$ ls
```

But if we add the `-a` flag to show everything, we can see that Git has created a hidden directory called `.git`:

```
$ ls -a
```

```
.          ..          .git
```

Adding, staging and committing changes

We will now create and save our first project file. This is a two-step process. First, we **add** any files for which we want to save the changes to a staging area, then we **commit** those changes to the repository. This two-stage process gives us fine-grained control over what should and should not be included in a particular commit.

Let's create a new file using the `touch` command, which is a quick way to create an empty file.

```
$ touch index.md
```

The `.md` extension above signifies that we have chosen to use the Markdown format, a lightweight markup language with plain text formatting syntax.

Let's check the status of our project again.

```
$ git status
```

This status is telling us that git has noticed a new file in our directory that we are not yet tracking. With colourised output, the filename will appear in red. To change this, and to tell Git we want to track any changes we make to `index.md`, we use `git add`.

```
$ git add index.md
```

This adds our Markdown file to the **staging area** (the area where git checks for file changes). To confirm this we want to use `git status` again.

```
$ git status
```

Adding, staging and committing changes

We will open the file `index.md` with any text editor we have at hand and enter `# Hello, world!`. The hash character is one way of writing a header with Markdown. Now, let's save the file within the text editor and check if Git has spotted the changes.

```
$ git status
```

This lets us know that git has indeed spotted the changes to our file, but that it hasn't yet staged them, so let's add the new version of the file to the staging area.

```
$ git add index.md
```

Now we are ready to **commit** our first changes. **Commit is similar to 'saving' a file to Git.** However, compared to saving, a commit provides a lot more information about the changes we have made, and this information will remain visible to us later.

```
$ git commit -m 'Add index.md'
```

We can see that one file has changed and that we made one insertion, which was a line with the text `'#Hello, world!'.` We can also see the commit message `'Add index.md',` which we added by using the `-m` flag after `git commit`. The commit message is used to record a short, descriptive, and specific summary of what we did to help us remember later on without having to look at the actual changes.

Having made a commit, we now have a permanent record of what was changed, and git has also recorded some additional metadata: who made the commit (you!) and when the commit was made (timestamp). You are building a **mini-history** of your process of working with the files in this directory.

A Metaphor for Adding and Committing

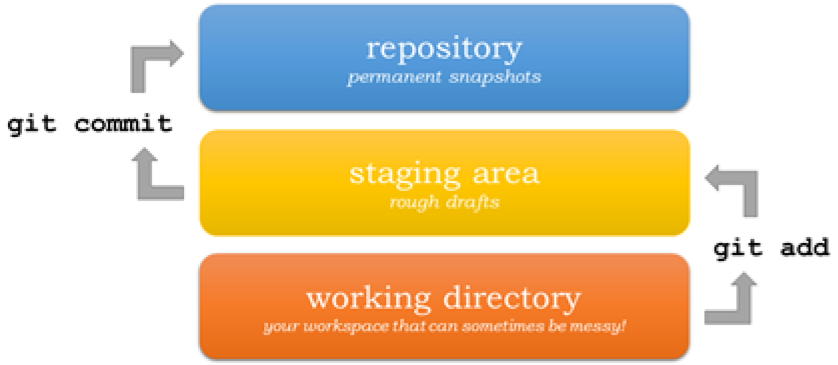
Git’s primary function is version control, or to track a project as it exists at particular points in time.
In Git, a *commit* is a snapshot of a repository that is entered into its permanent history. To commit a change to a repository, we take two steps:

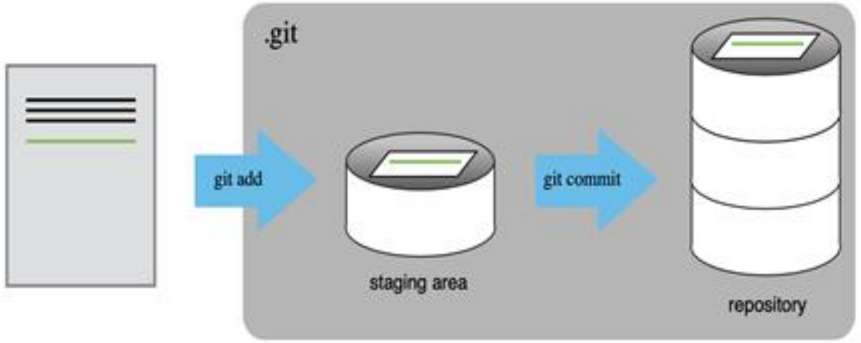
1. Adding files to a “staging area,” meaning that we intend to commit them.
2. Finalizing the commit.

Making a commit is a lot like taking a photo. First, you have to decide who will be in the photo and arrange your friends or family in front of the camera **(the staging process).** Staging a file or files is you telling Git, “Hey! Pay attention these files and the changes in them”. Once everyone is present and ready, you take the picture, entering that moment into the permanent record **(the commit process).**

*If you think of Git as taking snapshots of changes over the life of a project, `git add` specifies *what* will go in a snapshot (putting things in the staging area), and `git commit` then *actually takes* the snapshot, and makes a permanent record of it (as a commit). If you don’t have anything staged when you type `git commit`, Git will prompt you to use `git commit -a` or `git commit --all`, which is kind of like gathering *everyone* to take a group photo! However, it’s almost always better to explicitly add things to the staging area, because you might commit changes you forgot you made. (Going back to the group photo simile, you might get an extra with incomplete makeup walking on the stage for the picture).

A Metaphor for Adding and Committing





Why do you need both steps?

Sometimes when you're working on a project you don't want to pay attention to all the files you changed. Perhaps you fixed a bug in some code, but also did some work on your manuscript document. You may want to only commit the changes you made to the code because you still haven't finished your thoughts on the manuscript. You can stage, or add, the code file so Git knows to only commit the changes made to that file. Later, you can stage and then commit the manuscript changes on their own once you've finished your thought.

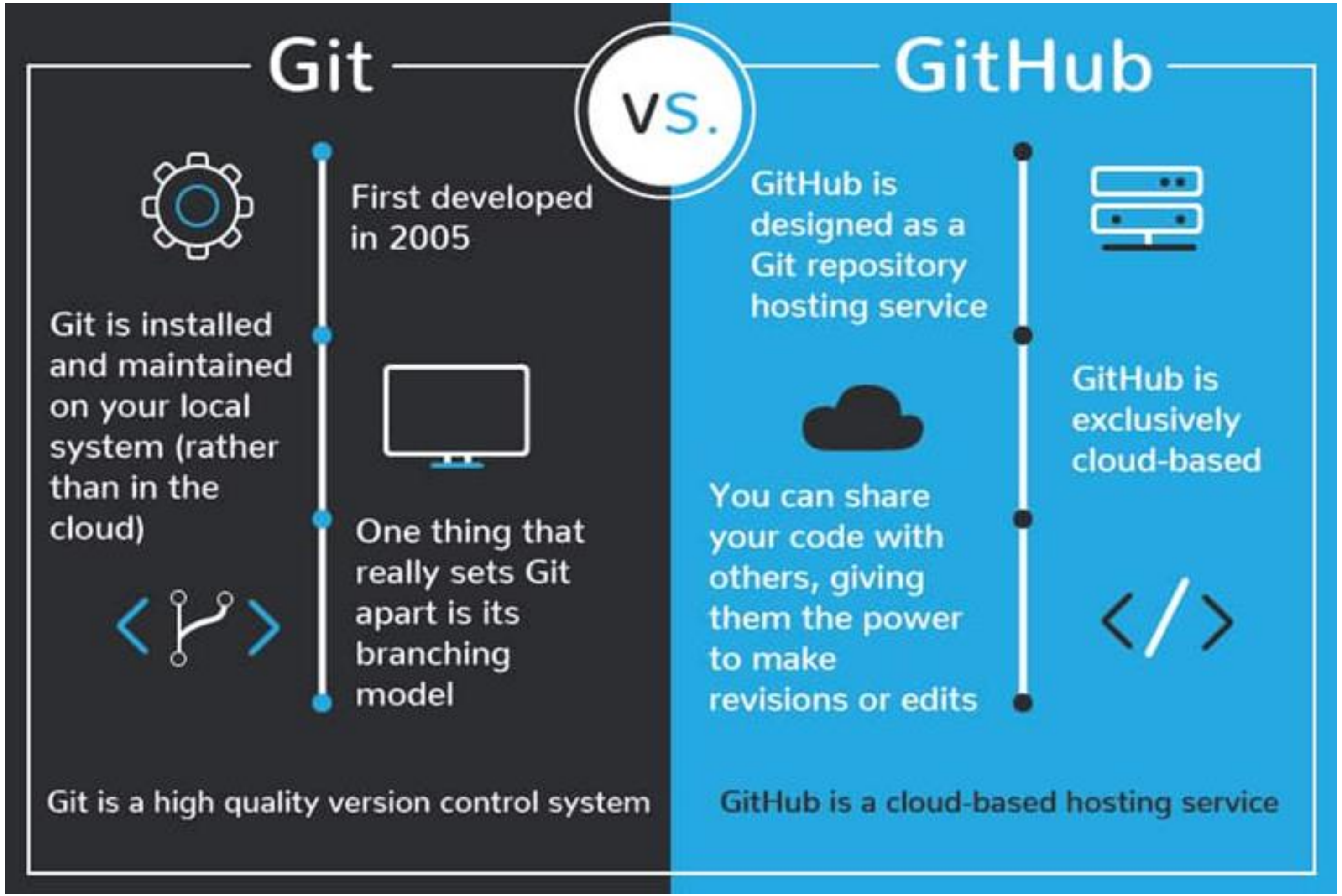
GitHub

Collaboration with GitHub

GitHub is a cloud software development platform. It is commonly used for saving files, tracking changes, and collaborating on development projects. In recent years, GitHub has become the most popular social platform for software development communities. Individuals can contribute to open-source projects and bug reports, discuss new projects and discover new tools. At the heart of GitHub is an open-source version control system (VCS) called Git. Git is responsible for everything GitHub-related that happens locally on your computer.

The power of sharing

- The real power of Git lies in being able to share your work with others and in being able to work collaboratively. The best way to do this is to use a remote hosting platform.
- Version control really comes into its own when we begin to collaborate with other people. We already have most of the machinery we need to do this; the only thing missing is to copy changes from one repository to another.
- Systems like Git allow us to move work between any two repositories. In practice, though, it’s easiest to use one copy as a central hub, and to keep it on the web rather than on someone’s laptop.
- Let’s start by sharing the changes we’ve made to our current project with the world. To this end we are going to create a *remote* repository that will be linked to our *local* repository.



Git VS. GitHub

Git

First developed in 2005

Git is installed and maintained on your local system (rather than in the cloud)

One thing that really sets Git apart is its branching model

Git is a high quality version control system

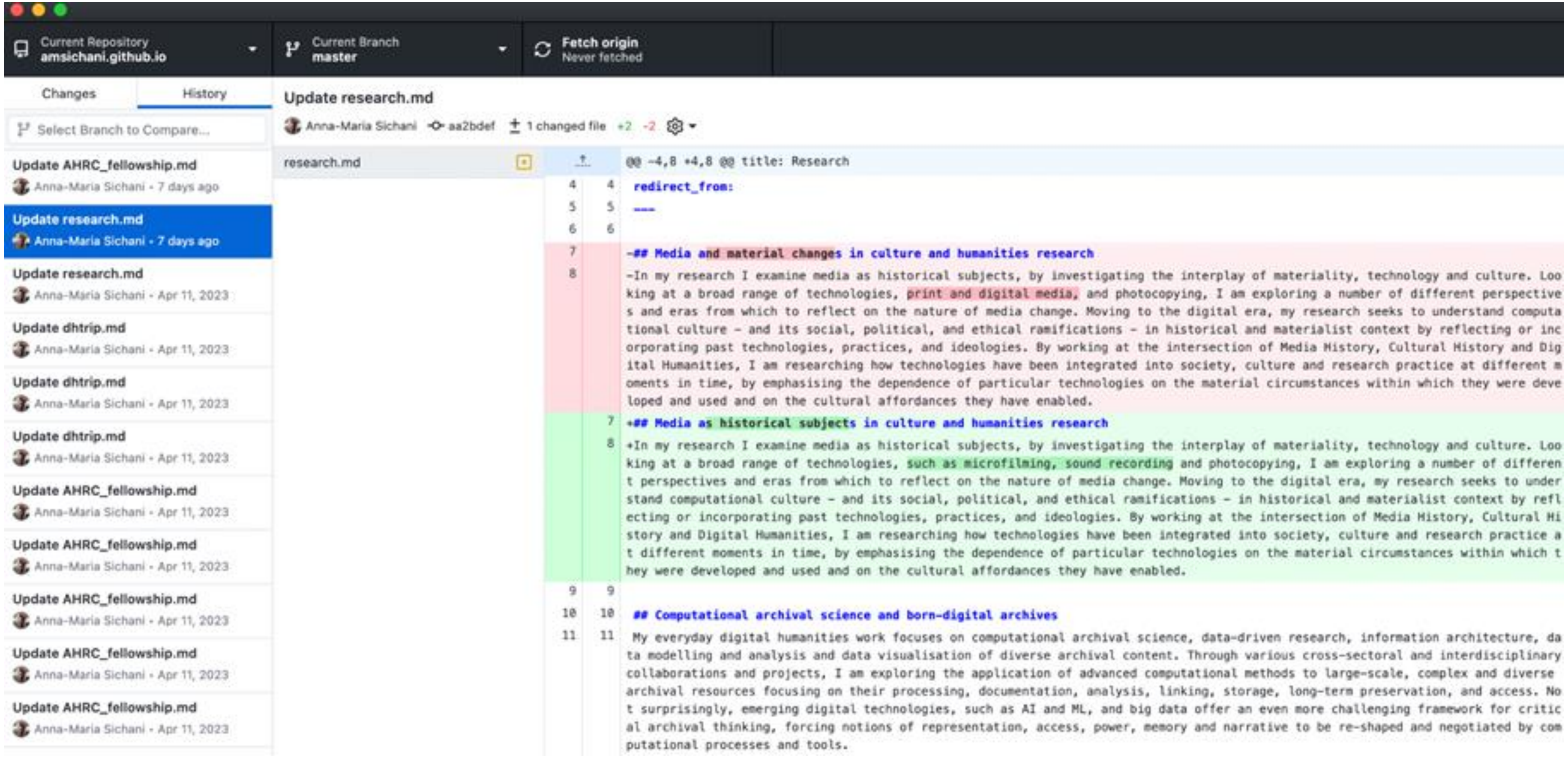
GitHub

GitHub is designed as a Git repository hosting service

GitHub is exclusively cloud-based

You can share your code with others, giving them the power to make revisions or edits

GitHub is a cloud-based hosting service



Update AHRC_fellowship.md
Anna-Maria Sichani • 7 days ago

Update research.md
Anna-Maria Sichani • 7 days ago

Update research.md
Anna-Maria Sichani • Apr 11, 2023

Update dhtrip.md
Anna-Maria Sichani • Apr 11, 2023

Update dhtrip.md
Anna-Maria Sichani • Apr 11, 2023

Update dhtrip.md
Anna-Maria Sichani • Apr 11, 2023

Update AHRC_fellowship.md
Anna-Maria Sichani • Apr 11, 2023

Update AHRC_fellowship.md
Anna-Maria Sichani • Apr 11, 2023

Update AHRC_fellowship.md
Anna-Maria Sichani • Apr 11, 2023

Update AHRC_fellowship.md
Anna-Maria Sichani • Apr 11, 2023

Update AHRC_fellowship.md
Anna-Maria Sichani • Apr 11, 2023

Update research.md

Anna-Maria Sichani · aa2bdef · 1 changed file +2 -2

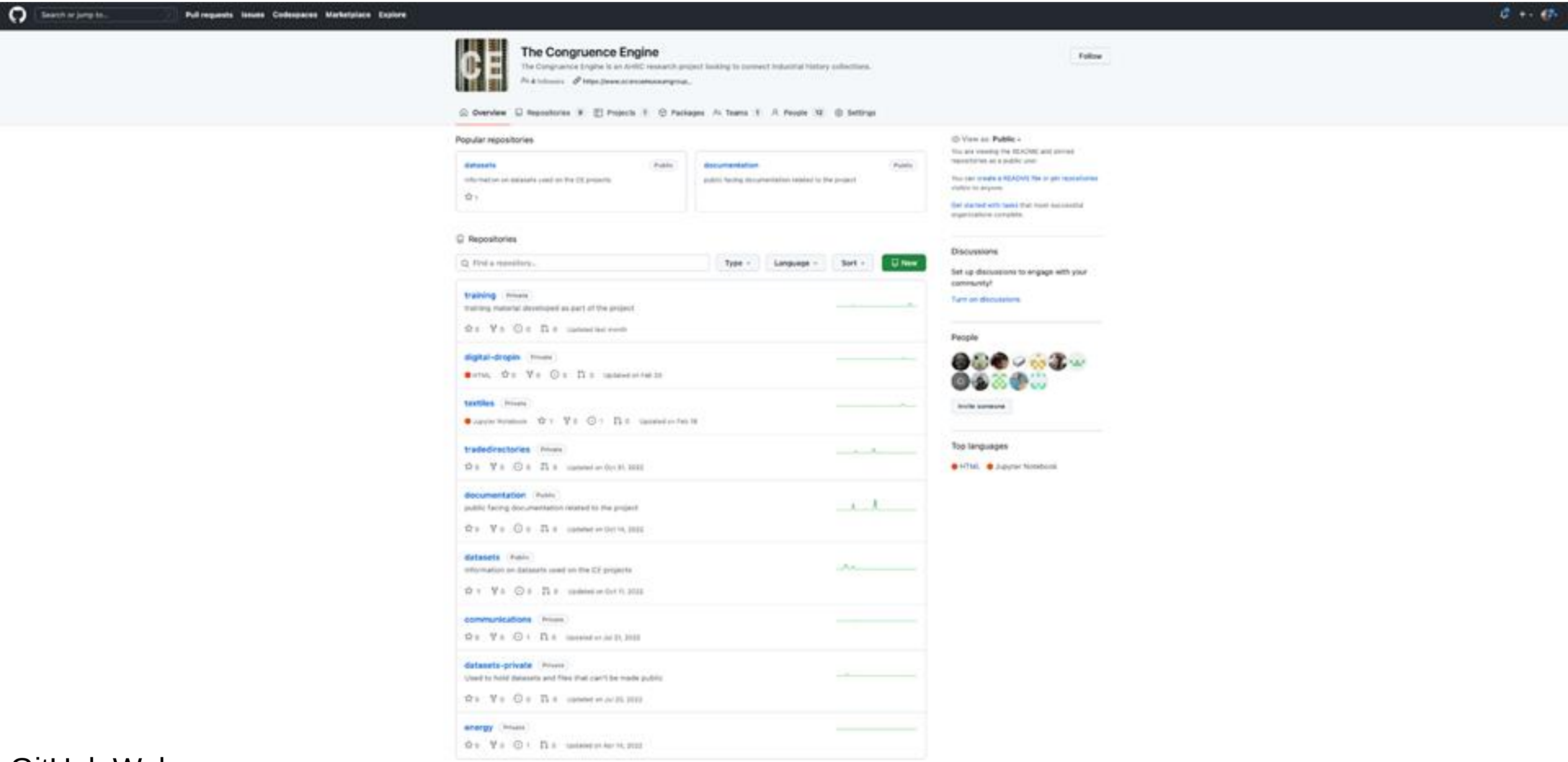
research.md

```
@@ -4,8 +4,8 @@ title: Research
 redirect_from:
 ---

-## Media and material changes in culture and humanities research
-In my research I examine media as historical subjects, by investigating the interplay of materiality, technology and culture. Loo
king at a broad range of technologies, print and digital media, and photocopying, I am exploring a number of different perspective
s and eras from which to reflect on the nature of media change. Moving to the digital era, my research seeks to understand computa
tional culture - and its social, political, and ethical ramifications - in historical and materialist context by reflecting or inc
orporating past technologies, practices, and ideologies. By working at the intersection of Media History, Cultural History and Dig
ital Humanities, I am researching how technologies have been integrated into society, culture and research practice at different m
oments in time, by emphasising the dependence of particular technologies on the material circumstances within which they were deve
loped and used and on the cultural affordances they have enabled.
+## Media as historical subjects in culture and humanities research
+In my research I examine media as historical subjects, by investigating the interplay of materiality, technology and culture. Loo
king at a broad range of technologies, such as microfilming, sound recording and photocopying, I am exploring a number of differen
t perspectives and eras from which to reflect on the nature of media change. Moving to the digital era, my research seeks to under
stand computational culture - and its social, political, and ethical ramifications - in historical and materialist context by refl
ecting or incorporating past technologies, practices, and ideologies. By working at the intersection of Media History, Cultural Hi
story and Digital Humanities, I am researching how technologies have been integrated into society, culture and research practice a
t different moments in time, by emphasising the dependence of particular technologies on the material circumstances within which t
hey were developed and used and on the cultural affordances they have enabled.

 ## Computational archival science and born-digital archives
 My everyday digital humanities work focuses on computational archival science, data-driven research, information architecture, da
ta modelling and analysis and data visualisation of diverse archival content. Through various cross-sectoral and interdisciplinary
collaborations and projects, I am exploring the application of advanced computational methods to large-scale, complex and diverse
archival resources focusing on their processing, documentation, analysis, linking, storage, long-term preservation, and access. No
t surprisingly, emerging digital technologies, such as AI and ML, and big data offer an even more challenging framework for critic
al archival thinking, forcing notions of representation, access, power, memory and narrative to be re-shaped and negotiated by com
putational processes and tools.
```

GitHub Desktop app



The Congruence Engine

The Congruence Engine is an AHRC research project looking to connect industrial history collections.

4 followers https://www.sciencemuseumgroup...

Popular repositories

datasets Public
information on datasets used on the CE projects
☆ 1

documentation Public
public facing documentation related to the project

Repositories

Find a repository...

training Private
training material developed as part of the project
☆ 0 0 0 0 Updated last month

digital-dropin Private
● HTML ☆ 0 0 0 0 Updated on Feb 23

textiles Private
● Jupyter Notebook ☆ 1 0 1 0 Updated on Feb 18

tradedirectories Private
☆ 0 0 0 0 Updated on Oct 31, 2022

documentation Public
public facing documentation related to the project
☆ 0 0 0 0 Updated on Oct 14, 2022

datasets Public
information on datasets used on the CE projects
☆ 1 0 0 0 Updated on Oct 11, 2022

communications Private
☆ 0 0 1 0 Updated on Jul 21, 2022

datasets-private Private
Used to hold datasets and files that can't be made public
☆ 0 0 0 0 Updated on Jul 20, 2022

energy Private
☆ 0 0 1 0 Updated on Apr 14, 2022

View as: Public

You are viewing the README and pinned repositories as a public user.

You can create a README file or pin repositories visible to anyone.

Get started with tasks that most successful organizations complete.

Discussions

Set up discussions to engage with your community?

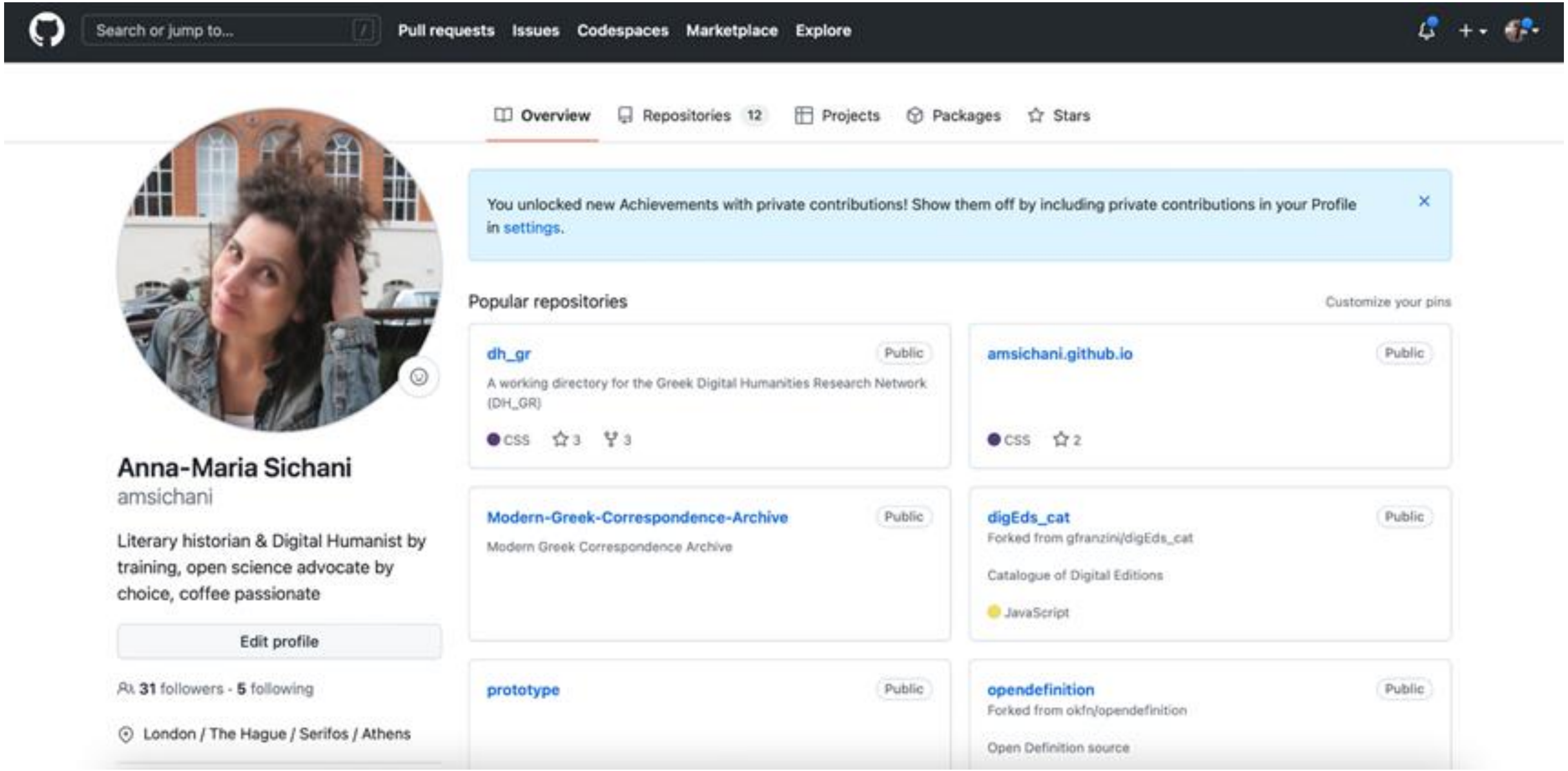
Turn on discussions

People

Top languages

● HTML ● Jupyter Notebook

GitHub Web



You unlocked new Achievements with private contributions! Show them off by including private contributions in your Profile in settings.

Popular repositories

Anna-Maria Sichani
amsichani

Literary historian & Digital Humanist by training, open science advocate by choice, coffee passionate

31 followers · 5 following

London / The Hague / Serifos / Athens

Repository	Visibility	Description	Language	Stars	Forks
dh_gr	Public	A working directory for the Greek Digital Humanities Research Network (DH_GR)	● CSS	3	3
amsichani.github.io	Public		● CSS	2	
Modern-Greek-Correspondence-Archive	Public	Modern Greek Correspondence Archive			
digEds_cat	Public	Forked from gfranzini/digEds_cat Catalogue of Digital Editions	● JavaScript		
prototype	Public				
opendefinition	Public	Forked from okfn/opendefinition Open Definition source			

GitHub Web

Create a remote repository on GitHub

Once we have logged in to GitHub, we can create a new repository by clicking the **+** icon in the upper-right corner of any page then selecting **New repository**.

Clicking New Repository will take you to a creation page with different options. For this workshop, we are not using any of the options available.

- Choose a name for your repository such as “`hello-world`”
- Enter a description, such as `Test for learning Git and GitHub.`
- Keep the Public — Anyone can see this repository selector checked.
- GitHub will ask if you want to add a `README.md,` license or a `.gitignore` file. Do not do any of that for now – We want you to start with a completely empty repository on GitHub, since you will be importing an existing repository from your computer.
- Click Create Repository button.

A repository (or repo) is essentially synonymous with the word “project.” Quite simply, a repository stores everything pertinent to a specific project including files, images, spreadsheets, data sets, and videos, often sorted into files.

Tips:

The README file: It is good practice to add a README file to each project to give a brief overview of what the project is about. If you put your README file in your repository’s root directory, GitHub will recognize and automatically surface your README to repository visitors

.gitignore file : Configuring ignored files for a single repository
You can create a *.gitignore* file in your repository's root directory to tell Git which files and directories to ignore when you make a commit. To share the ignore rules with other users who clone the repository, commit the *.gitignore* file in to your repository.GitHub maintains an official list of recommended *.gitignore* files for many popular operating systems, environments, and languages in the `github/gitignore` public repository. You can also use gitignore.io to create a *.gitignore* file for your operating system, programming language, or IDE.

licences: When a repository with source code, a manuscript or other creative works becomes public, it should include a file `LICENSE` or `LICENSE.txt` in the base directory of the repository that clearly states under which license the content is being made available. This is because creative works are automatically eligible for intellectual property (and thus copyright) protection. Reusing creative works without a license is dangerous, because the copyright holders could sue you for copyright infringement.

Markdown: Markdown is an easy-to-read, easy-to-write syntax for formatting plain text (GitHub Flavored Markdown). You can also use HTML tags if you wish.

Connect local to remote repository

We created a local repository on our own computer and just now we have also created a remote repository on GitHub. But at this point, the two are completely isolated from each other. We want to link them together to synchronize them and share our project with the world. To make this connection, we want to tell our local repository that GitHub is the `remote` repository. In order to do that we need the information that GitHub displays in the "Quick setup" box on this page. We do this by making the GitHub repository a remote for the local repository. The home page of the repository on GitHub includes the URL string we need to identify it.

1) We will use the **Secure Shell Protocol (SSH) f**or this lesson, so please make sure that button shows that it is selected (gray highlight) and that the address in the text box starts with git@github.
2) To connect the repository on our own computer (local) to the repository we just created on GitHub, we will use the commands provided by GitHub in the box with the heading "…or push an existing repository from the command line."

Move back to your shell application and enter the first command:

```
$ git remote add origin git@github.com:yourname/hello-world.git
```

- Make sure to use the URL for your actual repository user name rather than `yourname`: the only difference should be your username instead of `yourname`.
- `remote add` is the command we use to configure a remote repository, e.g., another Git repository that contains the same content as our local repository, but that is not on our computer.
- `origin` is the nickname we're telling our local machine to use to for the following long web address. After we enter this command, we can use `origin` to refer to this specific repository in GitHub instead of the URL.

We can check that it is set up correctly with the command: `$ git remote -v`

Push local changes to a remote repository

Now we have established a connection between the two repositories, but we still haven't synchronized their content, so the remote repository is still empty. To fix that, we will have to "push" our local changes to the GitHub repository. We do this using the `git push` command:

```
$ git push -u origin main
```

The nickname of our remote repository is "origin" and the default local branch name is "main". The `-u` flag tells git to remember the parameters, so that next time we can simply run `git push` and Git will know what to do ("pushing changes `upstream` to Github").

You may be prompted to enter your GitHub username and password to complete the command.

When we do a `git push`, we will see Git 'pushing' changes upstream to GitHub. Because our file is very small, this won't take long but if we had made a lot of changes or were adding a very large repository, we might have to wait a little longer. We can check where we're at with `git status`.

We can use the `git diff` command to see changes we have made before making a commit. Open `index.md` with any text editor and enter some text on a new line, for instance "A new line" or something else. We will then use `git diff` to see the changes we made:

We can now commit these changes:

```
$ git add index.md
```

```
$ git commit -m 'Add another line'
```

Pulling changes

When working with others, or when we’re making our own changes from different machines, we need a way of pulling those remote changes back into our local copy. For now, we can see how this works by making a change on the GitHub website and then ‘pulling’ that change back to our computer.

Let’s go to our repository in GitHub and make a change. Underneath where our `index.md` file is listed you will see a button to ‘Add a README’. Do this now, entering whatever you like, scrolling to the bottom and clicking ‘Commit new file’ (The default commit message will be ‘Create README.md’, which is fine for our purposes). Our local repository is now out of sync with our remote repository, so let’s fix that by pulling the remote changes into our local repository using the `git pull` command.

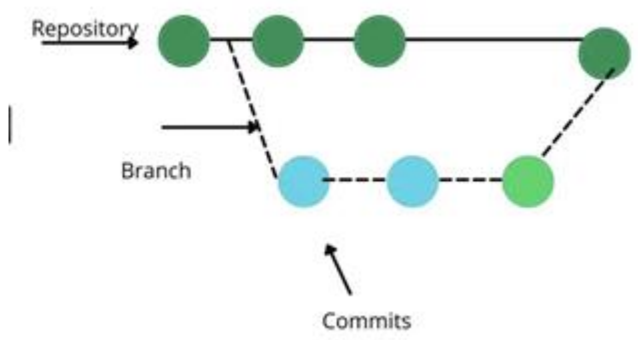
```
$ git pull
```

Create a Branch

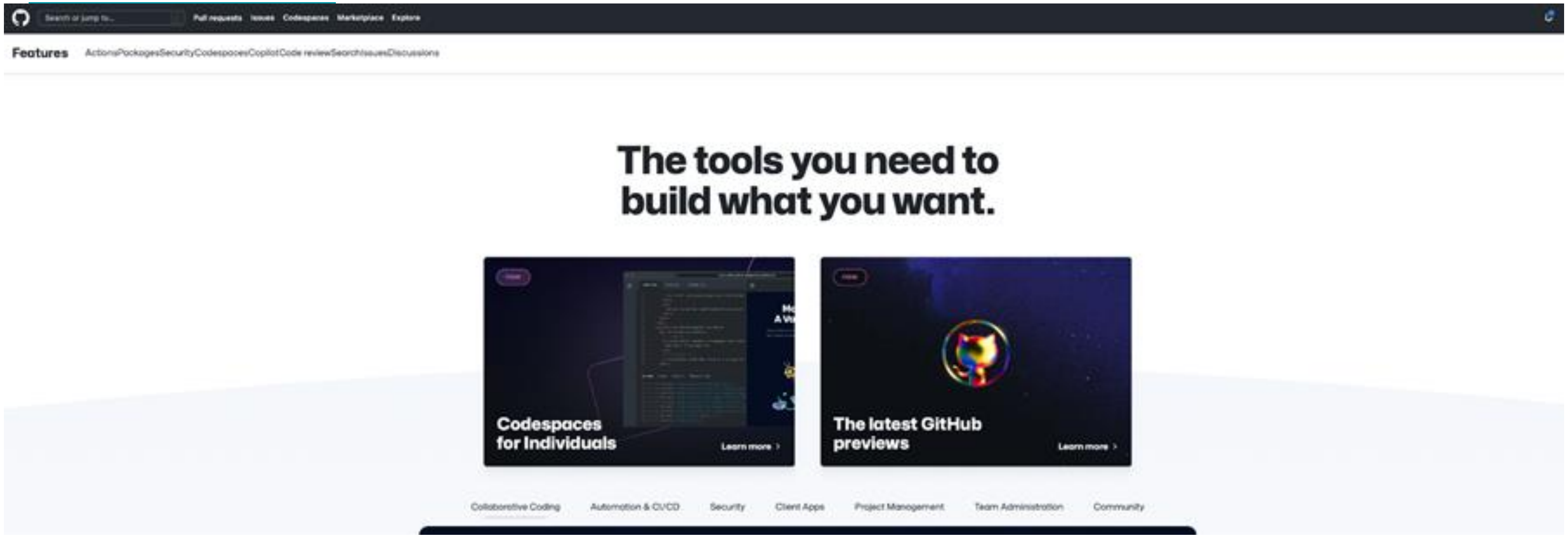
Projects are multi-faceted and many program versions are required when you're building. Branching enables you to edit multiple unique versions of a repository at once. Your repository automatically has a definitive branch called **master**. You can work on several different branches in order to make edits before eventually committing them to the master branch.

When a new branch is started, it'll be a copy of the master branch until you edit it to make new changes. A branch typically goes through many steps and approvals before it is ever merged into the master branch.

To start a new branch in GitHub, navigate to your new repository, click the dropdown that reads "branch: master," type a branch name (like README-edits), and then hit "create branch." Branches are ideal for new features or bug fixes.



Github features


Search or jump to...
Pull requests Issues Codespaces Marketplace Explore

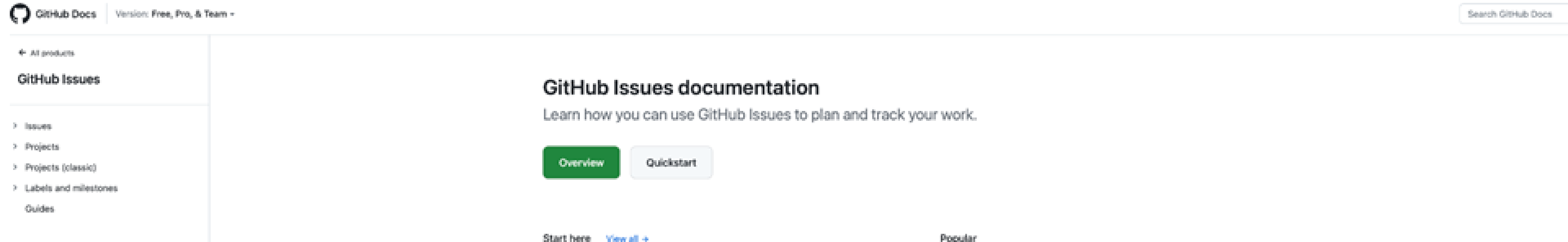
Features ActionsPackagesSecurityCodespacesCopilotCode reviewSearchIssuesDiscussions

The tools you need to build what you want.

Codespaces for Individuals Learn more ›

The latest GitHub previews Learn more ›

Collaborative Coding · Automation & CI/CD · Security · Client Apps · Project Management · Team Administration · Community


GitHub Docs | Version: Free, Pro, & Team ▾ | Search GitHub Docs

← All products

GitHub Issues

- › Issues
- › Projects
- › Projects (classic)
- › Labels and milestones
- Guides

GitHub Issues documentation

Learn how you can use GitHub Issues to plan and track your work.

Overview | Quickstart

Start here View all →

Popular

Open & merge a Pull Request

In order for any branch to be merged into another person's branch, you must open a pull request. A pull request is GitHub's way of notifying relevant parties about **your request to incorporate changes into their branch**. A pull request will show in `red` and `green` the differences of the content between branches. You can make a pull request any time you complete a commit. For best results, when sending a pull request, you can use the "@" feature to mention specific people from whom you need feedback.

To open a pull request, you'll go to the "pull request" tab and hit the button that says "new pull request." Next, in the "example comparisons" box, find the branch you made and compare it with the master. Ensure you like the changes and then click the "create pull request" button. Title your pull request and briefly describe the changes. To finish, click "create pull request."

Merging your pull request with the master branch is something you may need to pass on to your superiors to handle. For the sake of learning, though, you can practice doing it yourself. Simply hit the button that says "merge pull request," select "confirm merge," and then delete the branch you merged once it has been incorporated into the master.

Pull requests

Pull requests let you tell others about changes you've pushed to a branch in a repository on GitHub. Once a pull request is opened, you can discuss and review the potential changes with collaborators and add follow-up commits before your changes are merged into the base branch.
After initializing a pull request, you'll see a review page that shows a high-level overview of the changes between your branch (the compare branch) and the repository's base branch. You can add a summary of the proposed changes, review the changes made by commits, add labels, milestones, and assignees, and @mention individual contributors or teams.

Once you've created a pull request, you can push commits from your topic branch to add them to your existing pull request. These commits will appear in chronological order within your pull request and the changes will be visible in the "Files changed" tab.

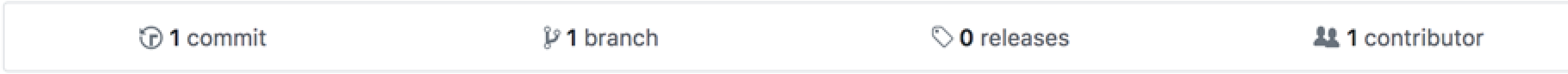
Other contributors can review your proposed changes, add review comments, contribute to the pull request discussion, and even add commits to the pull request. By default, in public repositories, any user can submit reviews that approve or request changes to a pull request. Organization owners and repository admins can limit who is able to give approving pull request reviews or request changes.

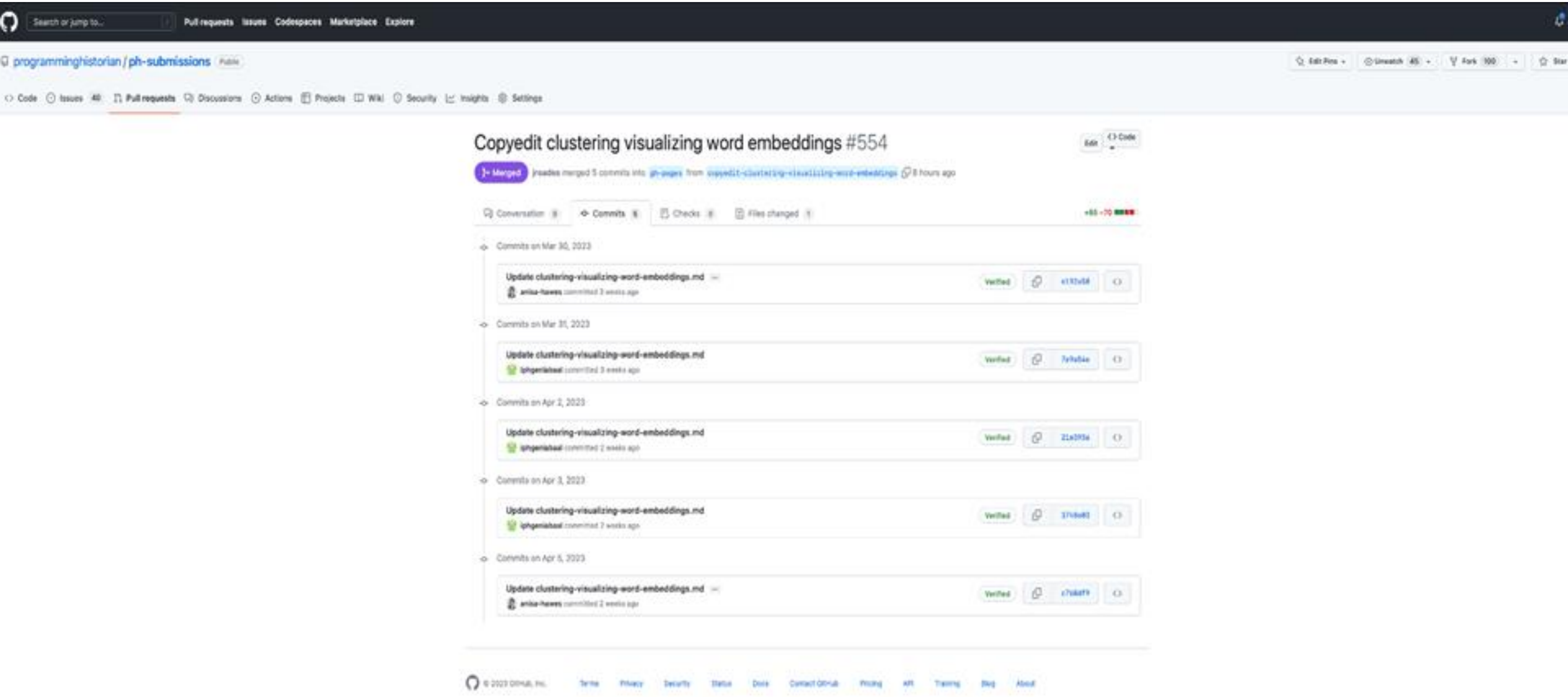
After you're happy with the proposed changes, you can merge the pull request. If you're working in a shared repository model, you create a pull request and you, or someone else, will merge your changes from your feature branch into the base branch you specify in your pull request.

Reviews allow collaborators to comment on the changes proposed in pull requests, approve the changes, or request further changes before the pull request is merged. Repository administrators can require that all pull requests are approved before being merged.

Create and Commit Changes to a Branch

To make changes to a branch in GitHub, go to the code view for your newly created branch. Click the file you want to change, then hit the pencil icon in the upper right, make any necessary edits, describe your changes by writing a commit message, and then click “commit changes.” Each saved change is called a commit. Every individual commit has its own commit message which gives more details into why a specific change was done. The commit messages give a history of changes and help project contributors understand how the project has changed over time.





Copyedit clustering visualizing word embeddings #554

Merged jrladd merged 5 commits into gh-pages from copyedit-clustering-visualizing-word-embeddings 8 hours ago

Commits on Mar 30, 2023

Update clustering-visualizing-word-embeddings.md
anisa-hawes committed 3 weeks ago
Verified e192a58

Commits on Mar 31, 2023

Update clustering-visualizing-word-embeddings.md
iphgeniabal committed 3 weeks ago
Verified 7a9a84e

Commits on Apr 2, 2023

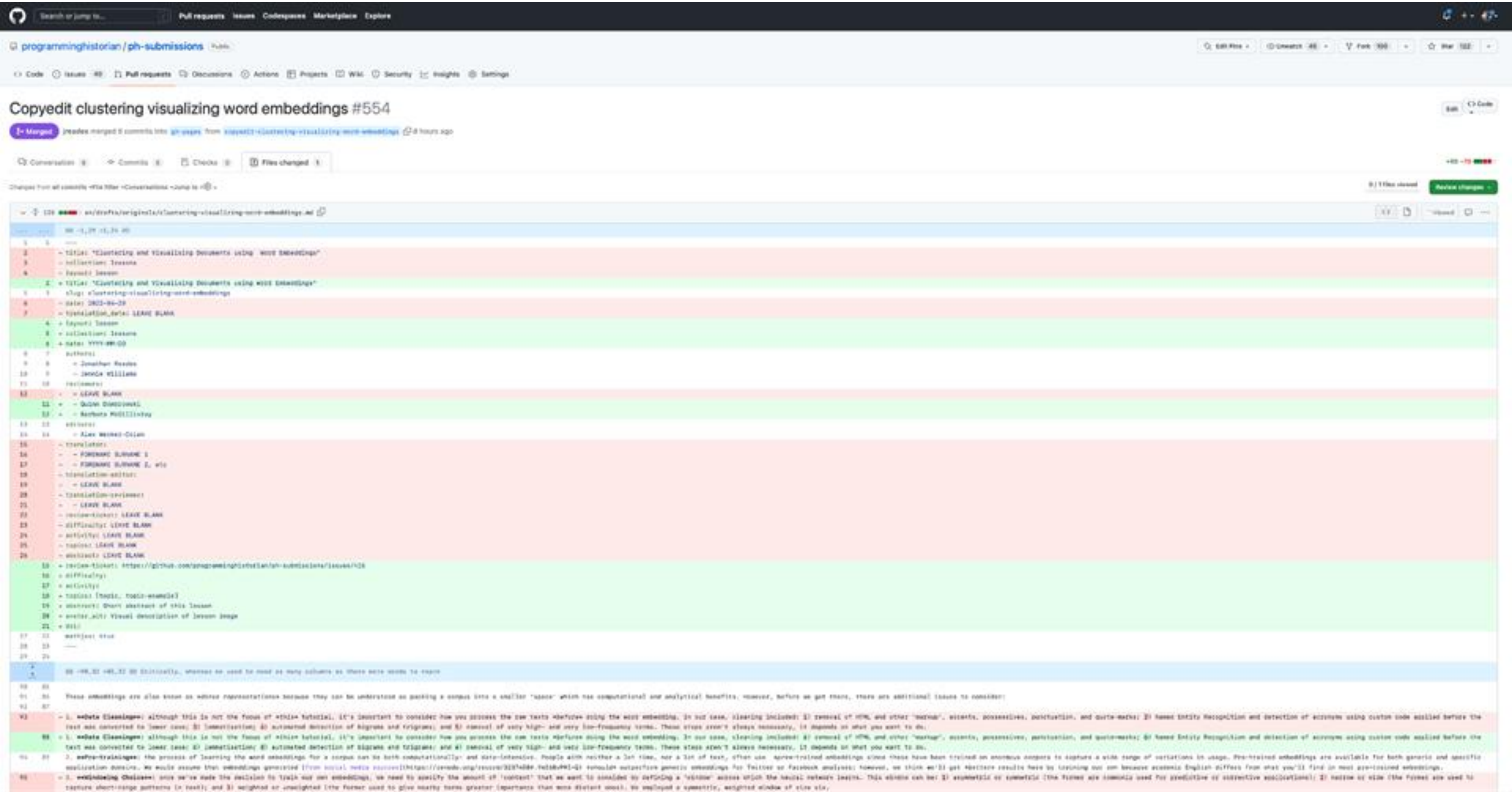
Update clustering-visualizing-word-embeddings.md
iphgeniabal committed 2 weeks ago
Verified 21a093e

Commits on Apr 3, 2023

Update clustering-visualizing-word-embeddings.md
iphgeniabal committed 2 weeks ago
Verified 37c6e83

Commits on Apr 5, 2023

Update clustering-visualizing-word-embeddings.md
anisa-hawes committed 2 weeks ago
Verified c7b8ef9


Search or jump to...
Pull requests Issues Codespaces Marketplace Explore
programminghistorian / ph-submissions Public
Code Issues Pull requests Discussions Actions Projects Wiki Security Insights Settings

Copyedit clustering visualizing word embeddings #554

Merged jreades merged 8 commits into gh-pages from copyedit-clustering-visualizing-word-embeddings 8 hours ago

Conversation Commits Checks Files changed 1
Review changes

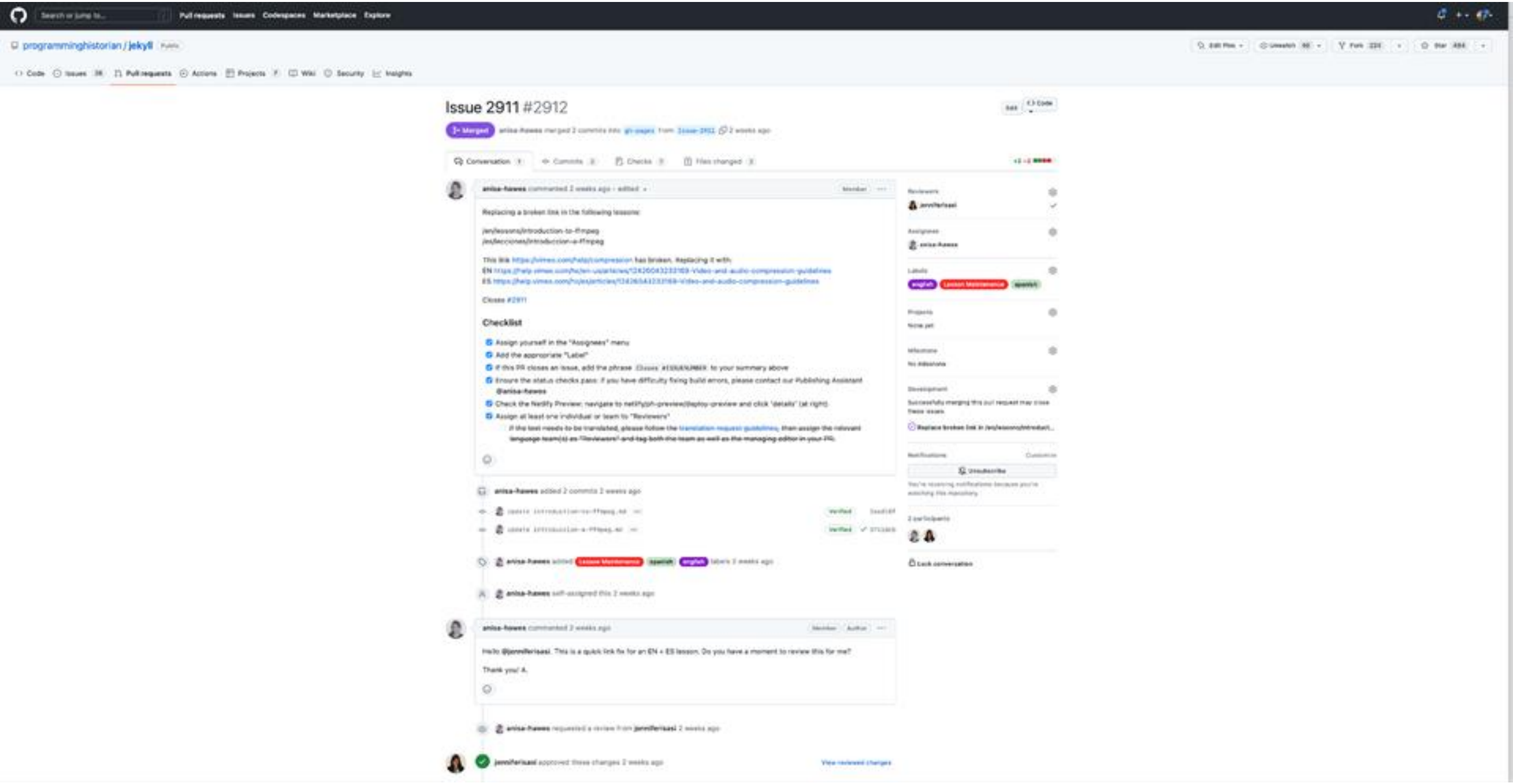
en/drafts/originals/clustering-visualizing-word-embeddings.md

```
@@ -1,29 +1,24 @@
  ---
- title: "Clustering and Visualising Documents using  Word Embeddings"
- collection: lessons
- layout: lesson
+ title: "Clustering and Visualising Documents using Word Embeddings"
  slug: clustering-visualizing-word-embeddings
- date: 2022-06-29
- translation_date: LEAVE BLANK
+ layout: lesson
+ collection: lessons
+ date: YYYY-MM-DD
  authors:
    - Jonathan Reades
    - Jennie Williams
  reviewers:
-   - LEAVE BLANK
+   - Quinn Dombrowski
+   - Barbara McGillivray
  editors:
    - Alex Wermer-Colan
- translator:
-   - FORENAME SURNAME 1
-   - FORENAME SURNAME 2, etc
- translation-editor:
-   - LEAVE BLANK
- translation-reviewer:
-   - LEAVE BLANK
- review-ticket: LEAVE BLANK
- difficulty: LEAVE BLANK
- activity: LEAVE BLANK
- topics: LEAVE BLANK
- abstract: LEAVE BLANK
+ review-ticket: https://github.com/programminghistorian/ph-submissions/issues/416
+ difficulty:
+ activity:
+ topics: [topic, topic-example]
+ abstract: Short abstract of this lesson
+ avatar_alt: Visual description of lesson image
+ doi:
  mathjax: true
  ---

@@ -90,32 +85,32 @@ Critically, whereas we used to need as many columns as there were words to repre

  These embeddings are also known as *dense representations* because they can be understood as packing a corpus into a smaller 'space' which has computational and analytical benefits. However, before we get there, there are additional issues to consider:

- 1. **Data Cleaning**: although this is not the focus of *this* tutorial, it's important to consider how you process the raw texts *before* doing the word embedding. In our case, cleaning included: 1) removal of HTML and other 'markup', accents, possessives, punctuation, and quote-marks; 2) Named Entity Recognition and detection of acronyms using custom code applied before the text was converted to lower case; 3) lemmatisation; 4) automated detection of bigrams and trigrams; and 5) removal of very high- and very low-frequency terms. These steps aren't always necessary, it depends on what you want to do.
+ 1. **Data Cleaning**: although this is not the focus of *this* tutorial, it's important to consider how you process the raw texts *before* doing the word embedding. In our case, cleaning included: 1) removal of HTML and other 'markup', accents, possessives, punctuation, and quote-marks; 2) Named Entity Recognition and detection of acronyms using custom code applied before the text was converted to lower case; 3) lemmatisation; 4) automated detection of bigrams and trigrams; and 5) removal of very high- and very low-frequency terms. These steps aren't always necessary, it depends on what you want to do.
  2. **Pre-Training**: the process of learning the word embeddings for a corpus can be both computationally- and data-intensive. People with neither a lot time, nor a lot of text, often use  *pre-trained* embeddings since these have been trained on enormous corpora to capture a wide range of variations in usage. Pre-trained embeddings are available for both generic and specific application domains. We would assume that embeddings generated [from social media sources](https://zenodo.org/record/3237458#.YeS6bvPMI-Q) *should* outperform generic embeddings for Twitter or Facebook analysis; however, we think we'll get *better* results here by training our own because academic English differs from what you'll find in most pre-trained embeddings.
- 3. **Windowing Choices**: once we've made the decision to train our own embeddings, we need to specify the amount of 'context' that we want to consider by defining a 'window' across which the neural network learns. This window can be: 1) asymmetric or symmetric (the former are commonly used for predictive or subjective applications); 2) narrow or wide (the former are used to capture short-range patterns in text); and 3) weighted or unweighted (the former used to give nearby terms greater importance than more distant ones). We employed a symmetric, weighted window of size six.
```



programminghistorian / jekyll

Issue 2911 #2912

Merged anisa-hawes merged 2 commits into gh-pages from Issue-2911 2 weeks ago

Conversation 1 · Commits 2 · Checks 1 · Files changed 2

anisa-hawes commented 2 weeks ago · edited

Replacing a broken link in the following lessons:

/en/lessons/introduction-to-ffmpeg
/es/lecciones/introduccion-a-ffmpeg

This link https://vimeo.com/help/compression has broken. Replacing it with:
EN https://help.vimeo.com/hc/en-us/articles/12426043233169-Video-and-audio-compression-guidelines
ES https://help.vimeo.com/hc/es/articles/12426043233169-Video-and-audio-compression-guidelines

Closes #2911

Checklist

- [x] Assign yourself in the "Assignees" menu
- [x] Add the appropriate "Label"
- [x] If this PR closes an issue, add the phrase `Closes #ISSUENUMBER` to your summary above
- [x] Ensure the status checks pass: if you have difficulty fixing build errors, please contact our Publishing Assistant @anisa-hawes
- [x] Check the Netlify Preview: navigate to netlify/ph-preview/deploy-preview and click 'details' (at right)
- [x] Assign at least one individual or team to "Reviewers"
  - If the text needs to be translated, please follow the translation request guidelines, then assign the relevant ~~language team(s) as "Reviewers" and tag both the team as well as the managing editor in your PR.~~

anisa-hawes added 2 commits 2 weeks ago

```
update introduction-to-ffmpeg.md    Verified    5ead18f
update introduccion-a-ffmpeg.md     Verified ✓  371248b
```

anisa-hawes added Lesson Maintenance spanish english labels 2 weeks ago

anisa-hawes self-assigned this 2 weeks ago

anisa-hawes commented 2 weeks ago

Hello @jenniferisasi. This is a quick link fix for an EN + ES lesson. Do you have a moment to review this for me?

Thank you! A.

anisa-hawes requested a review from **jenniferisasi** 2 weeks ago

jenniferisasi approved these changes 2 weeks ago

Reviewers: jenniferisasi ✓
Assignees: anisa-hawes
Labels: english, Lesson Maintenance, spanish
Projects: None yet
Milestone: No milestone
Development: Successfully merging this pull request may close these issues.
Replace broken link in /en/lessons/introduct...
Notifications: You're receiving notifications because you're watching this repository.
2 participants

Cloning and Forking

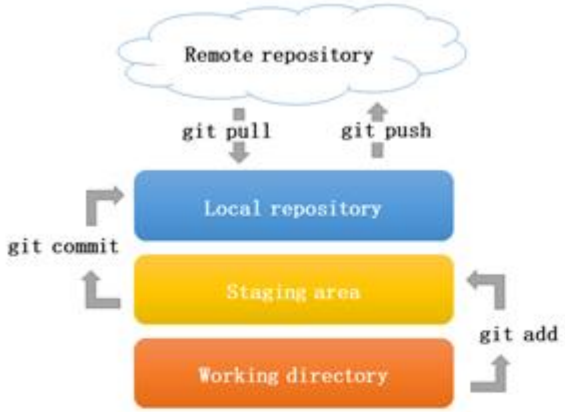
GitHub was built for sharing and collaborating on projects. A key advantage of the platform is that you can find lots of bits of software that do many different things—such as code for plugins for WordPress or Leaflet. If a project is public, you can save a copy of it to your local machine, work on it, save your amendations and share it on your own GitHub account. Cloning and forking are the basic functions of this capability.

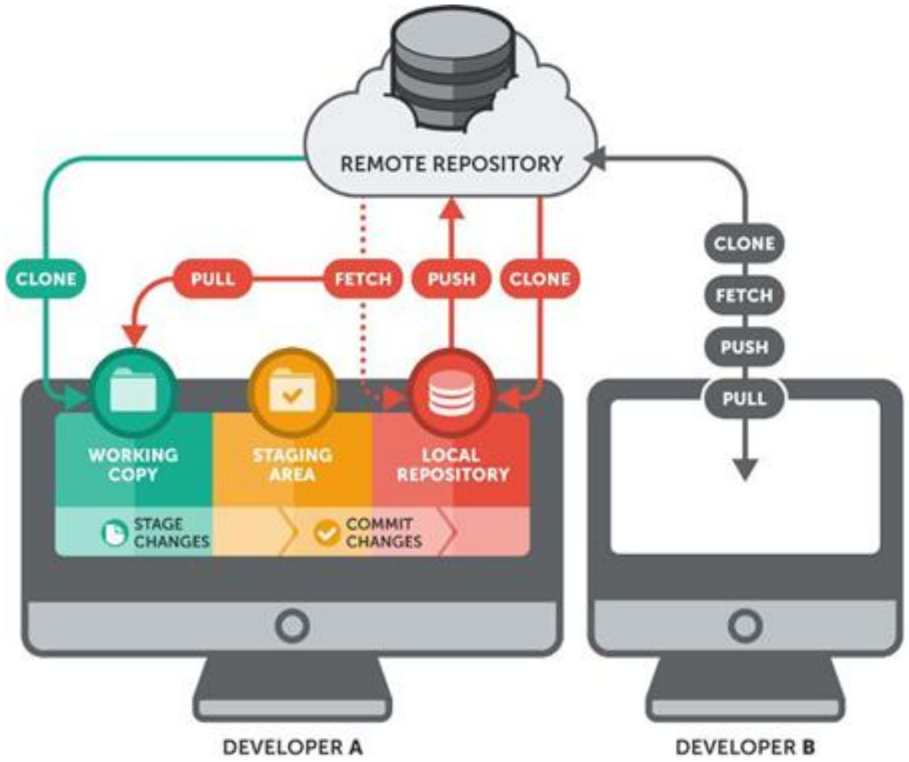
Cloning a repository means making a copy of a repository on GitHub, to download and work on locally—on your local machine. By entering the following code into your terminal, you can clone any public directory on GitHub:

```
$ git clone <repository-url>
```

When you clone a repository from GitHub, the folder that shows up on your local machine comes built-in with a few things. First, Git is already present, so you don't need to initialize the folder. Also, the connection between your local copy and the online repository is already made, so git push origin main will work (no `-u` flag needed).

Forking a repository means making a copy of someone else's repository on GitHub, and saving it to your account on GitHub. This function happens within GitHub, and has nothing to do with what is happening on your local machine. Note that *forking* will not automatically make the repository appear as a folder on your computer; that's the role of *cloning*.


Remote repository
git pull
git push
Local repository
git commit
Staging area
git add
Working directory


REMOTE REPOSITORY
CLONE
PULL
FETCH
PUSH
CLONE
CLONE
FETCH
PUSH
PULL
WORKING COPY
STAGING AREA
LOCAL REPOSITORY
STAGE CHANGES
COMMIT CHANGES
DEVELOPER A
DEVELOPER B

Issues

Use GitHub Issues to track ideas, feedback, tasks, or bugs for work on GitHub.
integrated with GitHub

Track work

You can organize and prioritize issues with projects. To track issues as part of a larger issue, you can use task lists. To categorize related issues, you can use labels and milestones.

Stay up to date

To stay updated on the most recent comments in an issue, you can subscribe to an issue to receive notifications about the latest comments. To quickly find links to recently updated issues you're subscribed to, visit your dashboard.

Community management

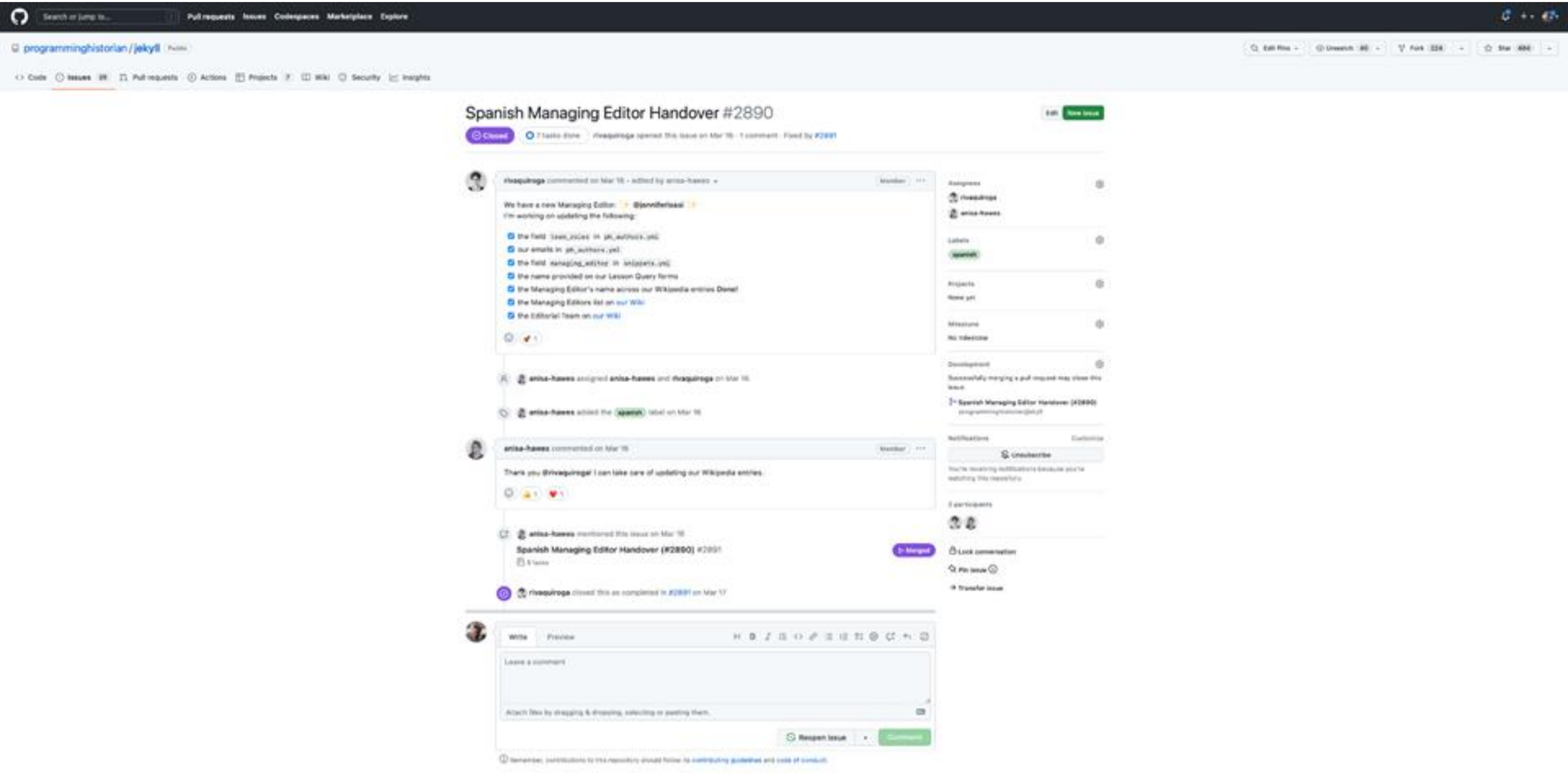
To help contributors open meaningful issues that provide the information that you need, you can use issue forms and issue templates.

Efficient communication

You can @mention collaborators who have access to your repository in an issue to draw their attention to a comment. To link related issues in the same repository, you can type # followed by part of the issue title and then clicking the issue that you want to link. To communicate responsibility, you can assign issues.

Comparing issues and discussions

Some conversations are more suitable for GitHub Discussions. You can use GitHub Discussions to ask and answer questions, share information, make announcements, and conduct or participate in conversations about a project.



programminghistorian / jekyll Public

Spanish Managing Editor Handover #2890

Closed · 7 tasks done · rivaquiroga opened this issue on Mar 16 · 1 comment · Fixed by #2891

rivaquiroga commented on Mar 16 · edited by anisa-hawes · Member

We have a new Managing Editor ✨ @jenniferisasi ✨
I'm working on updating the following:

- [x] the field `team_roles` in `ph_authors.yml`
- [x] our emails in `ph_authors.yml`
- [x] the field `managing_editor` in `snippets.yml`
- [x] the name provided on our Lesson Query forms
- [x] the Managing Editor's name across our Wikipedia entries **Done!**
- [x] the Managing Editors list on our Wiki
- [x] the Editorial Team on our Wiki

anisa-hawes assigned **anisa-hawes** and **rivaquiroga** on Mar 16

anisa-hawes added the spanish label on Mar 16

anisa-hawes commented on Mar 16 · Member

Thank you **@rivaquiroga!** I can take care of updating our Wikipedia entries.

anisa-hawes mentioned this issue on Mar 16

Spanish Managing Editor Handover (#2890) #2891 · 5 tasks · Merged

rivaquiroga closed this as completed in #2891 on Mar 17

Assignees: rivaquiroga, anisa-hawes
Labels: spanish
Projects: None yet
Milestone: No milestone
Development: Successfully merging a pull request may close this issue. Spanish Managing Editor Handover (#2890) programminghistorian/jekyll
Notifications: Unsubscribe. You're receiving notifications because you're watching this repository.
2 participants

Projects

A project is an adaptable spreadsheet that integrates with your issues and pull requests on GitHub to help you plan and track your work effectively. You can create and customize multiple views by filtering, sorting, grouping your issues and pull requests, visualize work with configurable charts, and add custom fields to track metadata specific to your team. Rather than enforcing a specific methodology, a project provides flexible features you can customize to your team's needs and processes.

Your projects are built from the issues and pull requests you add, creating direct references between your project and your work. Information is synced automatically to your project as you make changes, updating your views and charts. This integration works both ways, so that when you change information about a pull request or issue in your project, the pull request or issue reflects that information. For example, change an assignee in your project and that change is shown in your issue. You can take this integration even further, group your project by assignee, and make changes to issue assignment by dragging issues into the different groups.

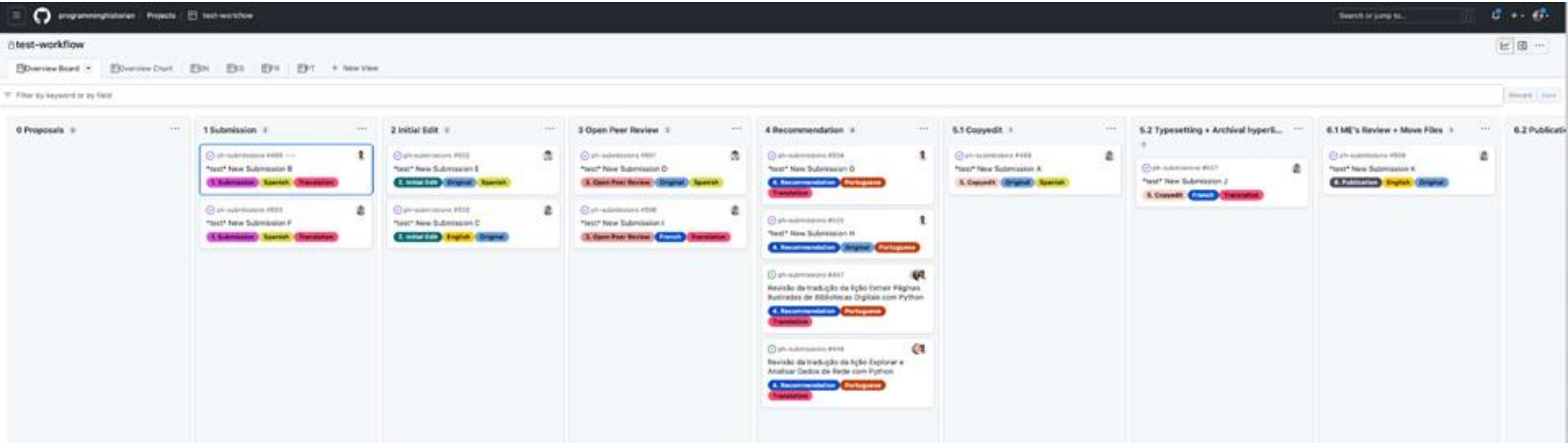
Adding metadata to your items

You can use custom fields to add metadata to your issues, pull requests, and draft issues and build a richer view of item attributes. You're not limited to the built-in metadata (assignee, milestone, labels, etc.) that currently exists for issues and pull requests.

Exploring the relationships between issues

You can use tasklists to build hierarchies of issues, dividing your issues into smaller subtasks, and creating new relationships between your issues. These relationships are displayed on the issue, as well as the Tracked by and Tracks fields in your projects. You can filter by issues which are tracked by another issue, and you can also group your table views by the Tracked by field to show all parent issues with a list of their subtasks.

Viewing your project from different perspectives


programminghistorian / Projects / test-workflow

test-workflow

Overview Board · Overview Chart · EN · ES · FR · PT · + New View

Filter by keyword or by field

0 Proposals 0

1 Submission 2

- ph-submissions #499 — *test* New Submission B — 1. Submission · Spanish · Translation
- ph-submissions #503 — *test* New Submission F — 1. Submission · Spanish · Translation

2 Initial Edit 2

- ph-submissions #502 — *test* New Submission E — 2. Initial Edit · Original · Spanish
- ph-submissions #500 — *test* New Submission C — 2. Initial Edit · English · Original

3 Open Peer Review 2

- ph-submissions #501 — *test* New Submission D — 3. Open Peer Review · Original · Spanish
- ph-submissions #506 — *test* New Submission I — 3. Open Peer Review · French · Translation

4 Recommendation 4

- ph-submissions #504 — *test* New Submission G — 4. Recommendation · Portuguese · Translation
- ph-submissions #505 — *test* New Submission H — 4. Recommendation · Original · Portuguese
- ph-submissions #447 — Revisão da tradução da lição Extrair Páginas Ilustradas de Bibliotecas Digitais com Python — 4. Recommendation · Portuguese · Translation
- ph-submissions #448 — Revisão da tradução da lição Explorar e Analisar Dados de Rede com Python — 4. Recommendation · Portuguese · Translation

5.1 Copyedit 1

- ph-submissions #498 — *test* New Submission A — 5. Copyedit · Original · Spanish

5.2 Typesetting + Archival hyperli... 1

- ph-submissions #507 — *test* New Submission J — 5. Copyedit · French · Translation

6.1 ME's Review + Move Files 1

- ph-submissions #508 — *test* New Submission K — 6. Publication · English · Original

6.2 Publicati

Labels and milestones

You can manage your work on GitHub by creating labels to categorize issues, pull requests, and discussions. You can apply labels in the repository the label was created in. Once a label exists, you can use the label on any issue, pull request, or discussion within that repository.

About default labels

GitHub provides default labels in every new repository. You can use these default labels to help create a standard workflow in a repository.

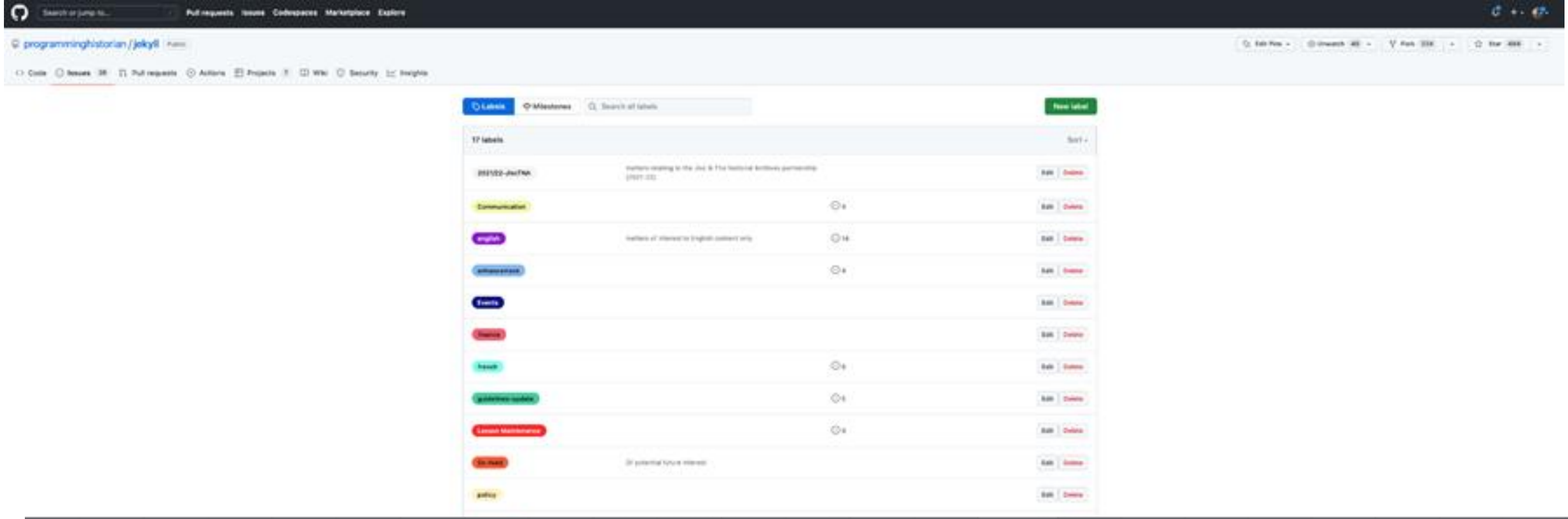
Creating a label

Anyone with write access to a repository can create a label.

Milestones

You can use milestones to track progress on groups of issues or pull requests in a repository.

When you create a milestone, you can associate it with issues and pull requests.


programminghistorian / jekyll Public

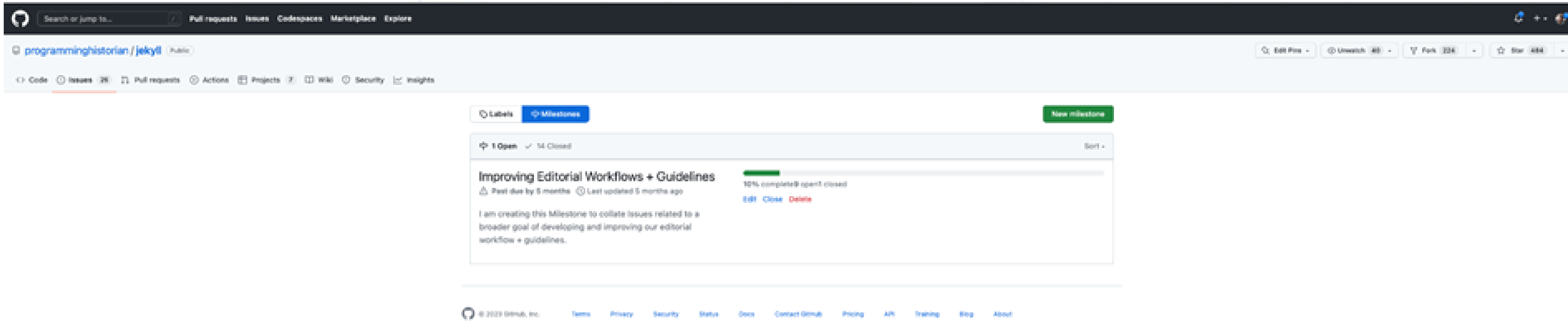
Pull requests Issues Codespaces Marketplace Explore

Code Issues Pull requests Actions Projects Wiki Security Insights

Labels Milestones Search all labels New label

17 labels Sort

2021/22-JiscTNA	Matters relating to the Jisc & The National Archives partnership (2021-22)		Edit Delete
Communication		6	Edit Delete
english	Matters of interest to English content only	14	Edit Delete
enhancement		6	Edit Delete
Events			Edit Delete
finance			Edit Delete
french		6	Edit Delete
guidelines update		5	Edit Delete
Lesson Maintenance		6	Edit Delete
On Hold	Of potential future interest		Edit Delete
policy			Edit Delete


programminghistorian / jekyll Public

Pull requests Issues Codespaces Marketplace Explore

Code Issues Pull requests Actions Projects Wiki Security Insights

Labels Milestones New milestone

1 Open ✓ 14 Closed Sort

Improving Editorial Workflows + Guidelines

Past due by 5 months Last updated 5 months ago

I am creating this Milestone to collate issues related to a broader goal of developing and improving our editorial workflow + guidelines.

10% complete 9 open 1 closed

Edit Close Delete

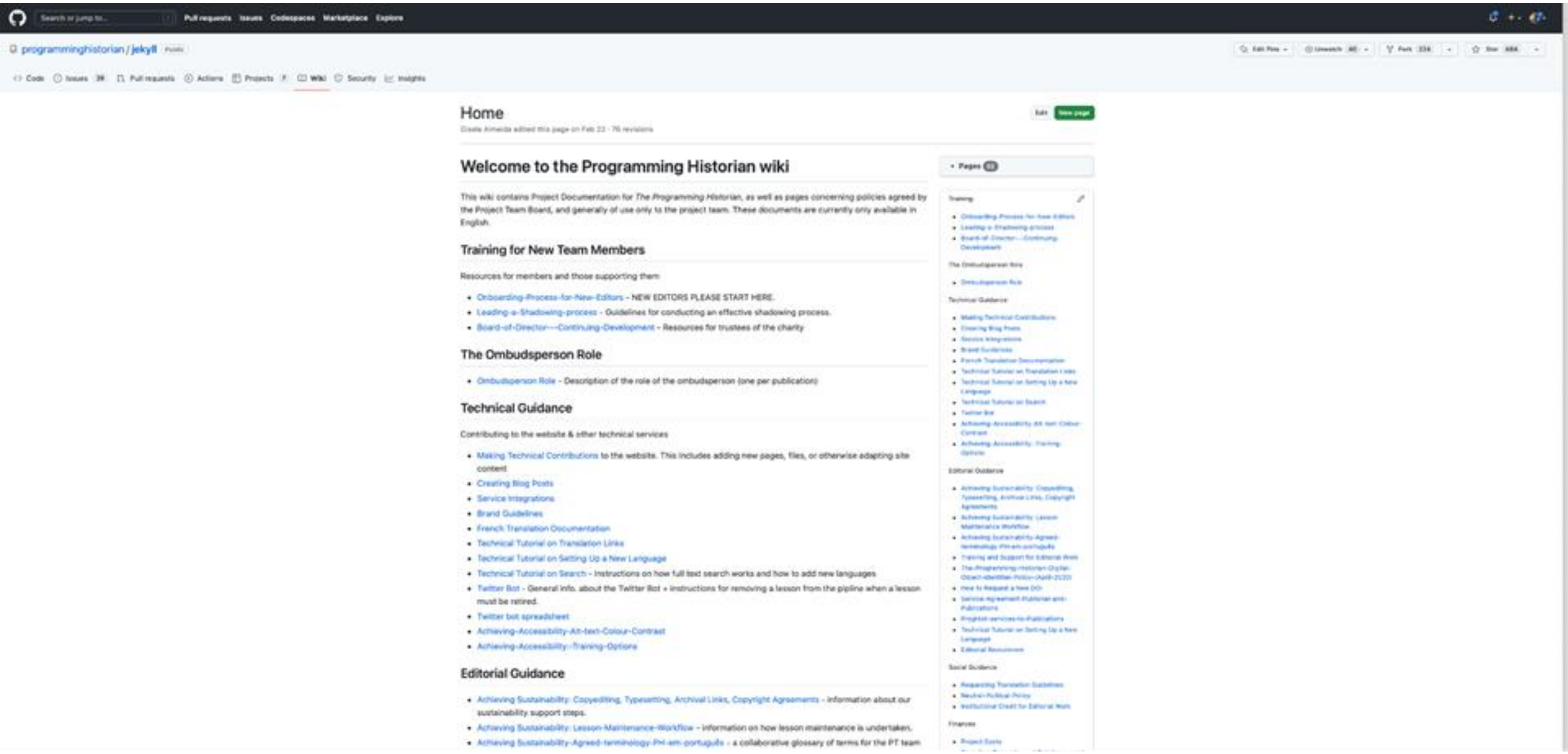
© 2023 GitHub, Inc. Terms Privacy Security Status Docs Contact GitHub Pricing API Training Blog About

About wikis

You can host documentation for your repository in a wiki, so that others can use and contribute to your project.

Every repository on GitHub.com comes equipped with a section for hosting documentation, called a wiki. You can use your repository's wiki to share long-form content about your project, such as how to use it, how you designed it, or its core principles. A README file quickly tells what your project can do, while you can use a wiki to provide additional documentation.

With wikis, you can write content just like everywhere else on GitHub. We use our open-source Markup library to convert different formats into HTML, so you can choose to write in Markdown or any other supported format. You can use Markdown to add rendered math expressions, diagrams, maps, and 3D models to your wiki.



Home

Gisele Almeida edited this page on Feb 22 · 76 revisions

Welcome to the Programming Historian wiki

This wiki contains Project Documentation for *The Programming Historian*, as well as pages concerning policies agreed by the Project Team Board, and generally of use only to the project team. These documents are currently only available in English.

Training for New Team Members

Resources for members and those supporting them

- Onboarding-Process-for-New-Editors - NEW EDITORS PLEASE START HERE.
- Leading-a-Shadowing-process - Guidelines for conducting an effective shadowing process.
- Board-of-Director---Continuing-Development - Resources for trustees of the charity

The Ombudsperson Role

- Ombudsperson Role - Description of the role of the ombudsperson (one per publication)

Technical Guidance

Contributing to the website & other technical services

- Making Technical Contributions to the website. This includes adding new pages, files, or otherwise adapting site content
- Creating Blog Posts
- Service Integrations
- Brand Guidelines
- French Translation Documentation
- Technical Tutorial on Translation Links
- Technical Tutorial on Setting Up a New Language
- Technical Tutorial on Search - Instructions on how full text search works and how to add new languages
- Twitter Bot - General info. about the Twitter Bot + instructions for removing a lesson from the pipline when a lesson must be retired.
- Twitter bot spreadsheet
- Achieving-Accessibility-Alt-text-Colour-Contrast
- Achieving-Accessibility:-Training-Options

Editorial Guidance

- Achieving Sustainability: Copyediting, Typesetting, Archival Links, Copyright Agreements - information about our sustainability support steps.
- Achieving Sustainability: Lesson-Maintenance-Workflow - information on how lesson maintenance is undertaken.
- Achieving Sustainability-Agreed-terminology-PH-em-português - a collaborative glossary of terms for the PT team

Training

- Onboarding-Process-for-New-Editors
- Leading-a-Shadowing-process
- Board-of-Director---Continuing-Development

The Ombudsperson Role

- Ombudsperson Role

Technical Guidance

- Making Technical Contributions
- Creating Blog Posts
- Service Integrations
- Brand Guidelines
- French Translation Documentation
- Technical Tutorial on Translation Links
- Technical Tutorial on Setting Up a New Language
- Technical Tutorial on Search
- Twitter Bot
- Achieving-Accessibility-Alt-text-Colour-Contrast
- Achieving-Accessibility:-Training-Options

Editorial Guidance

- Achieving Sustainability: Copyediting, Typesetting, Archival Links, Copyright Agreements
- Achieving Sustainability: Lesson-Maintenance-Workflow
- Achieving Sustainability-Agreed-terminology-PH-em-português
- Training and Support for Editorial Work
- The-Programming-Historian-Digital-Object-Identifier-Policy-(April-2020)
- How to Request a New DOI
- Service-Agreement-Publisher-and-Publications
- ProgHist-services-to-Publications
- Technical Tutorial on Setting Up a New Language
- Editorial Recruitment

Social Guidance

- Requesting Translation Guidelines
- Neutral-Political-Policy
- Institutional Credit for Editorial Work

Finances

- Project Costs

GitHub Pages

GitHub Pages is a simple service to publish a website directly on GitHub from a Git repository. You add some files and folders to a repository and GitHub Pages turns it into a website. You can use HTML directly if you like, but they also provide Jekyll, which renders Markdown into HTML and makes it really easy to setup a blog or a template-based website.

GitHub Pages also mean that you can collaborate on a website with a lot of people without everyone having to communicate endlessly back and forwards about what changes need to be made, or have been made already. You can create ‘issues’ (things that need discussing or fixing), list things to do in the future, and allow other people visiting your website to quickly suggest, and help implement changes through pull requests.

GitHub Pages is turned off by default for all new repositories, and can be turned on in the settings menu for any repository.

Source branch (required)

Pages needs to know the branch in your repository from which you want to serve your site. This can be any branch, including `main`.

Theme (optional)

GitHub Pages provides different themes to visually style and organize your site’s content. Choosing a theme is optional, and themes can be interchanged quickly.

If you have any question, please drop me a line

annamaria.sichani@sas.ac.uk

ps:Training material will be soon uploaded at CE GitHub repo