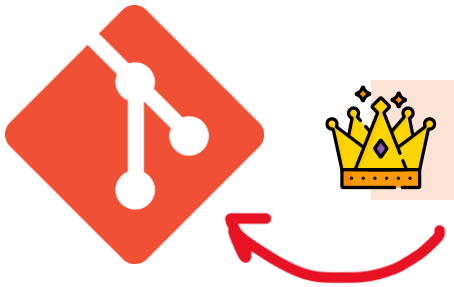
GitHub Foundations

Lecture Slides

Good luck on your exam!

Version Control Systems (VCS)

Version Control Systems (VCS) are **designed to track changes or revisions to code**.



The most popular VCS is **Git**.

1990	2000	2005	2005

Centralized VCS (DVCS)

Decentralized VCS (DVCS)

DVCS (and git specifically) became **very popular** for many reasons:

- Full local history and complete a copy of the repo locally
- Straightforward and efficient branching and merging
- Better performance, Improved fault tolerance, flexible workflows, work fully offline

Version Control normally represents revisions being represented as graph like structure. So you'll here terms like **tree**, **trunk** and **branches**

Git

Git is a **distributed version control systems (DVCS)** created by **Linus Torvald**.

creator of the linux kernel

Git Graph

Branches: Show All Show Rem

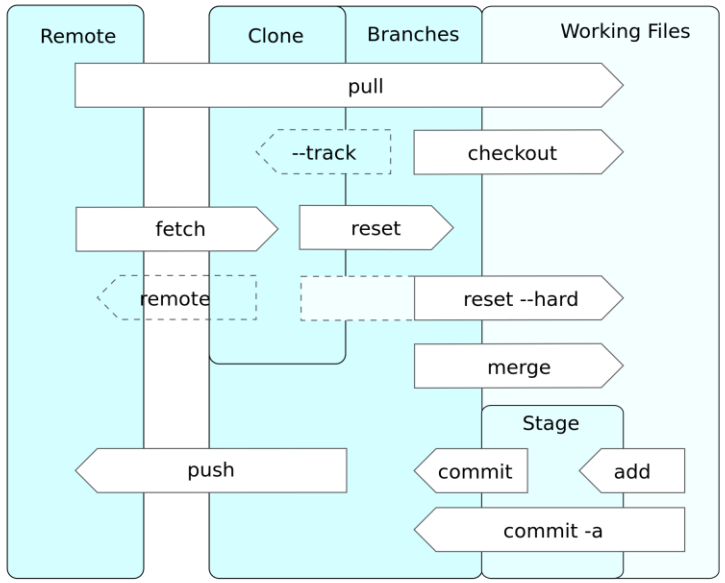
Gra...	Description	Date	Author	
	origin/65-final-category correct the town	11 Oct 2023 12:11	Andrew Brown	16796d58
	main origin origin/HEAD commit files (...	11 Oct 2023 11:48	Andrew Brown	abf341eb
	origin/63-payday-home commit files	7 Oct 2023 14:01	Andrew Brown	a8b839a9
	origin/terraform-cloud yes!	30 Sep 2023 13:59	Andrew Brown	1256b515
	please work!!!!	30 Sep 2023 13:54	Andrew Brown	e5baa1d5
	lets go	30 Sep 2023 13:49	Andrew Brown	33b8ea39
	comment out more	30 Sep 2023 13:42	Andrew Brown	a8e47468
	comment out code make things work	30 Sep 2023 13:40	Andrew Brown	c98c7107
	try and fix our issues.	30 Sep 2023 13:31	Andrew Brown	47b32b8f
	wip	30 Sep 2023 13:23	Andrew Brown	ad732e57
	2.6.0 #56 fix issues with provider, setup two diff...	29 Sep 2023 12:21	Andrew Brown	02fa4eb7
	origin/56-multi-home-terrafrom-cloud #56 fix issu...	29 Sep 2023 12:21	Andrew Brown	8e026527
	2.5.0 #52 test our terra home on terratowns test ...	29 Sep 2023 10:07	Andrew Brown	89dfc0e6
	origin/52-terratown-test #52 test our terra home ...	29 Sep 2023 10:06	Andrew Brown	895e72c3
	2.4.0 47 terratowns provider (#51)	28 Sep 2023 15:31	Andrew Brown	f0ef63f1
	origin/47-terratowns-provider Merge branch 'main...	28 Sep 2023 15:31	Andrew Brown	d23bf312
	$47 finish implementation for terraform provider	28 Sep 2023 15:29	Andrew Brown	004a68fc
	wip	28 Sep 2023 12:56	Andrew Brown	46f53c2e
	2.3.0 47 terratowns provider (#50)	28 Sep 2023 11:52	Andrew Brown	5f9189ee
	#47 setup skeleton for resource	28 Sep 2023 11:51	Andrew Brown	103d77d1
	Merge branch 'main' into 47-terratowns-provider	27 Sep 2023 16:35	Andrew Brown	257ca11b
	2.2.0 47 terratowns provider (#49)	27 Sep 2023 16:33	Andrew Brown	976bd7cb
	Merge branch 'main' into 47-terratowns-provider	27 Sep 2023 16:33	Andrew Brown	ad0c0347
	#47 implement terraform provider block in terraform	27 Sep 2023 16:31	Andrew Brown	72c519cb

2 public/index.html

```
@@ -6,6 +6,6 @@
     <title>Hello Terraformers</title>
 </head>
 <body>
-    <h1>Hello Terraformers!!!!!</h1>
+    <h1>Hello there Terraformers!<h1>
 </body>
 </html>
```

Each change of your code **(git commit)** can be captured and tracked throughout the history of your project **(git tree)**

Common Git Terms



- **Repository:** Represents the logical container holding the codebase
- **Commit:** Represents a change of data in the local repository.
- **Tree:** Represent the entire history of a repo
- **Remote:** A version of your project hosted elsewhere, used for exchanging commits.
- **Branches:** Divergent paths of development, allowing isolated changes.
  - **Main (previously known as master)** the most common name for the default branch
- **Clone:** Creates a complete local copy of a repository, including its history.
- **Checkout:** Switches between different branches or commits in your repo.
- **Pull:** Downloads changes from a remote repository and merges them into your branch.
- **Push:** Uploads your local repository changes to a remote repository.
- **Fetch:** Downloads data from a remote repo without integrating it into your work.
- **Reset**: Undoes local changes, with options to unstage or revert commits.
- **Merge:** Combines multiple commit histories into one.
- **Staging files:** Prepares and organizes changes for a commit.
  - **Commit:** Saves your changes as a snapshot in the local repository.
  - **Add:** Adds changes to the staging area for the next commit.

Version Control Services (VCS)

Version Control Services (VCS) are **fully managed cloud services that hosts your version controlled repositories**.
These services often have additional functionality going beyond just being a remote host for your repos.
Git is the most popular and often the only choice for VCS. Often, we call these services **"git providers"**

GitHub
Owned by Microsoft. The most popular VCS offering due to ease of use and being around the longest. GitHub is primarily where open-source projects are hosted and offer rich functionality such as Issues Tracking, Automation pipelines and a host of other features.

GitLab
GitLab was an emerging competitor to GitHub and at the time had unique features such as a CI/CD pipeline and improve security measures.

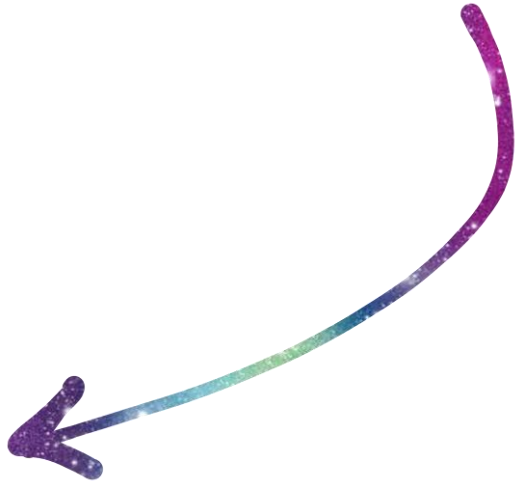
BitBucket
Owned by Atlassian. Originally hosted Mercurial, later adding Git and then sunsetting Mercurial.
Tight integration with Atlassian's industry standard agile project management software **JIRA.**
Larger organizations commonly use BitBucket because of JIRA adoption.

SourceForge
One of the oldest places to host your source code and was the first to free of charge to open-source projects

GitHub

GitHub is a **Version Control Service** that initially offered hosting managed remote git repositories and has **expanded to provide other offerings** around hosted codebases.

- Git Repository Hosting
- Project Management Tools
- Issue Tracking
- Pull Requests and Code Review
- GitHub Pages and Wikis
- GitHub Actions
- GitHub Copilot
- GitHub Codespaces
- GitHub Marketplace
- GitHub Gists
- GitHub Discussions
- Collaboration Features (Organizations and Teams)
- API Access and Development Tools (Github Desktop, Github CLI)
- Security Features (eg. Autodetcting credentials in repos)
- Educational Resources and Course Automation (Github Classroom)



Fun Fact! GitHub was originally built using **Ruby on Rails**

GitHub Octocat

GitHub's Octocat is the platform's **official mascot**. The character, a hybrid of an octopus and a cat, symbolizes GitHub's friendly and engaging nature in the world of software development and collaboration.

You can **create your own** Octocat

https://myoctocat.com

Git vs GitHub

	Git	GitHub
Nature	Distributed Version control system (DVCS)	Version Control as a Service (VCaaS)
Functionality	Manages source code history	Provides cloud storage for Git repositories
Access	Local system installation	Accessed via web interface
Scope	Local repository management	Online collaboration and repository hosting
Collaboration	Local changes, requires manual sharing	Integrated tools for collaboration (issues, PRs)
Usage	Command-line interface	Graphical interface and additional features

GitHub Repo

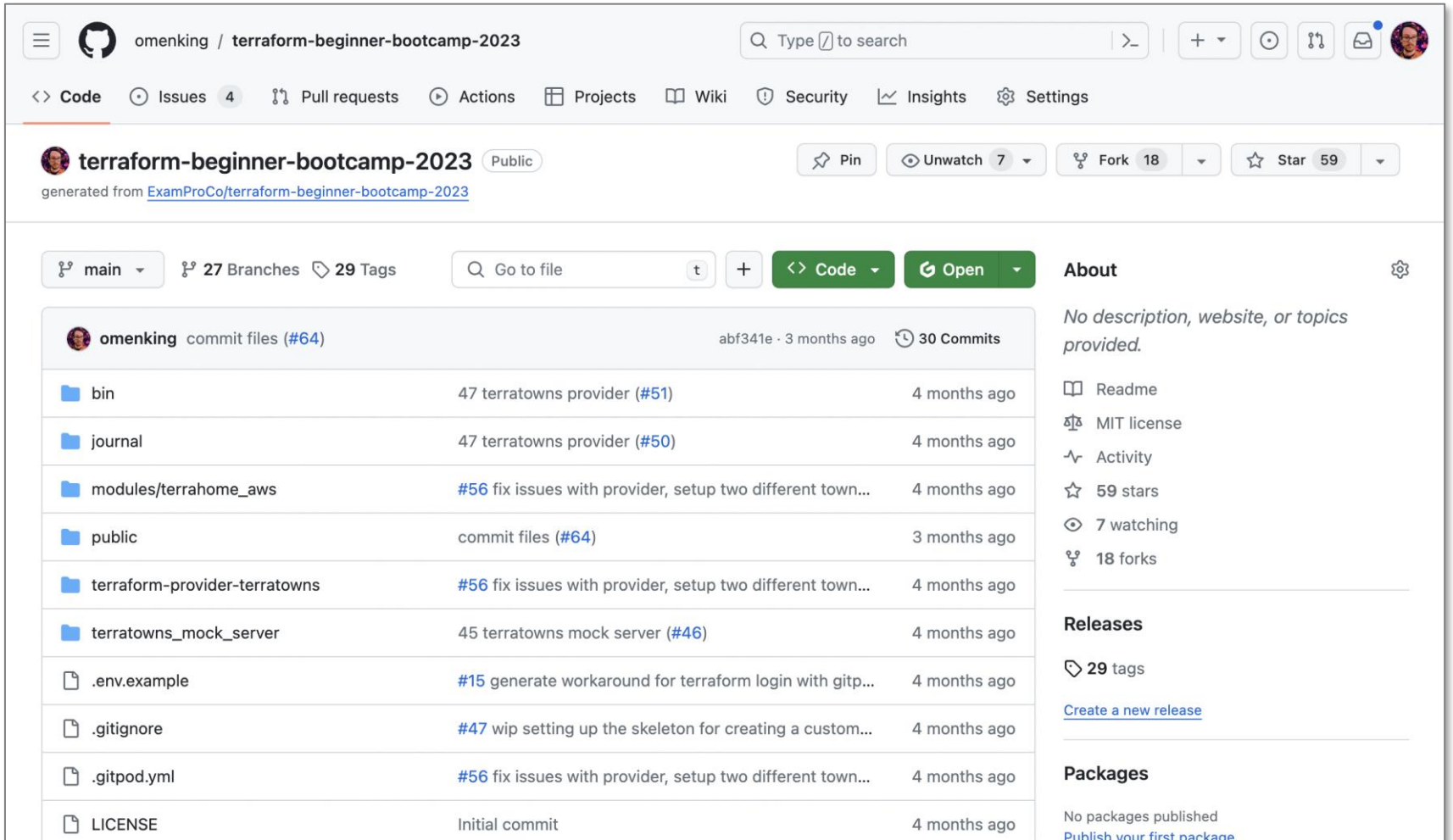
A **GitHub Repo** is **your git repo that you push upstream to GitHub**
GitHub allows you access and manage your git repo with several functionality

From your GitHub repo page you can:

- View different branches
- View tags
- View commit history
- Explore repo's files
- View releases
- See codebase language breakdown
- View the top-level markdown files
  - Readme.md
  - Licence.md

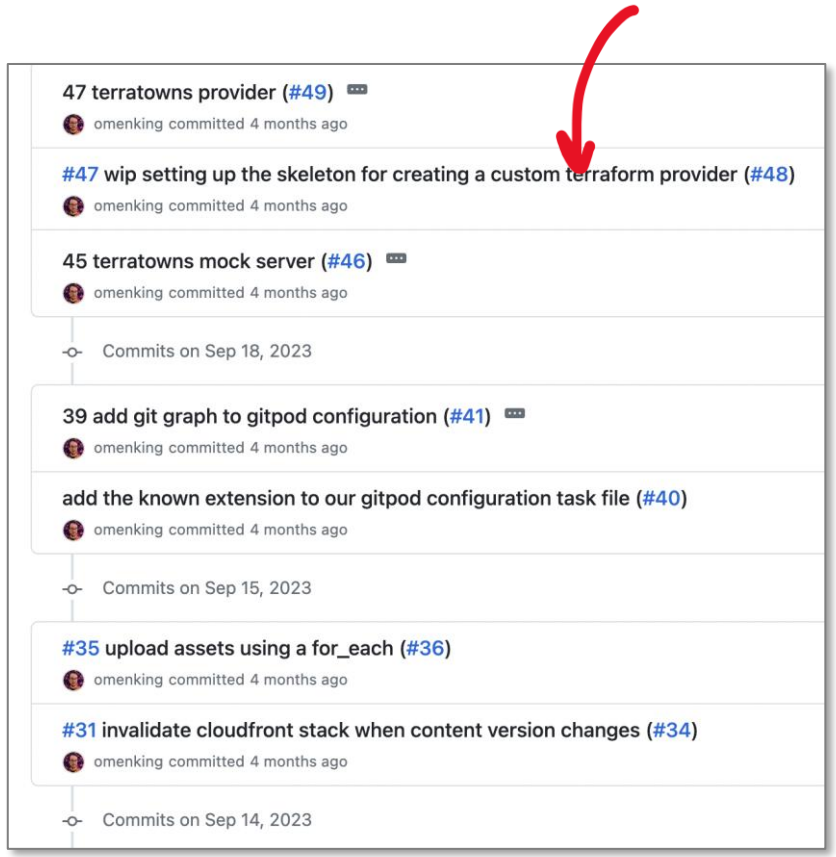
You can perform actions such as:

- Pinning
- Watching
- Forking
- Starring
- Cloning (downloading)



Git Commit

A Git Commit represent incremental changes to a codebase represented with a git tree (graph) at a specific time.



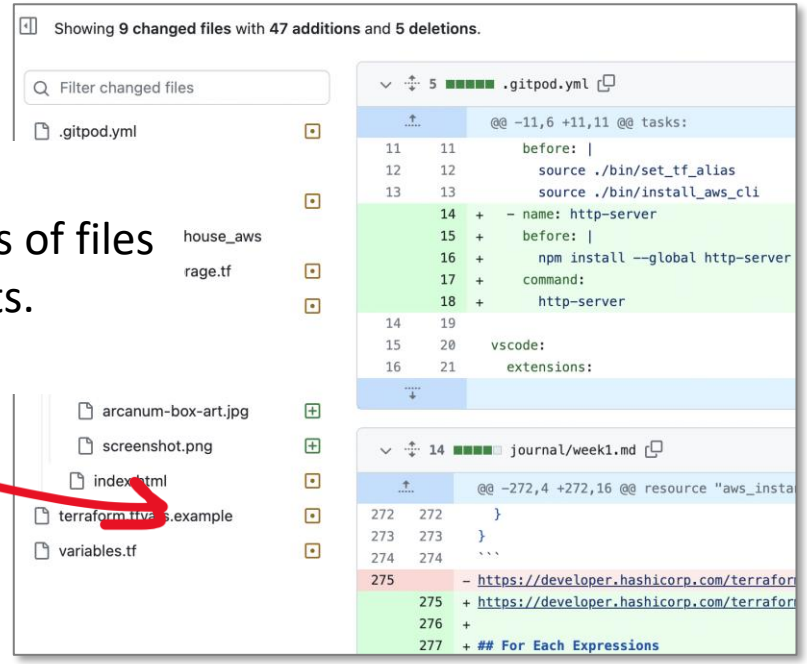
Each Commit has a SHA hash that acts as an ID

- Eg. 661a91ad3b66926c4591f9d3c73c087906945f3b

You can use this **Commit SHA** to *checkout* specific commits.

A git commit contains:

- Additions, **modifications** and deletions of files
- Additions and deletions of file contents.
- Not the whole files themselves.



Git does not store the whole files in each commit but rather the state of changes. This greatly reduces the file size. To the developer the files will appear whole

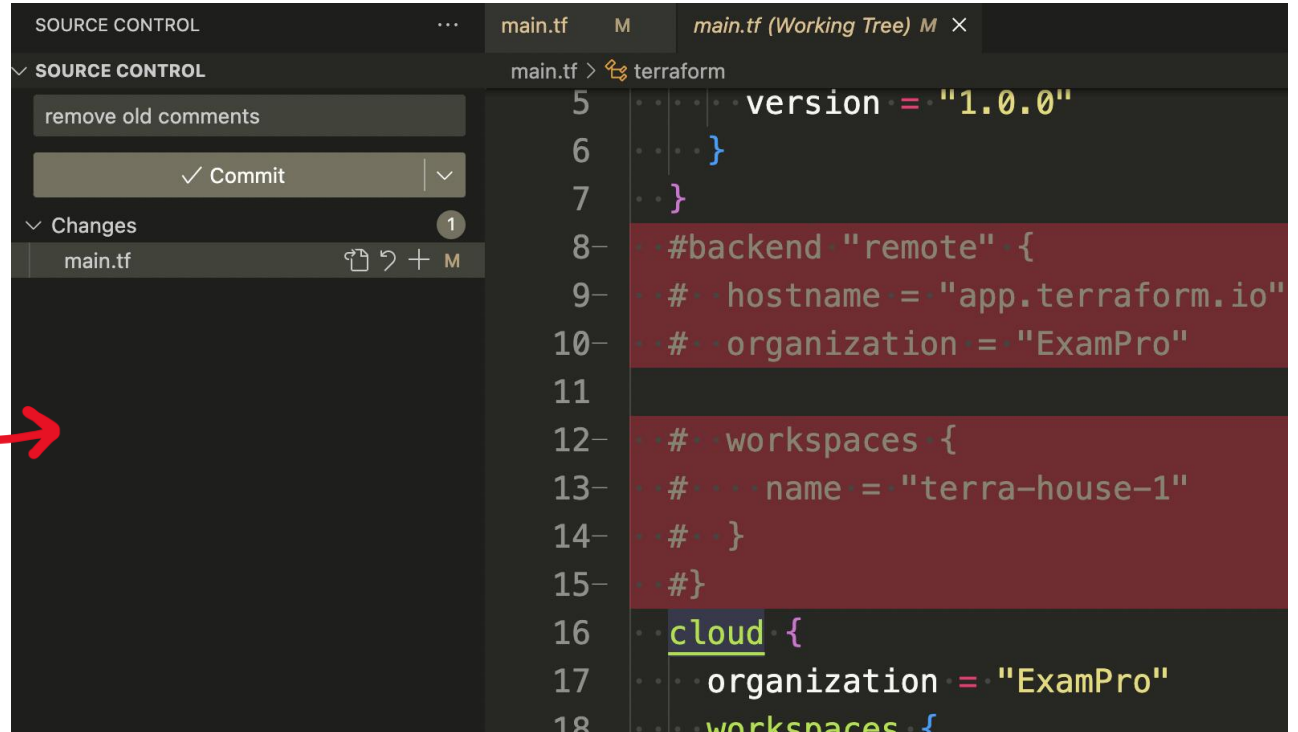
Git Commit

Components of Git Commit:

- **Commit Hash**: A unique SHA-1 hash identifier for the commit.
- **Author Information**: The name and email of the person who made the commit.
- **Commit Message**: A description of what changes the commit contains.
- **Timestamp**: The date and time when the commit was made.
- **Parent Commit Hash(es)**: The SHA-1 hash of the commit(s) this commit is based on.
- **Snapshot of Content**: A snapshot of the project at the time of the commit (not the actual files, but references to them).

Commit messages are often written in a tool as its more convenient to quickly add, remote files, audit changes

VS Code has that ability via the **Source Control** window



Git Commit

Common **Git Commands** you should know:

You need add or remove files to be stage changes

```
# Files to be staged
git add my-file
git add . # add all files
git rm my-other file # remove a specific file

# Commits staged changes with a message
git commit -m "Commit message"

# Automatically stages all tracked, modified files before the commit
git commit -a -m "Commit message"

#Modifies the most recent commit
git commit --amend

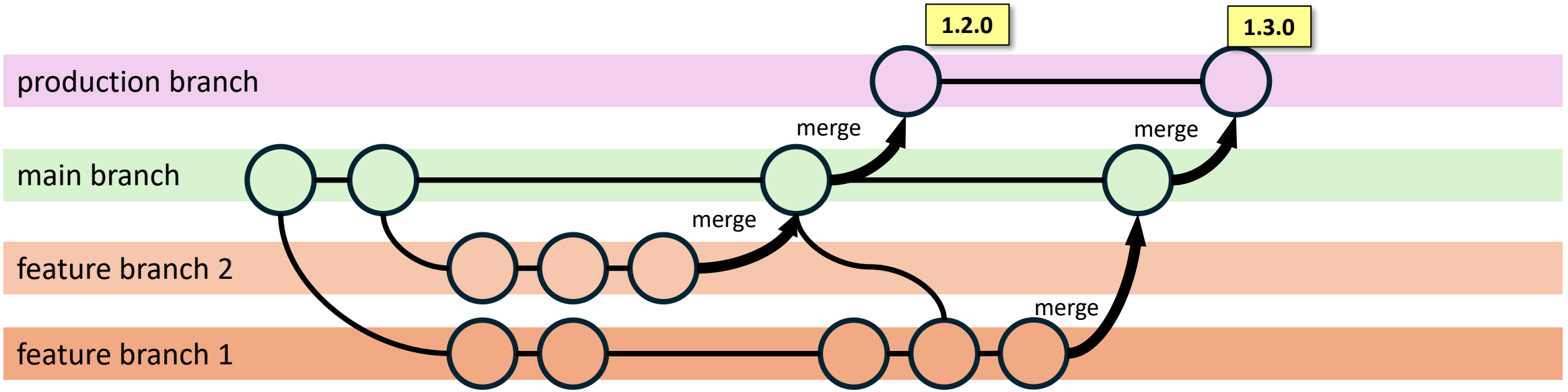
#Creates an empty commit, useful as a placeholder
git commit -m "Initial commit" --allow-empty

# Commits with a specified author.
git commit -m "Message" --author="Author Name <email@example.com>"

# checkout to a specific commit based on SHA hash
git checkout 661a91ad3b66926c4591f9d3c73c087906945f3b
```

Git Branch

A **git branch** is **a divergence of the state of the repo**.
You can *think* of branches as being copies of a point in time that have been modified to be different.



You can have branches for:

- Specific environments: eg. Staging, development, production
- Specific to developers: eg. andrew, bayko, cindy
- branches per feature: eg. payment-gateway-feature
- Branches per bug eg. hotfix-blank-homepage

This type of workflow is is very close to **GitHub Flow**

Git Branch

Common **git branch commands** you should know

```
#Lists all local branches
git branch

#Creates a new branch
git branch [branch-name]

# Checkout (switch to) a branch
git checkout [branch-name]

# Create and checkout a branch
git checkout -b [branch name]

#Deletes a branch
git branch -d [branch-name]:

# Renames a branch
git branch -m [old-name] [new-name]

# Lists both remote and local branches
git branch -a
```

A **common workflow** for developers is to create a branch or a feature, and they need to push their branch upstream to the remote name origin.

```
git checkout -b my-new-branch
# ... makes changes to files
git add .
git commit -m "my changes"
git push -u origin my-new-branch
```

Git Remote

A git **remote** represents **the reference to remote location** where a copy of the repository is hosted.

You can have multiple remote entries for your git repo.

- "origin" as a remote name almost always seen for a repo.
  - It indicates the central or golden repo everyone is working from and represents the source of truth.

The remote entries are stored in **.git/config**

```
[remote "origin"]
  url = https://github.com/examproco/exampro.git
  fetch = +refs/heads/*:refs/remotes/origin/*
[branch "production"]
  remote = origin
  merge = refs/heads/production
[branch "main"]
  remote = origin
  merge = refs/heads/main
```

Notice remote names can be **referenced**

```
# Lists all remote repositories along with their URLs.
git remote -v

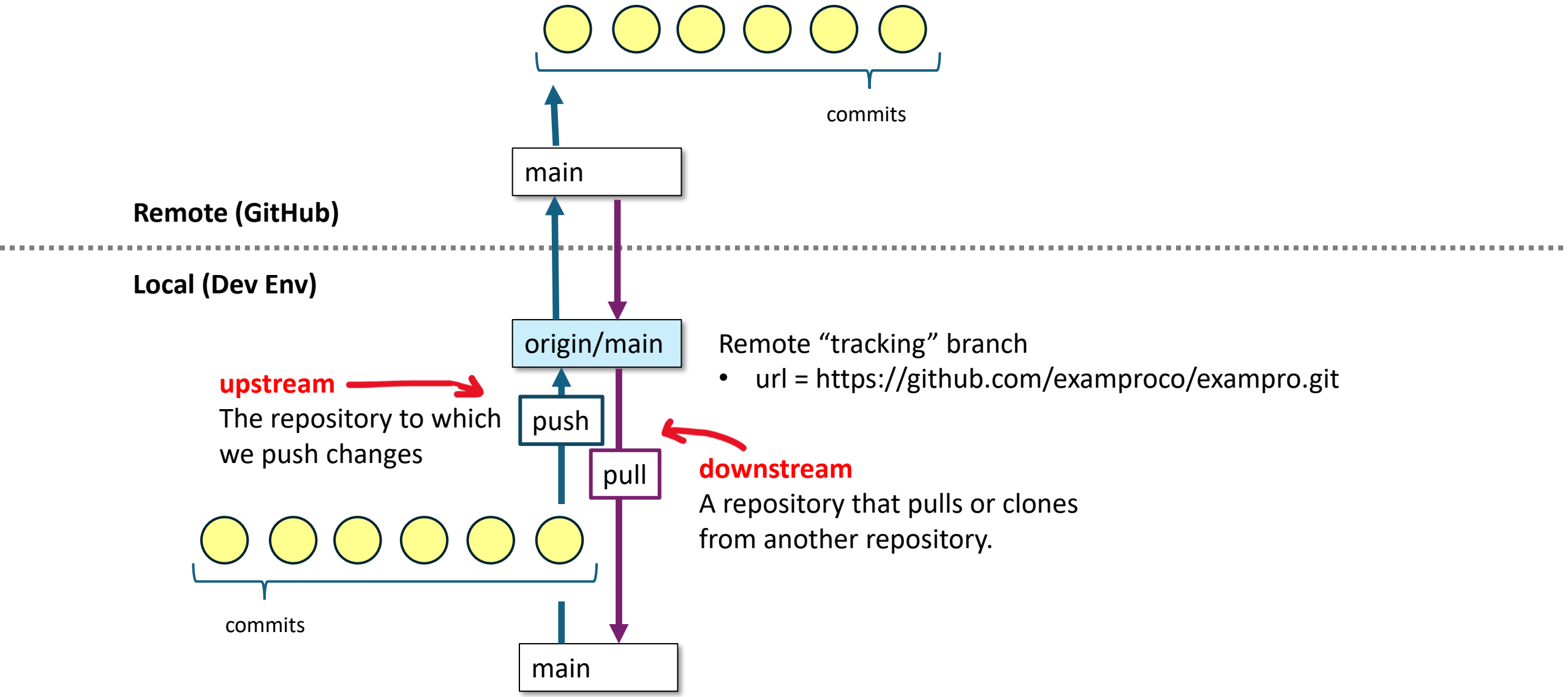
git remote add [name] [URL]
git remote remove [name]
git remote rename [old-name] [new-name]

#Pushes a branch and its commits to the specific remote.
git push [remote-name] [branch]

#Pulls updates from a remote branch.
git pull [remote-name] [branch]

# Fetch updates without pulling.
git fetch [remote-name]
```

Git Remote – Upstream and Downstream



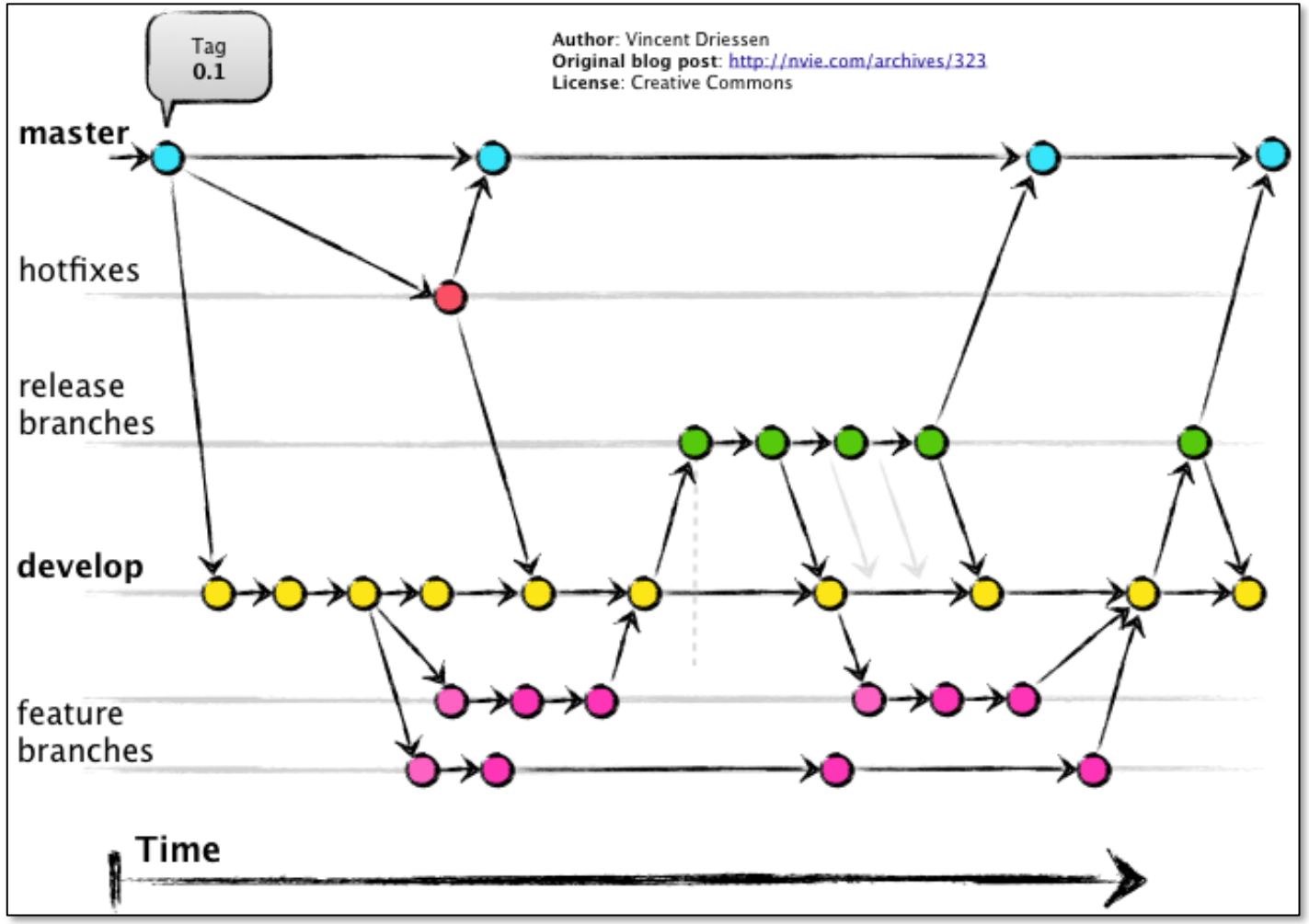
Remote “tracking” branch

- url = https://github.com/examproco/exampro.git

GitHub Flow

GitHub Flow is a **light-weight workflow** for multiple developers working on a single repository

Create a Branch: For each new task or feature, create a new branch off the main branch.
Add Commits: Make changes and commit them to your branch.
Open a Pull Request: Start a discussion about your commits, reviewing code in the pull request.
Discuss and Review: Share your pull request with teammates for feedback.
Deploy: Test your changes in a production environment.
Merge: Once your changes are verified, merge them into the main branch.



GitHub CLI

The **GitHub CLI** is a **command line interface to interact with your GitHub Account.**

You can quickly perform common GitHub actions without leaving you developer environment

```
gh auth login
gh repo create github-examples --public
gh issue create --title "Issue title" --body "Issue body"
gh pr review --comment -b "interesting"
```

The GitHub CLI can be installed on

Windows, Linux, and macOS

```
brew install gh
```

You can quickly install the CLI for GitHub Codespaces via Features:

```
"features": {
  "ghcr.io/devcontainers/features/github-cli:1": {}
}
```

Core commands

- gh auth
- gh browse
- gh codespace
- gh gist
- gh issue
- gh org
- gh pr
- gh project
- gh release
- gh repo

GitHub Actions commands

- gh cache
- gh run
- gh workflow

Additional commands

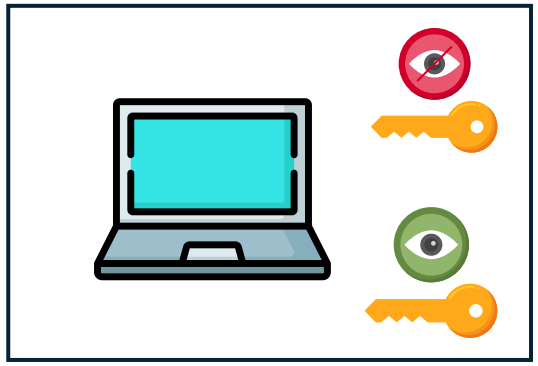
- gh alias
- gh api
- gh completion
- gh config
- gh extension
- gh gpg-key
- gh label
- gh ruleset
- gh search
- gh secret
- gh ssh-key
- gh status
- gh variable

SSH Keys

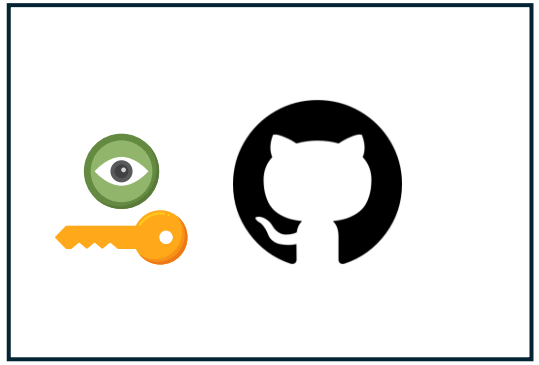
A common strategy for **authenticating to perform git operations** on you remote GitHub repo is using SSH keys

```
ssh-keygen -t rsa
```

SSH uses a public and private key to authorize remote access to servers.
ssh-keygen is a **well known command** to generate a public and private key



A copy of the public and private key reside on your local computer.



A copy of the public key is stored on GitHub.

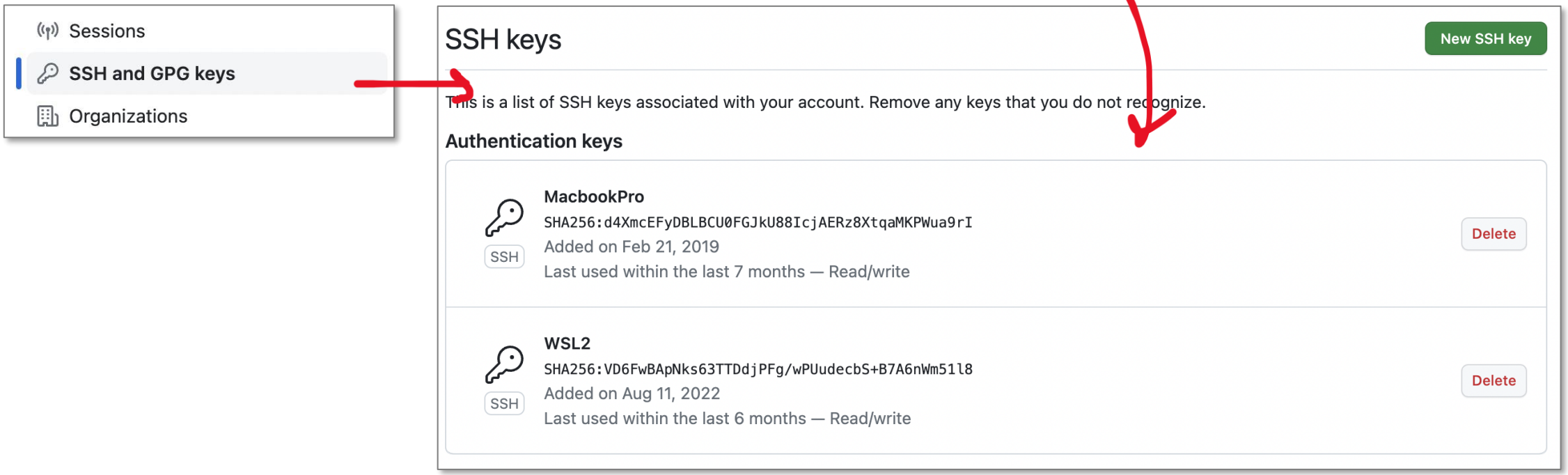
- the server checks if you have the same public key
- If you do a challenge message is sent
  - Its an encrypted message with your public key

- Only a private key can decrypt the message
- Once decrypted the client signs the message also using the private key

- The server using the public key will verify the signature
- A connection is established.

SSH Keys

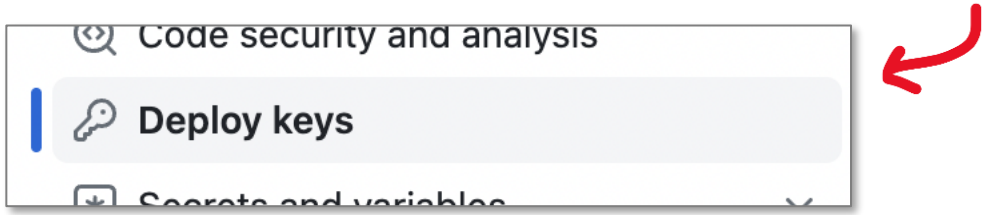
Under your Account Settings of **SSH and GPG keys** you can **add your Public Keys** to your GitHub account



Then you would clone your repo using the **ssh style address**.

Deploy Keys

Deploy Keys allows you **attach public keys directly to a Git Repo**



The use-case for deploy keys are for:

- **Build servers or CI/CD third-party** services that need to clone the repo so they perform a build or a deploy
- **Single Repo Access** so instead of using a shared key pair for multiple repos you have a single key pair for a single Git Repo
- Avoid Using **Personal Account Access Tokens**

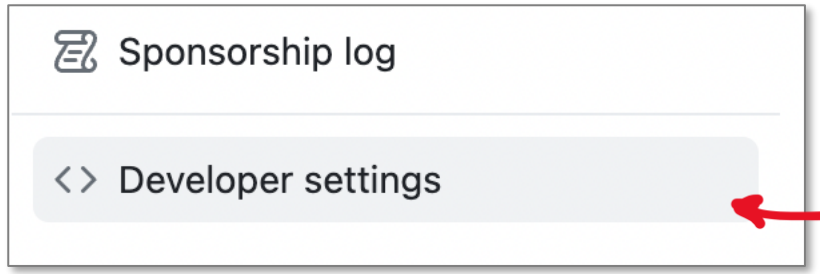
Personal Account Access Tokens

Personal Access Tokens (PAT) are an **alternative to using passwords** for authentication to GitHub when using the GitHub API or the command line

GitHub no longer lets you use passwords when interacting with the API

GitHub has two types of PATS:

- **Classic tokens**
  - They are less secure and no longer recommended for use
  - Customers with legacy systems may be still using classic token
- **Fine-grained personal access tokens**
  - granted specific permissions
  - Must have an expiry date
  - can only access specific repositories
  - can only access resources owned by a single user or organization



Personal Access Tokens is found under **Developer Settings**

Personal Account Access Tokens

When you **clone a git repo using HTTPS** you would use your personal access token to authorize the clone.

```
$ git clone https://github.com/USERNAME/REPO.git
Username: YOUR_USERNAME
Password: YOUR_PERSONAL_ACCESS_TOKEN
```

Instead of using gh login you could set the **GH_TOKEN**

```
export GH_TOKEN="MY_PRESONAL_ACCESS_TOKEN"
gh pr create --title "The bug is fixed" --body "Everything works again"
```

When using **GitHub SDKs,** you need to pass your PAT to authorize API calls

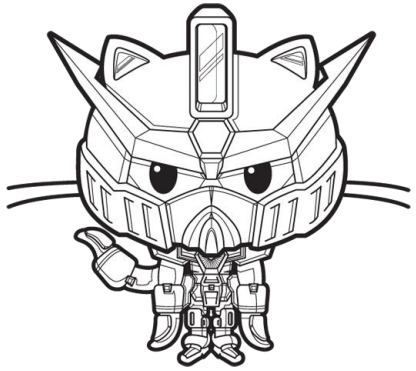
```
require 'octokit'
octokit = Octokit::Client.new(access_token: 'YOUR-TOKEN')
```

GitHub APIs

GitHub has two APIs **REST API** and **GraphQL API**

	REST API	GraphQL API
Design Philosophy	Resource-based. Different endpoints per resource	Single endpoint, query for precise data requests
HTTP Methods	Uses GET, POST, PUT, DELETE for operations	Mainly uses POST for all operations
Data Fetching	Multiple requests for related data	Single request for complex, related data
Over-fetching/Under-fetching	Can have over-fetching or under-fetching	Precise data fetching, no over/under-fetching
Performance	Less efficient for complex systems	More efficient for complex queries
Caching	Easier due to HTTP methods	More challenging due to POST method
Learning Curve	Generally easier to learn	Steeper but offers powerful features
Flexibility	Server-driven structure	Client-driven, highly flexible
Versioning	Often requires API versioning	Less need for versioning
Ecosystem	Mature, with extensive tools	Growing, with unique tools for queries

GitHub SDKs



Octokit are the official SDKs to programmatically interact with the Git REST API

GitHub official maintains SDKs for these languages

- JavaScript / TypeScript
- C# / .NET
- Ruby
- Terraform provider

Terraform managed a git repo as Infrastructure as Code

```
resource "github_repository" "example" {
  name        = "example"
  description = "My awesome codebase"

  visibility = "public"

  template {
    owner                = "github"
    repository           = "terraform-template-module"
    include_all_branches = true
  }
}
```

Creating a GitHub repo using **Ruby**

```
require 'octokit'

client = Octokit::Client.new(
  access_token: ENV['GH_TOKEN']
)

repo_name = "my_new_repo"
options = {
  description: "This is my new repository",
  private: false,
  has_issues: true,
  has_projects: true,
  has_wiki: true
}
client.create_repository(repo_name, options)
```

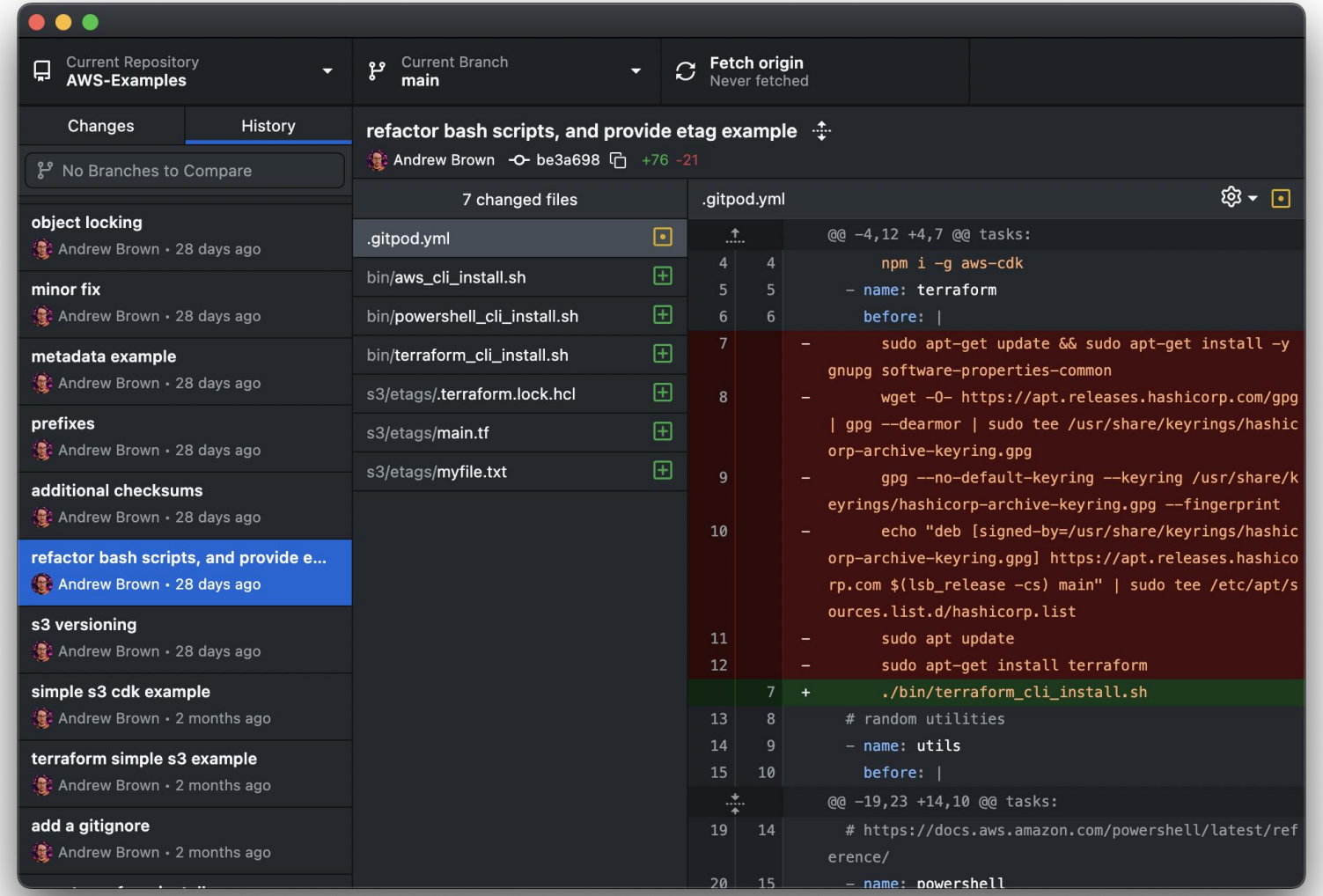
GitHub Desktop

GitHub Desktop is stand-alone application to interact with GitHub repos without the browser or via code.

Common Git and GitHub operations can be performed via the GUI for an easy-to-use experience.

GitHub desktops supports:

- MacOS
- Windows
- Linux



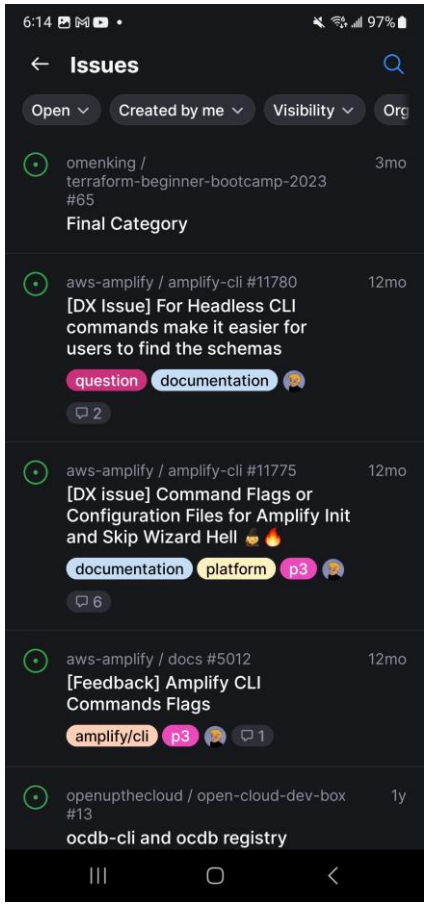
GitHub Desktop vs GitHub

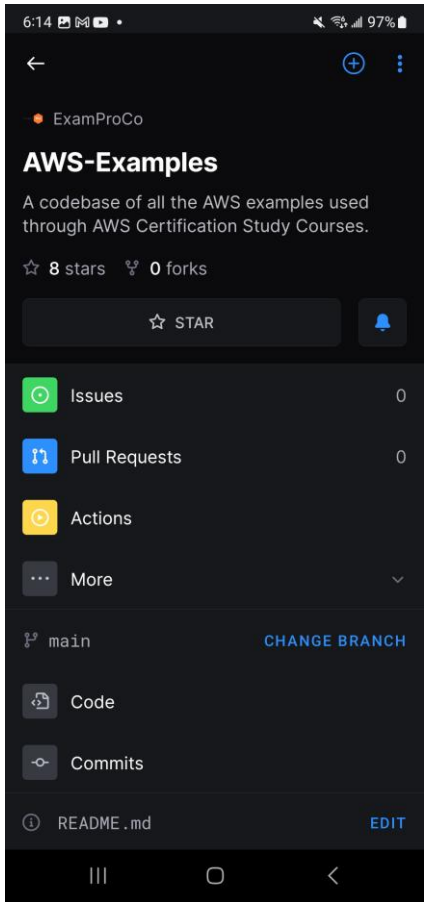
	Github.com	GitHub Desktop
Interface	Web-based interface accessed via browser	Standalone desktop application
Accessibility	Accessible anywhere with internet	Accessible only on installed computer
Repo Management	Directly manage repositories online	Manage local copies of repositories
Basic Git Ops.	Perform operations like fork, star, watch	Simplified interface for commit, pull, push
Collaboration	Direct collaboration tools like issues, PRs	Focuses on local repo management
Advanced Git Features	Supports advanced features via commands	Simplified experience for basic features
Integration with Tools	Integrates with various online CI/CD tools	Limited to local Git tools

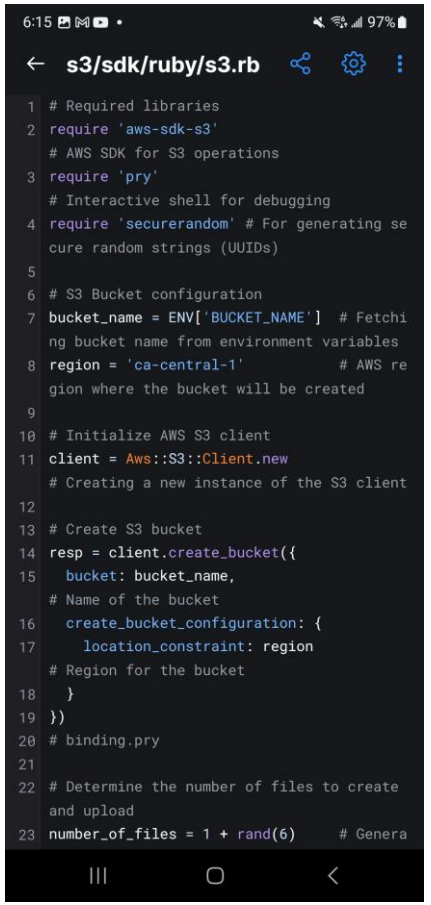
GitHub Mobile

GitHub Mobile is a mobile application you can install on iOS or Android phones to **perform read-only or basic GitHub repo management tasks**

- Manage, triage, and clear notifications
- Read, review, and collaborate on issues and pull requests
- Edit files in pull requests
- Search for, browse, and interact with users, repositories, and organizations
- Receive a push notification when someone mentions your username
- Search through code in a specific repository
- **Secure your GitHub.com account with two-factor authentication**
- Verify your sign in attempts on unrecognized devices

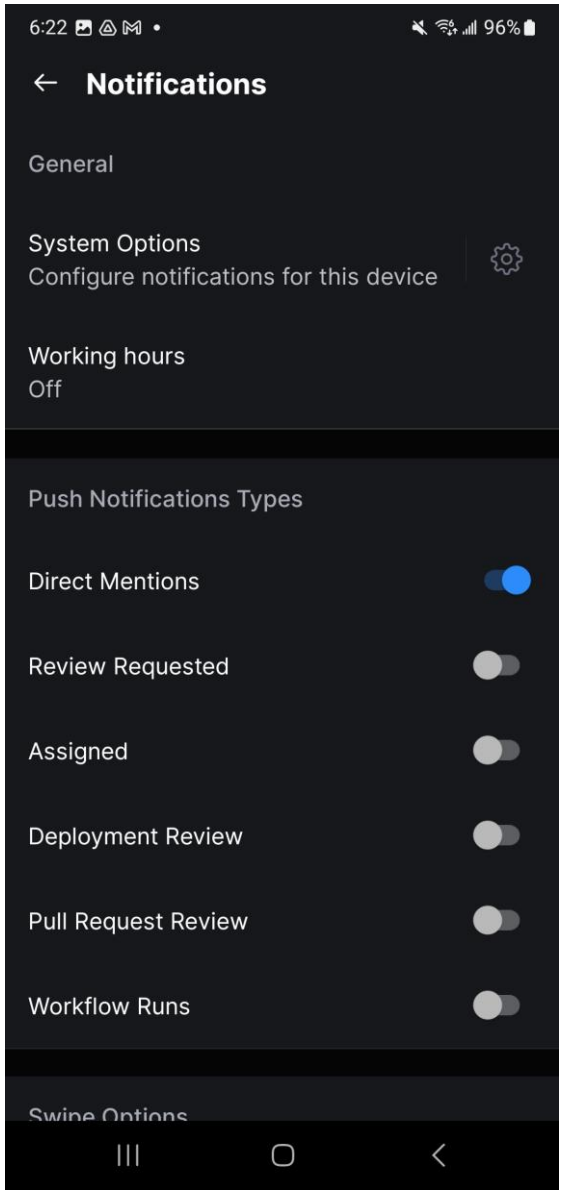






GitHub Mobile can be used for UFA, which is a convenient way over other methods

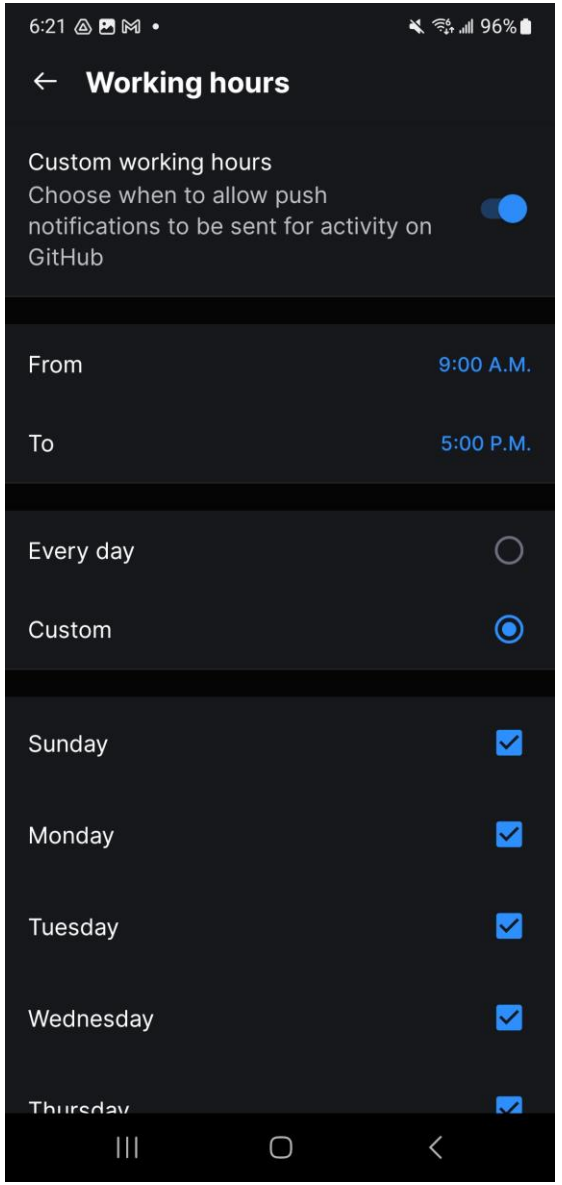
GitHub Mobile Notifications



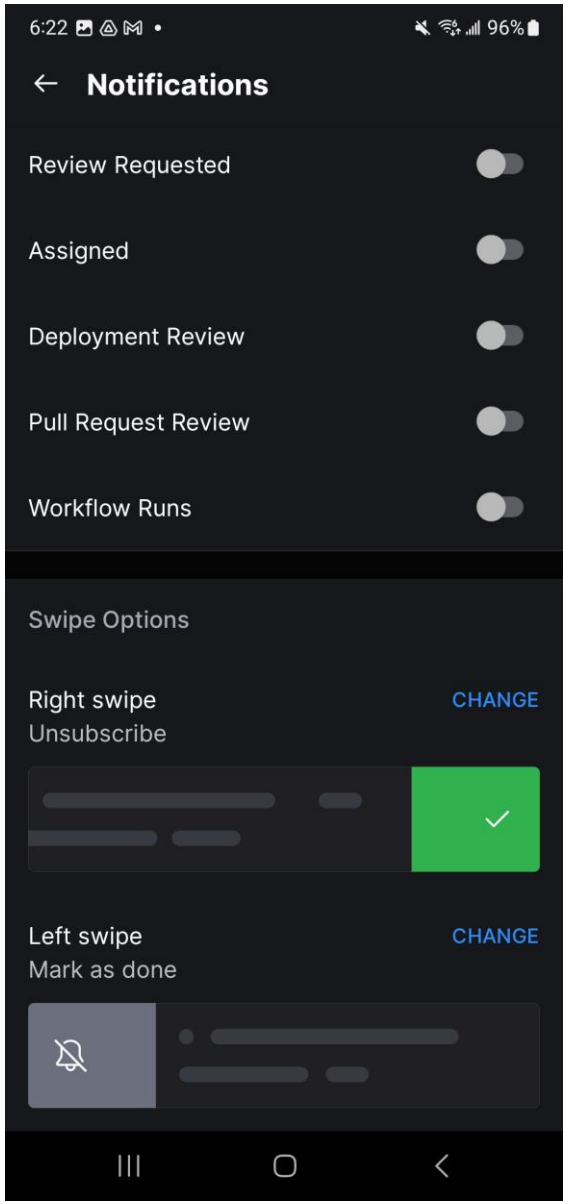
You set what **Push Notifications** you want to receive:

- Direct Mentions
- Review Requested
- Assigned
- Deployment Review
- Pull Request Review
- Workflow Runs

You set **Working Hours** to only receive push notifications for specific periods of time.



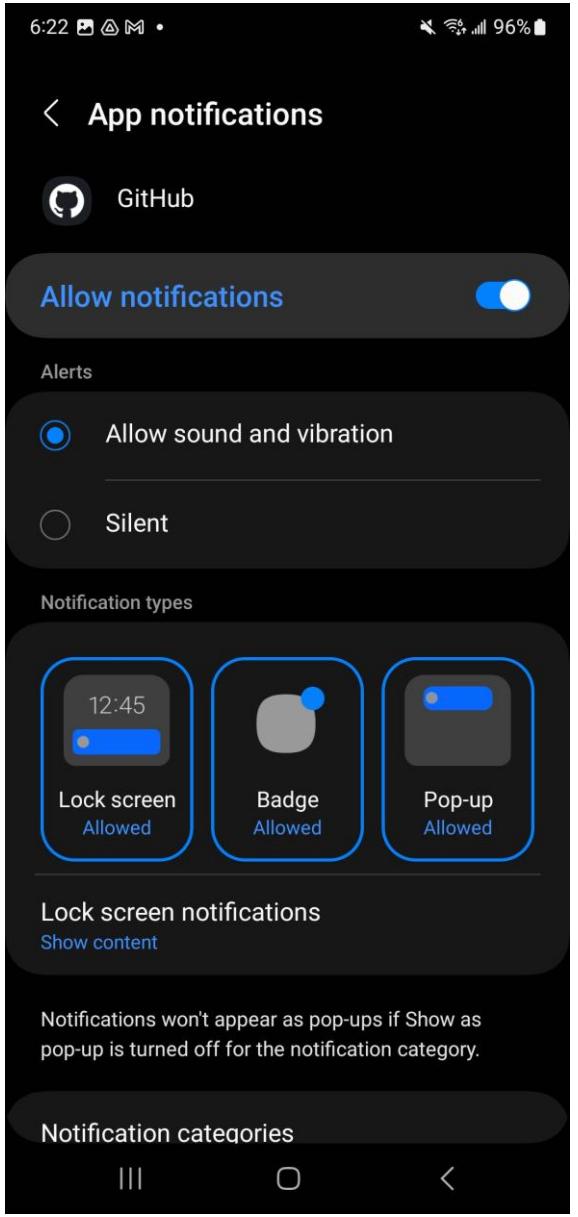
GitHub Mobile Notifications



You can set **swipe options** on notifications to perform:

- Save
- Mark as read
- Mark as done
- Unsubscribe

You can change app notifications style



Types of GitHub Accounts

There are three types of GitHub Accounts: **Personal, Organizational, Enterprise**

Personal Accounts:
These are individual accounts with a username and profile.
They can own resources like repositories and projects, and actions taken are attributed to the personal account.
They can use either GitHub Free or GitHub Pro.

Organization Accounts:
Shared accounts where multiple people collaborate on projects.
They can also own resources like repositories but are managed through individual personal accounts.
Organizations offer different roles with varying levels of access and come with security features.

Enterprise Accounts
Part of GitHub Enterprise Cloud and Server, these accounts allow for central management of multiple organizations.
They're geared towards larger setups needing centralized policy and billing management.

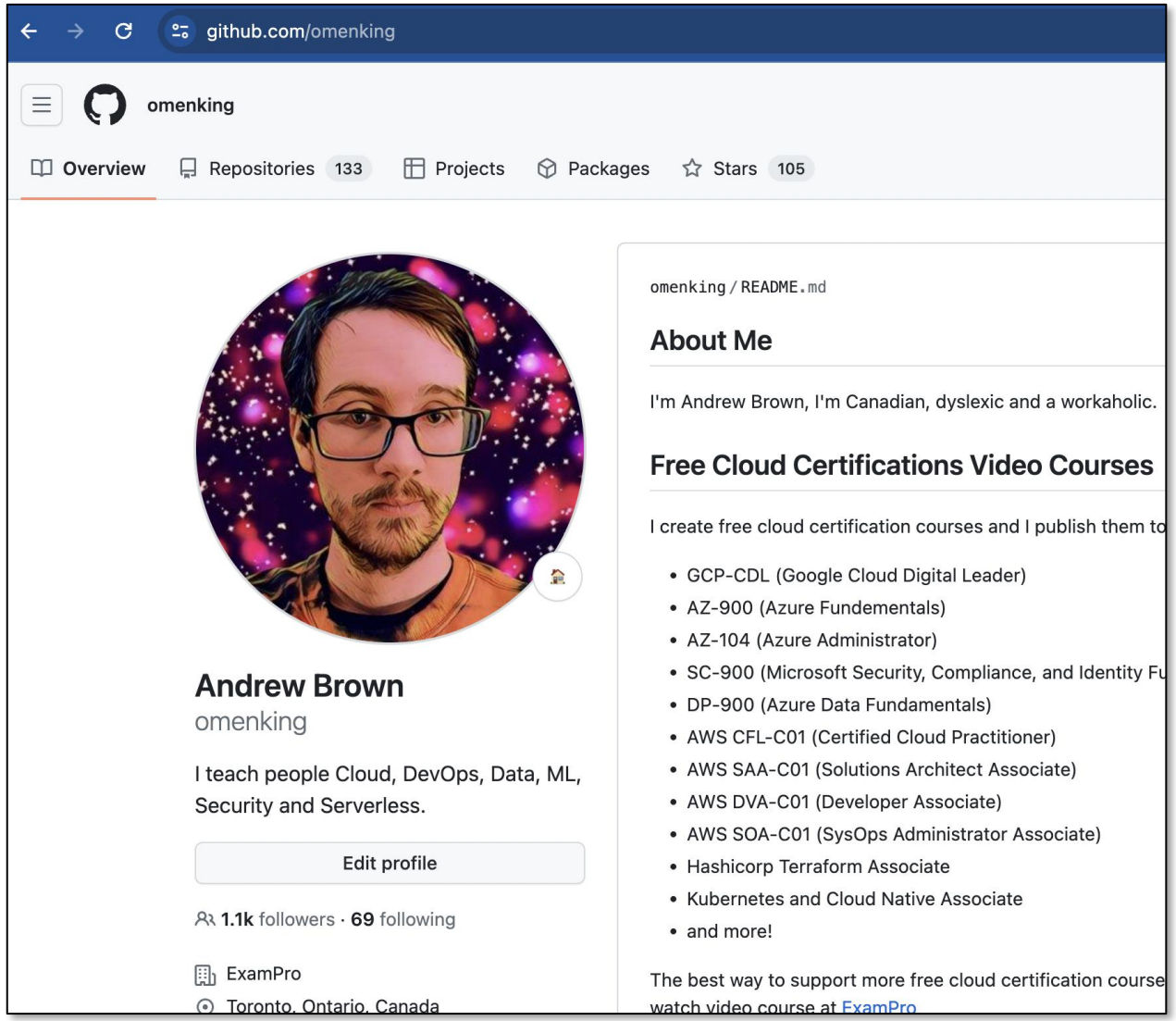
GitHub Personal

With a GitHub Personal account, you can have a **public profile** and also host your own public website.

Personal accounts have their own usernames: Eg. omenking

Repo URLs for personal accounts look like:

https://github.com/**omenking**/hello



GitHub Organization

With a GitHub Organization account can have its own **public profile** page.

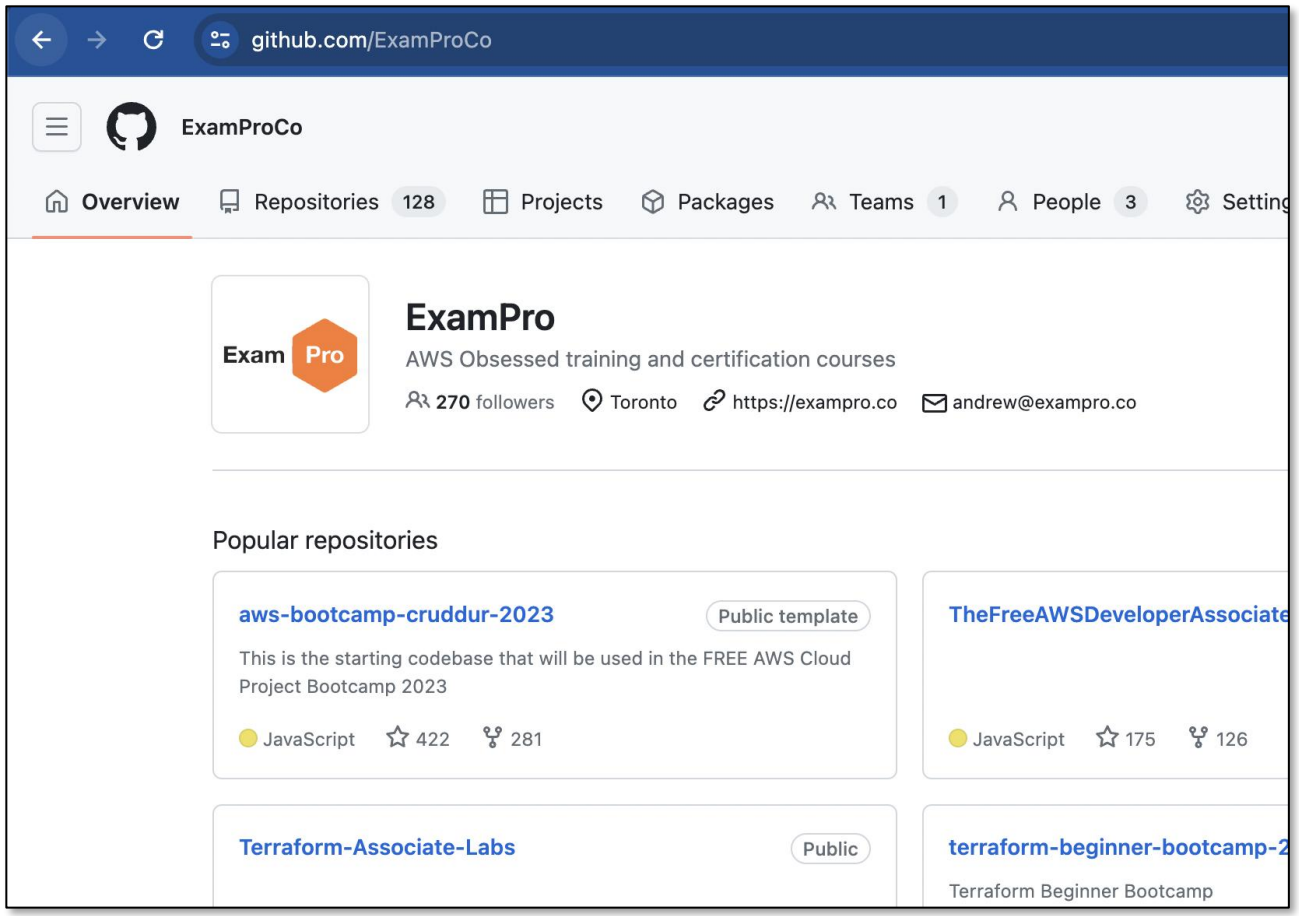
It can show who belongs within the organization.

You can manage people into teams with different levels of access

Organizations have their own usernames.
Eg. ExamPro

To access a Organization repo would look like:

https://github.com/**ExamPro**/AWS-Examples



GitHub Free vs Pro

GitHub Free

- GitHub Community Support
- Dependabot alerts
- Deployment protection rules for public repositories
- Two-factor authentication enforcement
- 500 MB GitHub Packages storage
- 120 GitHub Codespaces core hours per month
- 15 GB GitHub Codespaces storage per month
- GitHub Actions features:
  - 2,000 minutes per month
  - Deployment protection rules for public repositories

GitHub Pro

- **Everything from GitHub FREE** and….
- GitHub Support via email
- 3,000 GitHub Actions minutes per month
- 2 GB GitHub Packages storage
- 180 GitHub Codespaces core hours per month
- 20 GB GitHub Codespaces storage per month
- Advanced tools and insights in private repositories:
  - Required pull request reviewers
  - Multiple pull request reviewers
  - Protected branches
  - Code owners
  - Auto-linked references
  - GitHub Pages
- Wikis
- Repository insights graphs

GitHub Organizations Plans

GitHub Organizations has three plans: **Free, Teams and *Enterprise**

FREE

- **Everything from GitHub FREE** and….
- Team access controls for managing groups
- 2,000 GitHub Actions minutes per month
- 500 MB GitHub Packages storage

Teams

- **Everything from GitHub Organizations FREE** and….
- GitHub Support via email
- 3,000 GitHub Actions minutes per month
- 2 GB GitHub Packages storage
- Advanced tools and insights in private repositories
- Required pull request reviewers
- Multiple pull request reviewers

Teams (continued…)

- Draft pull requests
- Team pull request reviewers
- Protected branches
- Code owners
- Scheduled reminders
- GitHub Pages
- Wikis
- Repository insights graphs
- The option to enable or disable GitHub Codespaces

GitHub Enterprises Deployment Options

GitHub Enterprise includes two deployment options

- GitHub Enterprise Cloud (hosted on GitHub.com)
- GitHub Enterprise Server (self-hosted)

GitHub Enterprise (both Cloud and Server)

- **Everything from GitHub Organizations Team**
- GitHub Enterprise Support
- Additional security, compliance, and deployment controls
- Authentication with SAML single sign-on
- Access provisioning with SAML or SCIM
- Deployment protection rules with GitHub Actions for private or internal repositories
- GitHub Connect
- option to purchase GitHub Advanced Security

GitHub Enterprise Cloud (specific features)

- 50,000 GitHub Actions minutes per month
- 50 GB GitHub Packages storage
- A service level agreement for 99.9% monthly uptime
- option to centrally manage policy and billing for multiple GitHub.com organizations with an enterprise account
- option to provision and manage the user accounts for your developers, by using Enterprise Managed Users

GitHub Enterprises Deployment Options

GitHub Enterprise includes two deployment options

- GitHub Enterprise Cloud (hosted on GitHub.com)
- GitHub Enterprise Server (self-hosted)

GitHub Enterprise (both Cloud and Server)

- **Everything from GitHub Organizations Team**
- GitHub Enterprise Support
- Additional security, compliance, and deployment controls
- Authentication with SAML single sign-on
- Access provisioning with SAML or SCIM
- Deployment protection rules with GitHub Actions for private or internal repositories
- GitHub Connect
- option to purchase GitHub Advanced Security

GitHub Enterprise Cloud (specific features)

- 50,000 GitHub Actions minutes per month
- 50 GB GitHub Packages storage
- A service level agreement for 99.9% monthly uptime
- option to centrally manage policy and billing for multiple GitHub.com organizations with an enterprise account
- option to provision and manage the user accounts for your developers, by using Enterprise Managed Users

Markdown

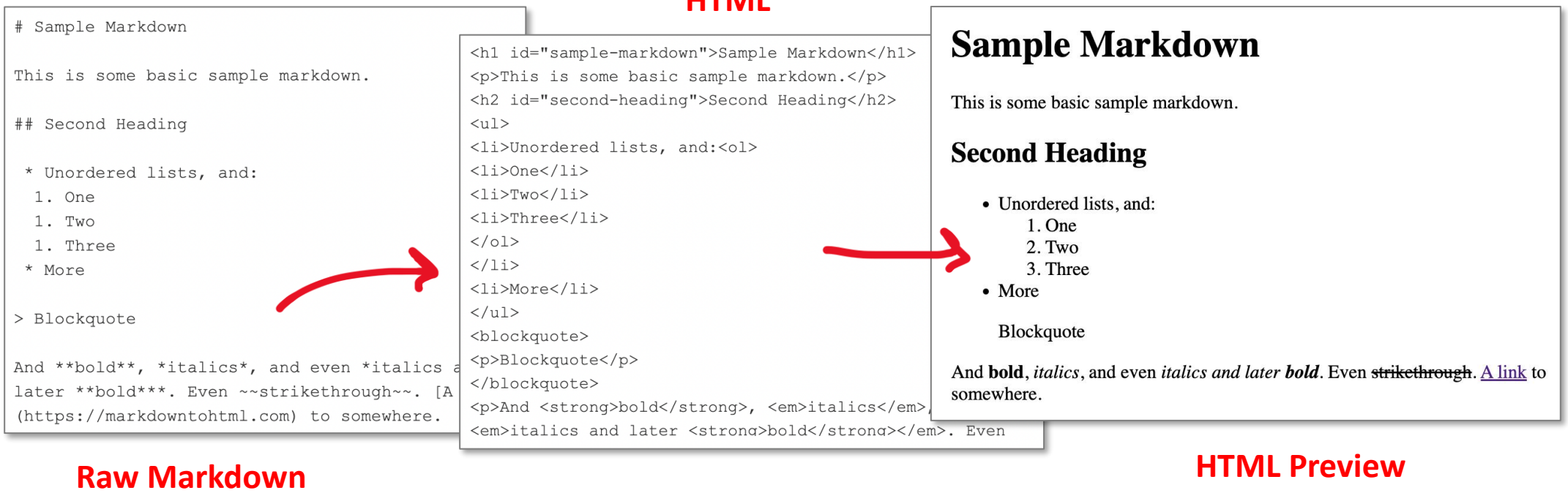
What is a Markup language?

A Markup language is a way of formatting and presenting text data in a different format.

*A common use-case for a markup language is to present data in **HTML**.*

Markdown is markup language that provides a shorthand syntax to format information into HTML.
Markdown is popular due to its easy syntax, and being readable in its raw format.

Markdown files extensions are **.md** and **.markdown**



Markdown Basic Syntax

All markdown parsers supports the following markdown syntax

Headings
H1
H2
H3

Text Formatting
italicized text
bold text

Blockquote
> blockquote

Unordered List
- First item
- Second item
- Third item

Ordered List
1. First item
1. Second item
1. Third item

the numbers will change to be the correct numbers

Link
[title](https://www.example.com)

Horizontal Rule

Image
![alt text](image.jpg)

Code Inline
`puts “hello world”`

codeblocks use backticks and not single quotes.
Backticks key is above the tab key on the keyboard

Markdown Extended Syntax

Markdown parsers may support the following extended markdown syntax

Table

```
| Syntax | Description |
| ---------- | ---------- |
| Header | Title |
| Paragraph | Text |
```

Fenced Code Blocks

````
```rb
def hello_world
  puts “hello world”
end
```
````

You can sometimes have syntax highlighting for codeblocks by telling the codeblock the language being used.

Heading ID

```
### My Great Heading {#custom-id}
```

Definition List

```
term
: definition
```

Strikethrough

```
~~The world is flat.~~
```

Footnote

```
Here's a sentence with a footnote. [^1]

[^1]: This is the footnote.
```

Task List

```
- [x] Write the press release
- [ ] Update the website
- [ ] Contact the media
```

Emoji

```
That is so funny! :joy:
```

Highlight

```
I need to highlight these ==very important words==.
```

Subscript

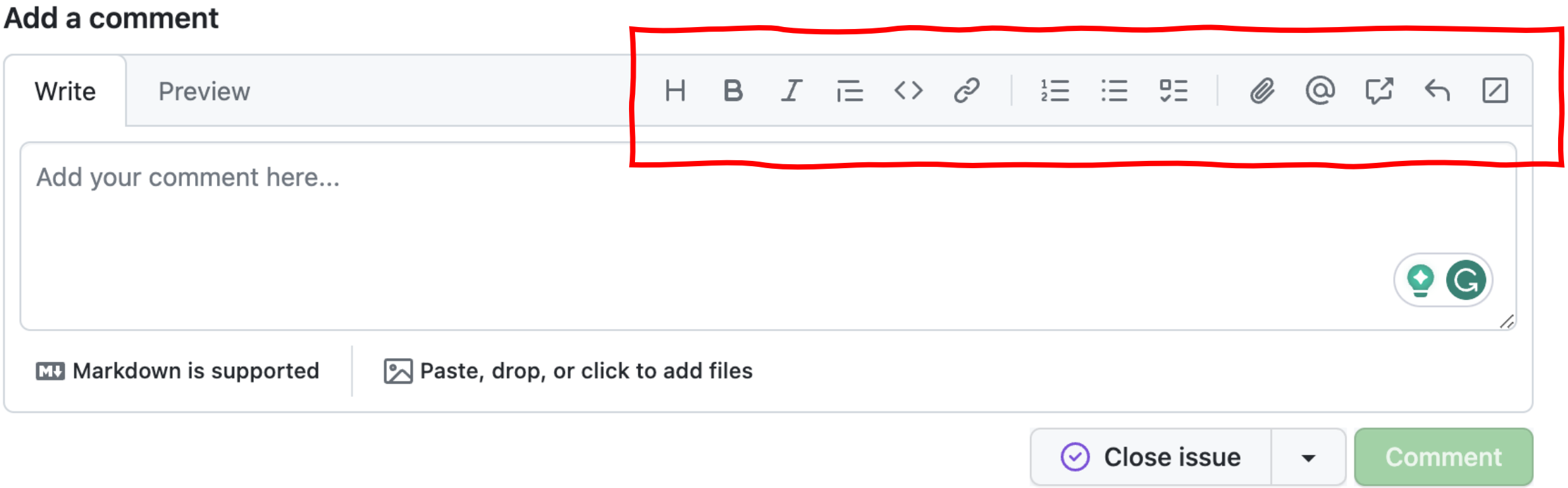
```
H~2~O
```

Superscript

```
X^2^
```

Text Formatting Toolbar

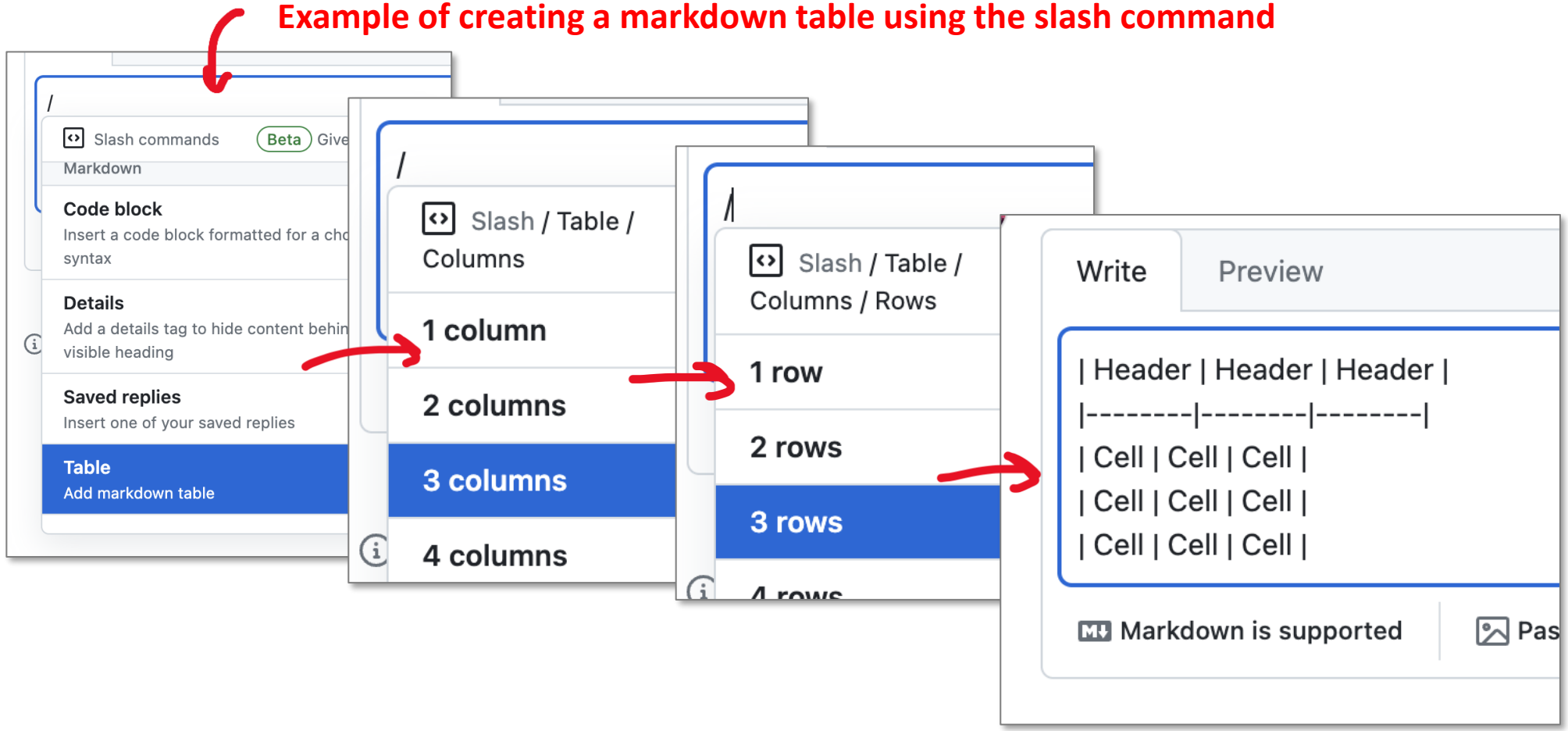
Every comment field on GitHub contains a **text formatting toolbar**, which allows you to format your text without learning GitHub-flavoured Markdown syntax



Slash Commands

Slash Commands (within comment boxes) on GitHub.com provides convenience features such as formatting markdown.

Example of creating a markdown table using the slash command



GitHub Flavoured Markdown (gfm)

GitHub Flavoured Markdown (GFM) is the **dialect of Markdown** that is currently **supported for user content on GitHub.com and GitHub Enterprise**

GFM provides powerful functionality:

- Collapsible sections
- Embed mathematical expressions
- Embed diagrams
- Relative paths or linking to files in the same repo
- Tasks lists can be converted into Issues
- Extended formatting for tables
- Shorthands to autolink to issues, pull requests and repos
- Render code snippets from other codesbase via linking
- And more…

The detailed **GFM spec** is located here:

https://github.github.com/gfm/

GitHub Flavoured Markdown (gfm)

Supports embedded syntax within tables

```
| Command | Description |
| --- | --- |
| `git status` | List all *new or modified* files |
| `git diff` | Show file differences that **haven't been** staged |
```

Command	Description
`git status`	List all *new or modified* files
`git diff`	Show file differences that **haven't been** staged

Supports advanced cell alignment

```
| Left-aligned | Center-aligned | Right-aligned |
| :---         |     :---:      |          ---: |
| git status   | git status     | git status    |
| git diff     | git diff       | git diff      |
```

Left-aligned	Center-aligned	Right-aligned
git status	git status	git status
git diff	git diff	git diff

GitHub Flavoured Markdown (gfm)

The **<details>** tag can be used to create collapsible sections

```
<details>

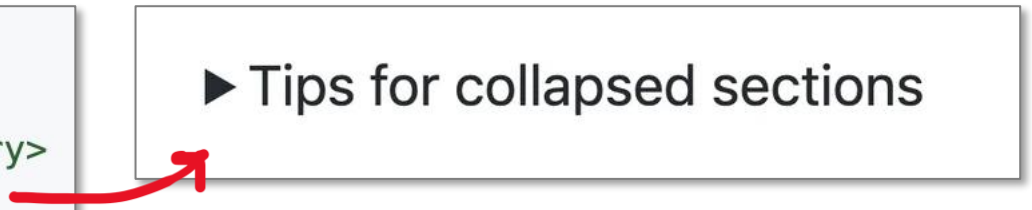
<summary>Tips for collapsed sections</summary>

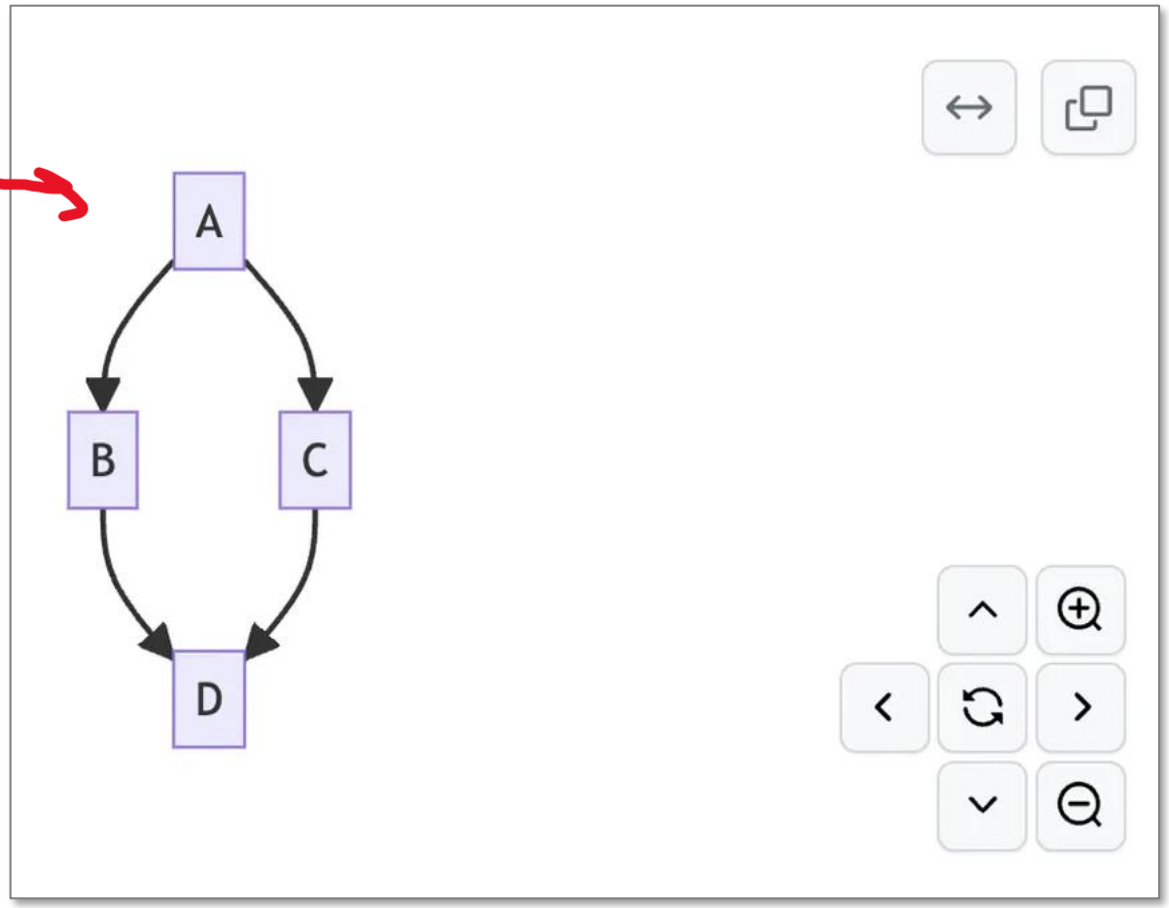
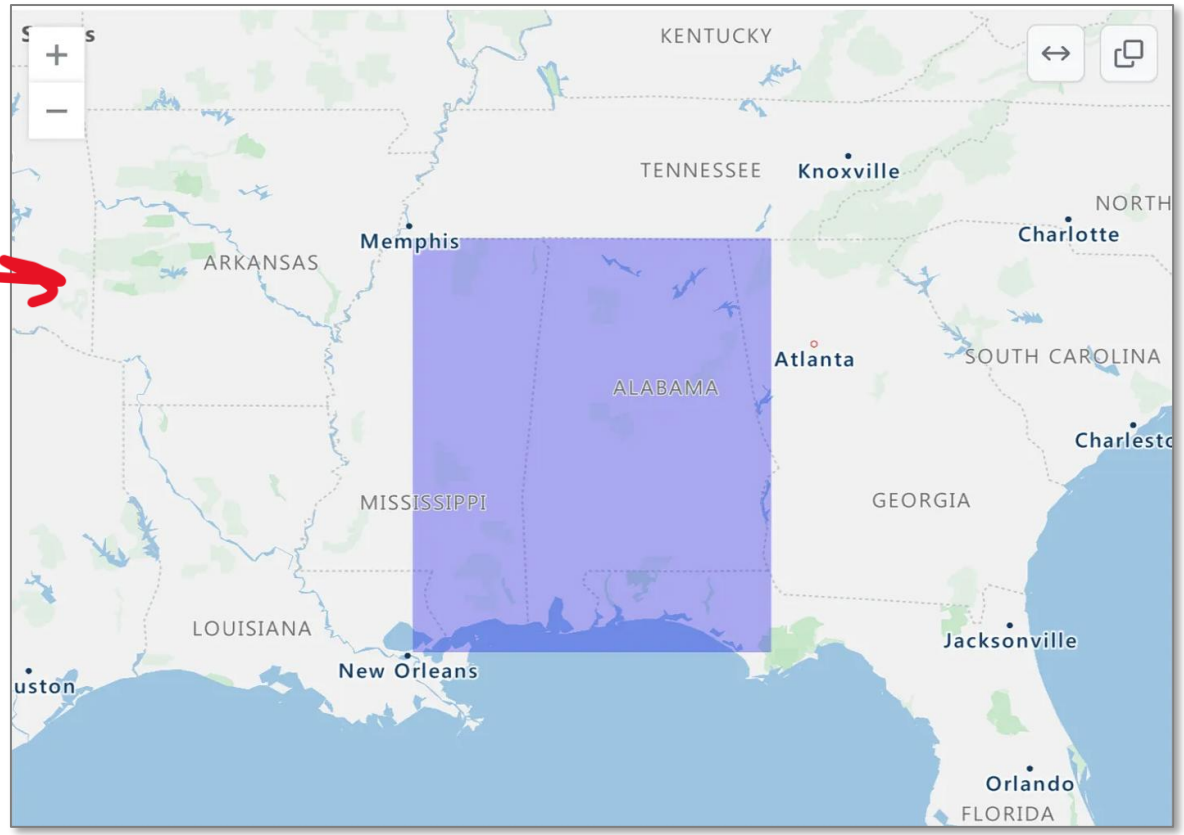
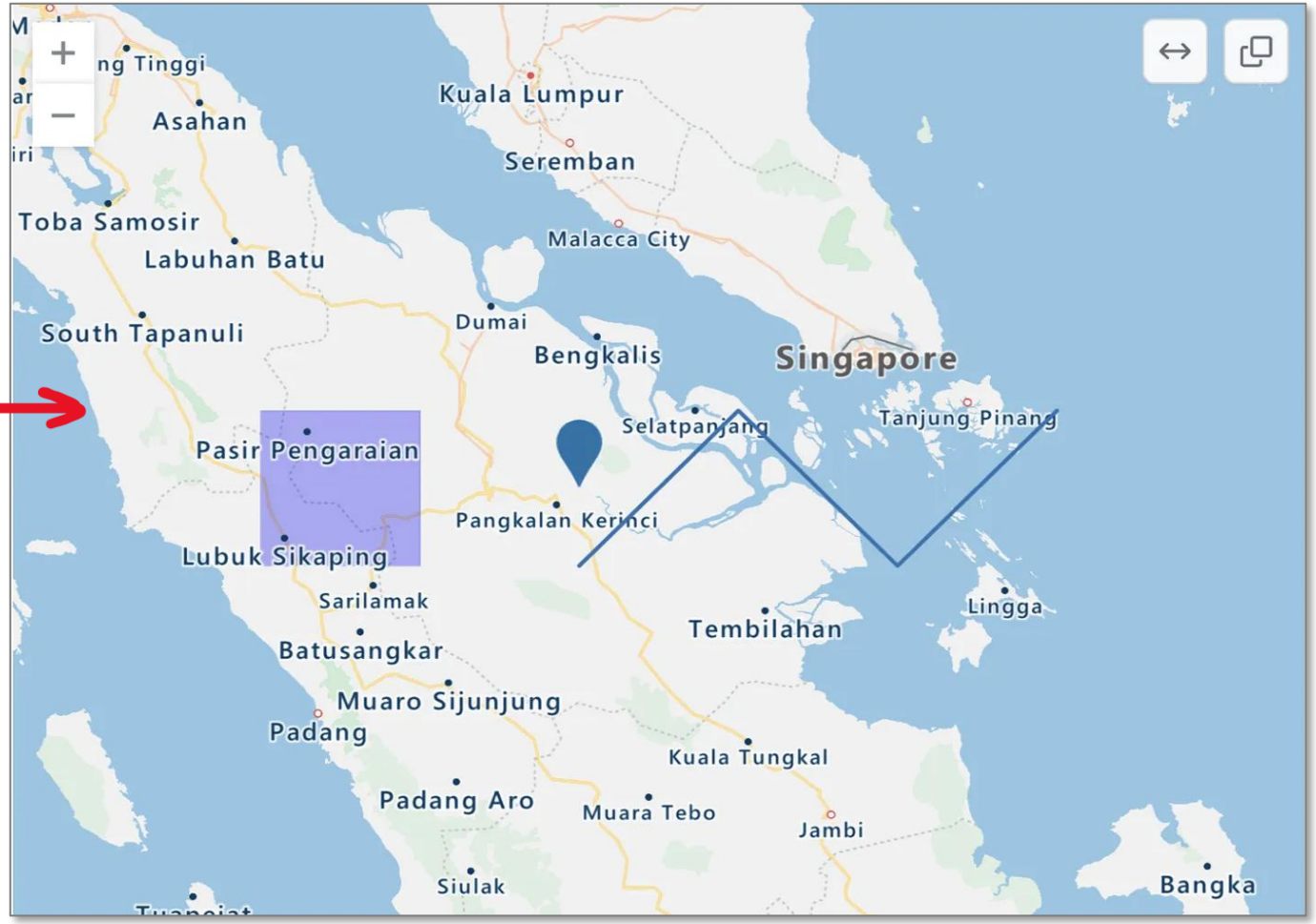
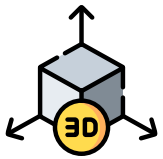
### You can add a header

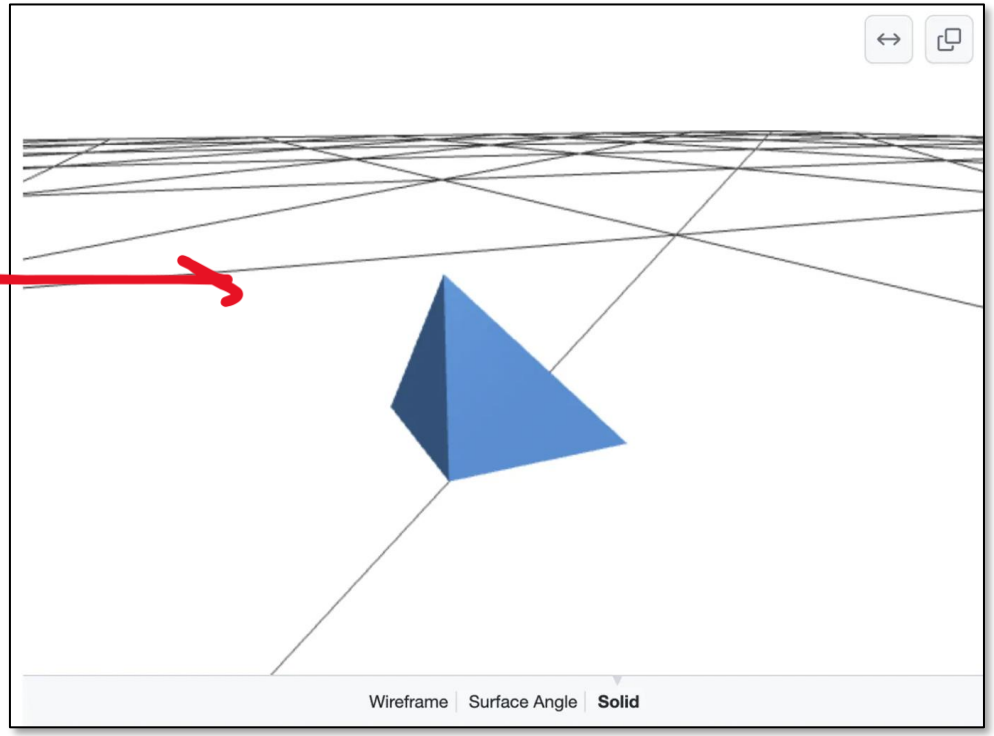
You can add text within a collapsed section.

You can add an image or a code block, too.

```ruby
   puts "Hello World"
```

</details>
```



GitHub Flavoured Markdown (gfm)

Mermaid is a Markdown-inspired tool that renders text into diagrams

````
Here is a simple flow chart:

```mermaid
graph TD;
    A-->B;
    A-->C;
    B-->D;
    C-->D;
```
````



GitHub Flavoured Markdown (gfm)

GeoJSON or TopoJSON syntax to create interactive maps.

````
```geojson
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "id": 1,
      "properties": {
        "ID": 0
      },
      "geometry": {
        "type": "Polygon",
        "coordinates": [
          [
              [-90,35],
              [-90,30],
              [-85,30],
              [-85,35],
              [-90,35]
          ]
        ]
      }
    }
  ]
}
```
````



Geojson example

GitHub Flavoured Markdown (gfm)

```topojson
{
  "type": "Topology",
  "transform": {
    "scale": [0.0005000500050005, 0.0001000100010001000
    "translate": [100, 0]
  },
  "objects": {
    "example": {
      "type": "GeometryCollection",
      "geometries": [
        {
          "type": "Point",
          "properties": {"prop0": "value0"},
          "coordinates": [4000, 5000]
        },
        {
          "type": "LineString",
          "properties": {"prop0": "value0", "prop1": 0}
          "arcs": [0]
        },
        {
          "type": "Polygon",
          "properties": {"prop0": "value0",
            "prop1": {"this": "that"}
          },
          "arcs": [[1]]
        }
      ]
    }
  },
  "arcs": [[[4000, 0], [1999, 9999], [2000, -9999], [20
}
```



Topojson example

GitHub Flavoured Markdown (gfm)



ASCII STL syntax directly in markdown to create interactive 3D models

````
```stl
solid cube_corner
  facet normal 0.0 -1.0 0.0
    outer loop
      vertex 0.0 0.0 0.0
      vertex 1.0 0.0 0.0
      vertex 0.0 0.0 1.0
    endloop
  endfacet
  facet normal 0.0 0.0 -1.0
    outer loop
      vertex 0.0 0.0 0.0
      vertex 0.0 1.0 0.0
      vertex 1.0 0.0 0.0
    endloop
  endfacet
  facet normal -1.0 0.0 0.0
    outer loop
      vertex 0.0 0.0 0.0
      vertex 0.0 0.0 1.0
      vertex 0.0 1.0 0.0
    endloop
  endfacet
  facet normal 0.577 0.577 0.577
    outer loop
      vertex 1.0 0.0 0.0
      vertex 0.0 1.0 0.0
      vertex 0.0 0.0 1.0
    endloop
  endfacet
endsolid
```
````



GitHub Flavoured Markdown (gfm)

GFM can render mathematical expressions via Latex formatted math.

GFM uses MathJax underneath as the tool to render

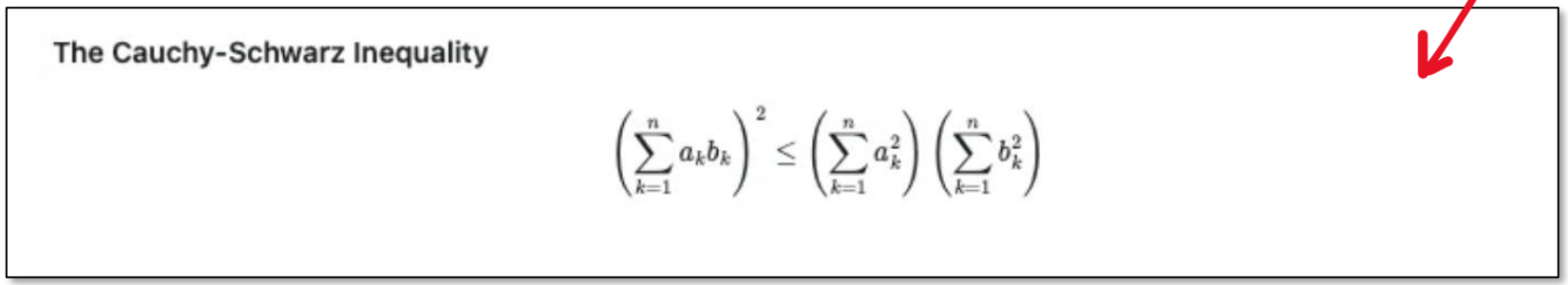
Use the **$$** to denote start and end of latex

```
**The Cauchy-Schwarz Inequality**
$$\left( \sum_{k=1}^n a_k b_k \right)^2 \leq \left( \sum_{k=1}^n a_k^2 \right)
\left( \sum_{k=1}^n b_k^2 \right)$$
```

Or use the ```match Fencing block

```
**The Cauchy-Schwarz Inequality**

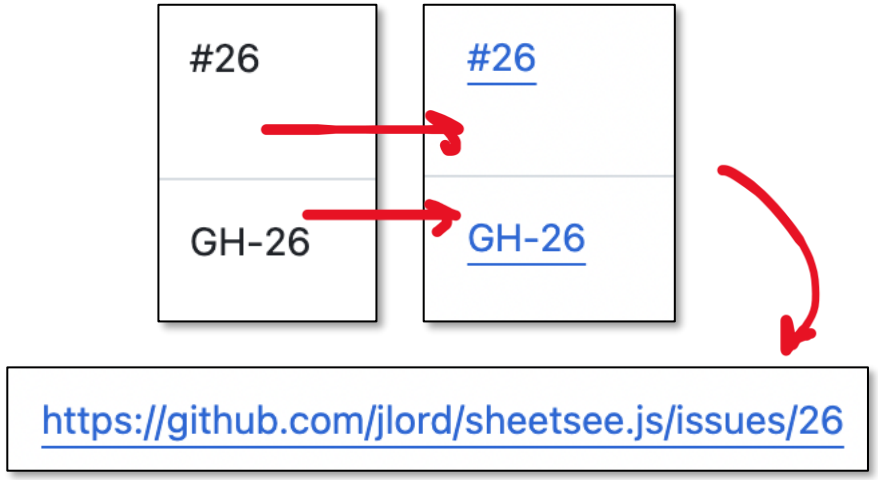
```math
\left( \sum_{k=1}^n a_k b_k \right)^2 \leq \left( \sum_{k=1}^n a_k^2 \right) \left(
\sum_{k=1}^n b_k^2 \right)
```
```



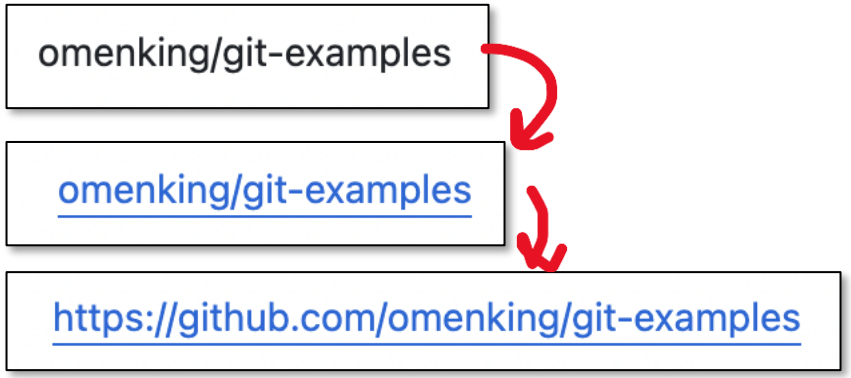
GitHub Flavoured Markdown (gfm)

GFM will turn raw link automatically into a link

Within comments you can quickly link to issues and pull requests



Within comments you can quickly link to repos:



Commit SHAs can be autolinked as well:

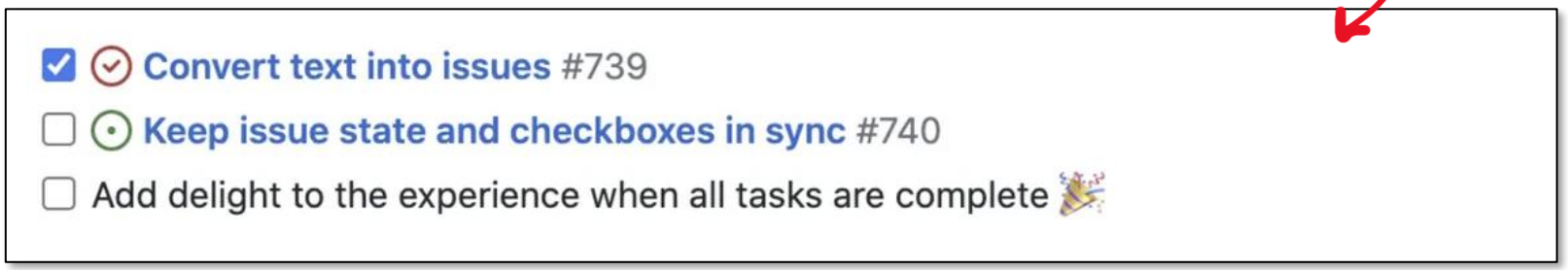
Reference type	Raw reference	Short link
Commit URL	`https://github.com/jlord/sheetsee.js/commit/a5c3785ed8d6a35868bc169f07e40e889087fd2e`	a5c3785
SHA	a5c3785ed8d6a35868bc169f07e40e889087fd2e	a5c3785
User@SHA	jlord@a5c3785ed8d6a35868bc169f07e40e889087fd2e	jlord@a5c3785
`Username/Repository@SHA`	`jlord/sheetsee.js@a5c3785ed8d6a35868bc169f07e40e889087fd2e`	`jlord/sheetsee.js@a5c3785`

GitHub Flavoured Markdown (gfm)

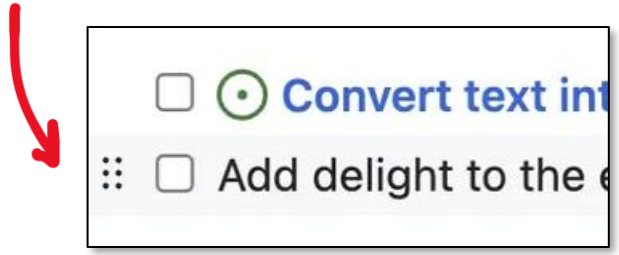
Tasks lists can within an open issue can be created.

- You can associate tasks to specific issues (creating subtasks)
- GitHub can keep track of open of closed state of linked issue
- You can onclick turn a task and associate with an issue

```
- [x] #739
- [ ] https://github.com/octo-org/octo-repo/issues/740
- [ ] Add delight to the experience when all tasks are complete :tada:
```

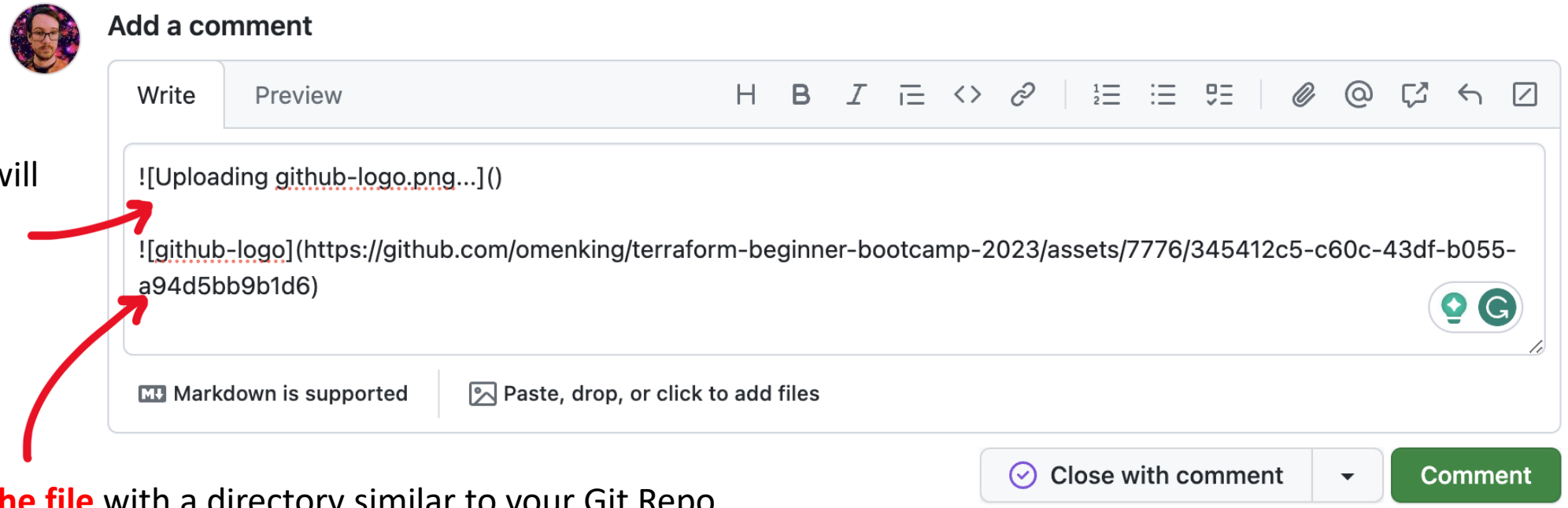


You can also easily reorder tasks



Uploading Files

You can directly upload files by dragging them into the markdown supported textareas in GitHub



While its upload it will show a **placeholder**

GitHub will **host the file** with a directory similar to your Git Repo

- Assume files you upload for public repos are public.

Uploading Files

The maximum file size is:

- **10MB** for images and gifs
- **10MB** for videos uploaded to a repository owned by a user or organization on a free GitHub plan
- **100MB** for videos uploaded to a repository owned by a user or organization on a paid GitHub plan
- **25MB** for all other files

Supported files:

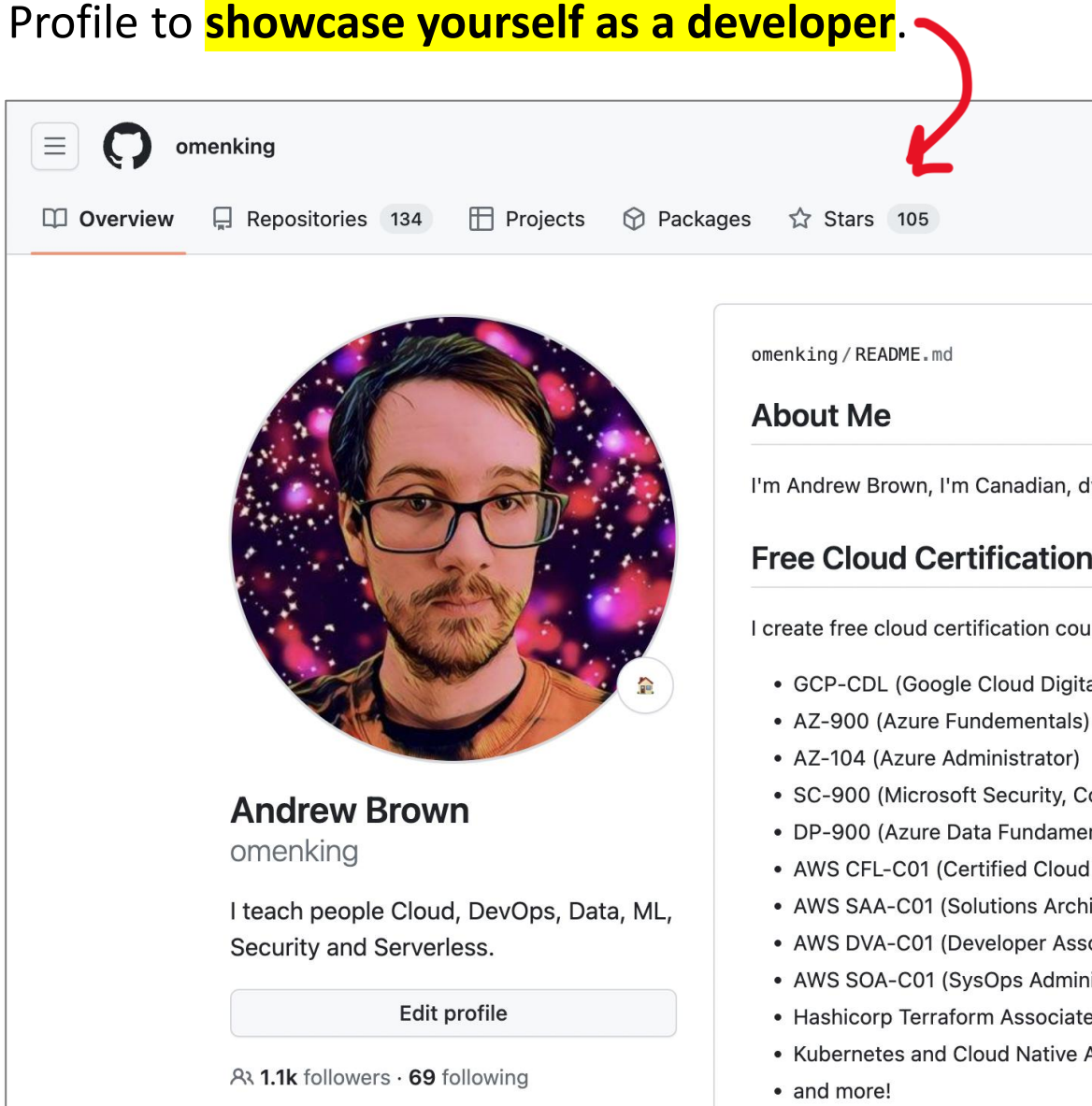
- PNG (*.png*)
- GIF (*.gif*)
- JPEG (*.jpg*, *.jpeg*)
- SVG (*.svg*)
- Log files (*.log*)
- Markdown files (*.md*)
- Microsoft Word (*.docx*)
- PowerPoint (*.pptx*)
- Excel (*.xlsx*) documents
- Text files (*.txt*)
- PDFs (*.pdf*)
- ZIP (*.zip*, *.gz*, *.tgz*)
- Video (*.mp4*, *.mov*, *.webm*)

GitHub User Profile Features

For Personal Accounts you can have a public GitHub User Profile to **showcase yourself as a developer**.

Features of your GitHub user profile include:

- **Profile Picture and Bio:** You can upload a profile picture and write a short bio about yourself.
- **User Profile Readme:** a readme file rendered on your profile.
- **Repositories:** Showcases your repositories, including options to pin your favorite ones.
- **Contributions Graph:** A graph displaying your contributions over the past year
  - (commits, pull requests, issues, and code reviews).
- **Followers and Following:** Displays the number of followers you have and the number of users you are following.
- **Stars:** Shows repositories you have starred.
- **Organizations:** Lists the GitHub organizations you are a part of.
- **Contribution Activity:** Detailed list of recent activity on GitHub.

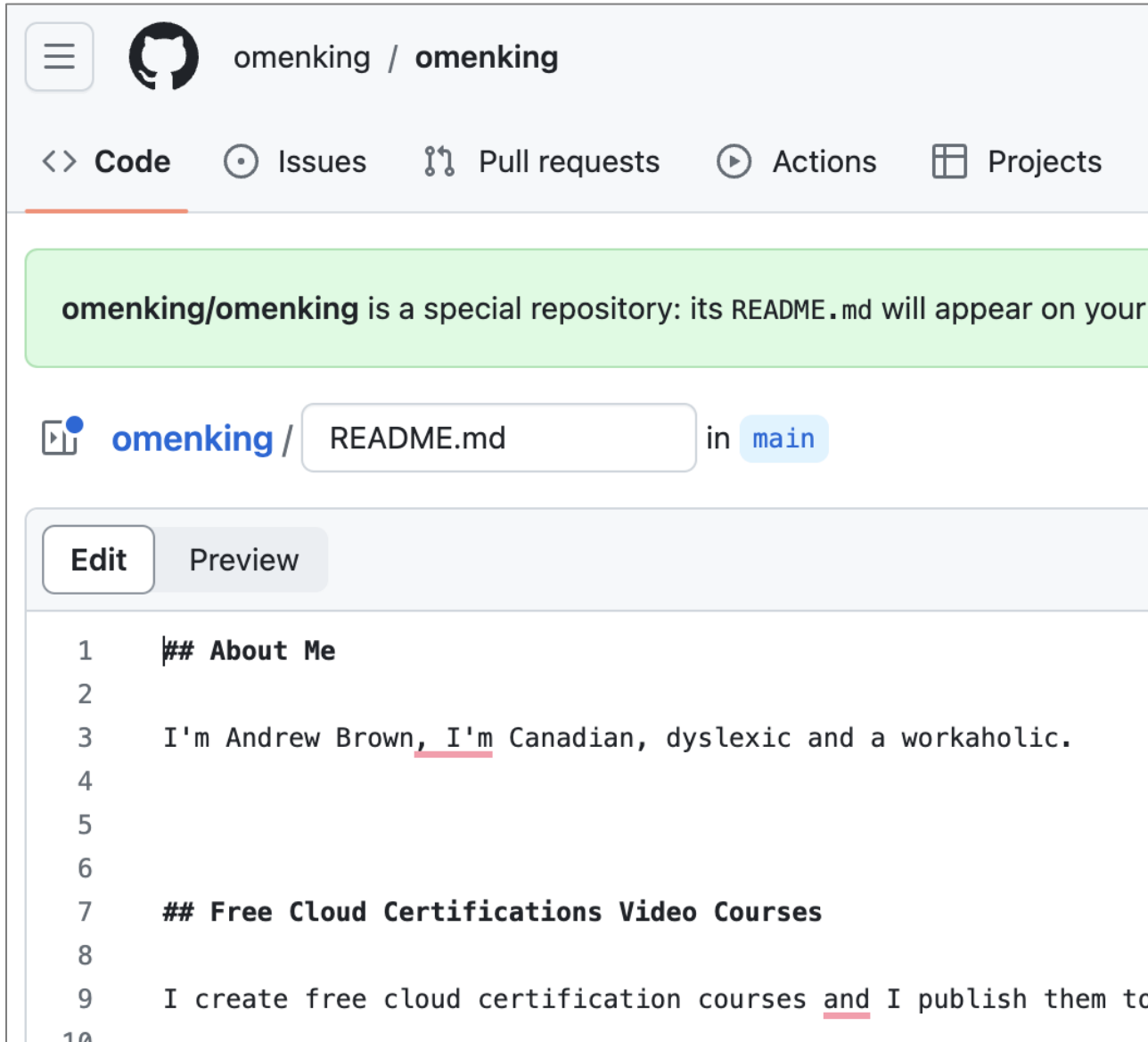


GitHub User Profile Features

The user profile readme allows you place whatever you like on your profile.

In order to create a user profile readme you need to:

- Create a public repo with with your username.
  - eg. Omenking/omenking
- Create a README.md in the repo

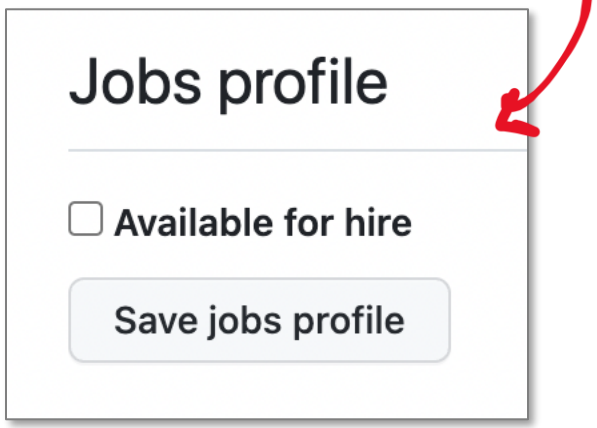


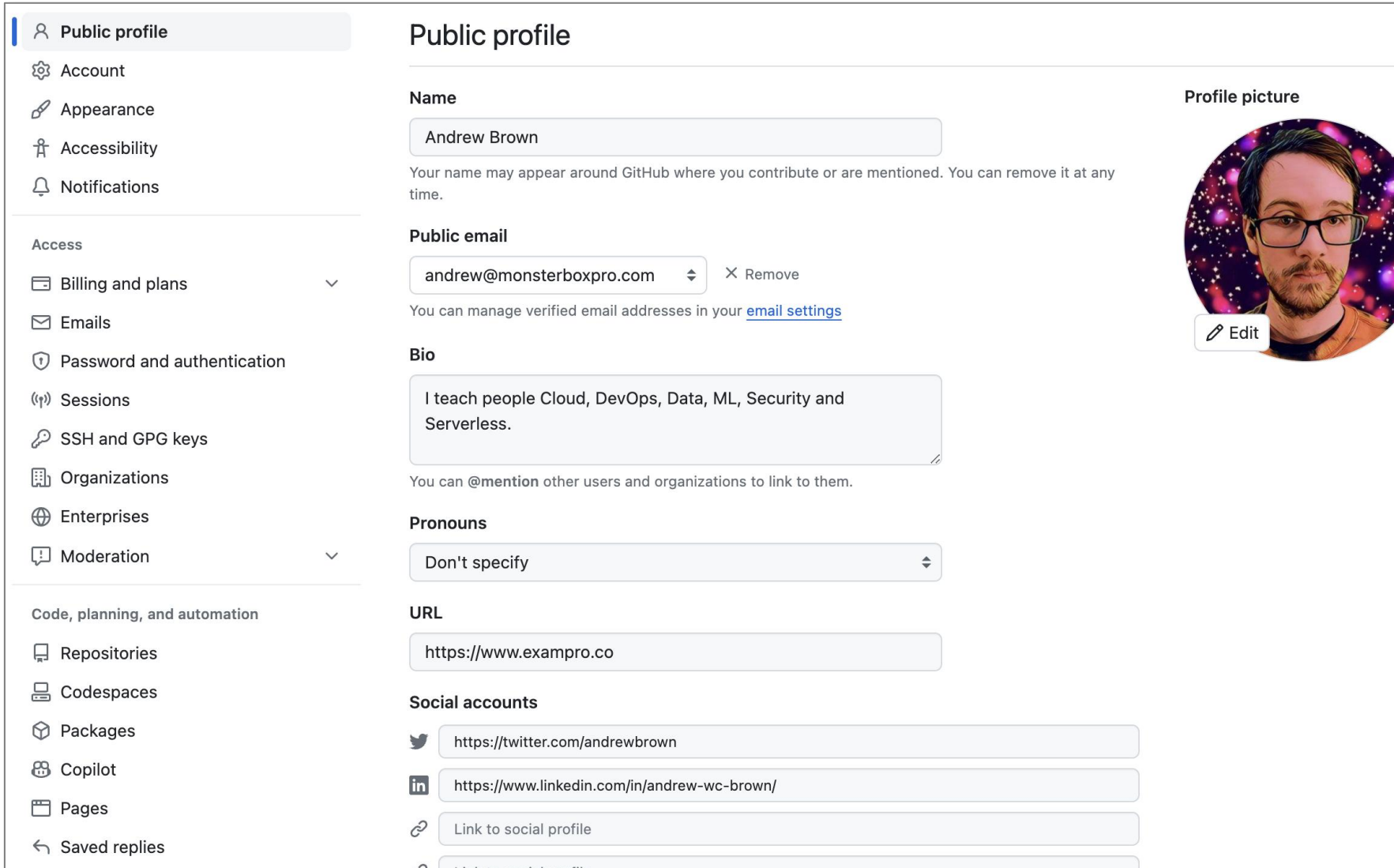
GitHub User Profile Features

In your account settings under Public Profile you can change:

- Display name
- Bio
- URL
- Social Account URLs

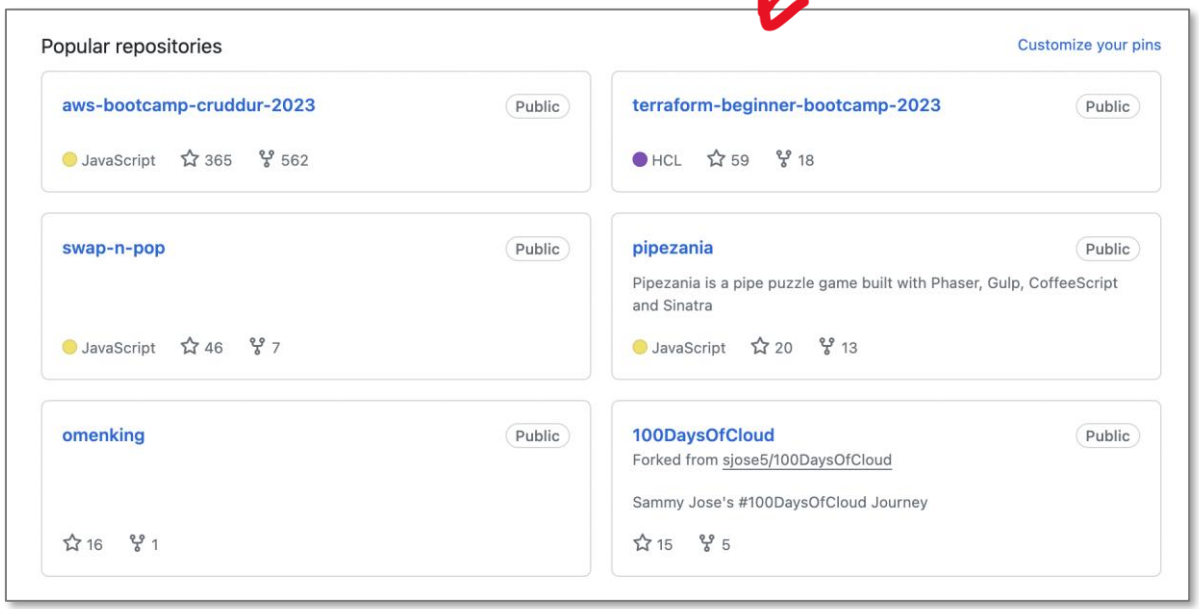
You can also specify if you are looking for work

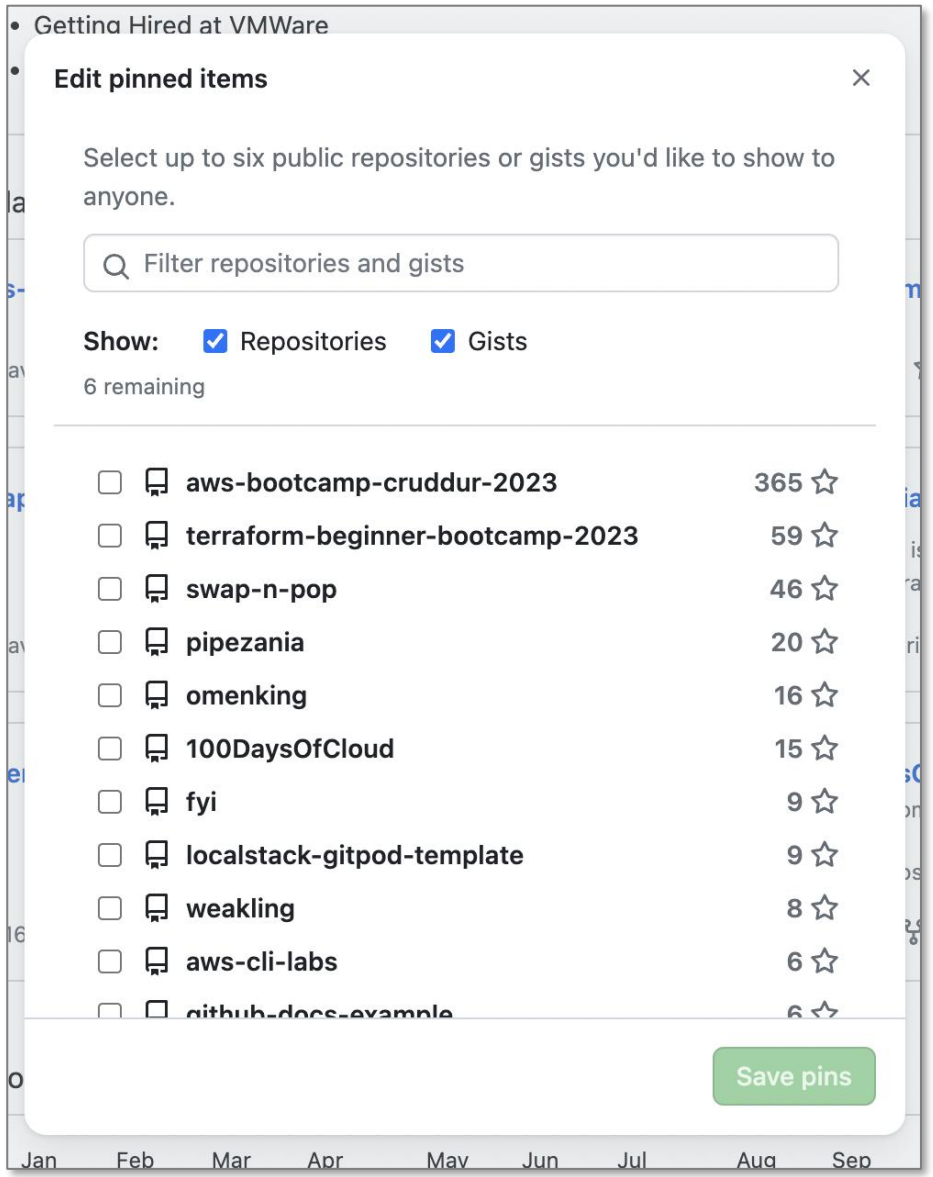




GitHub User Profile Features

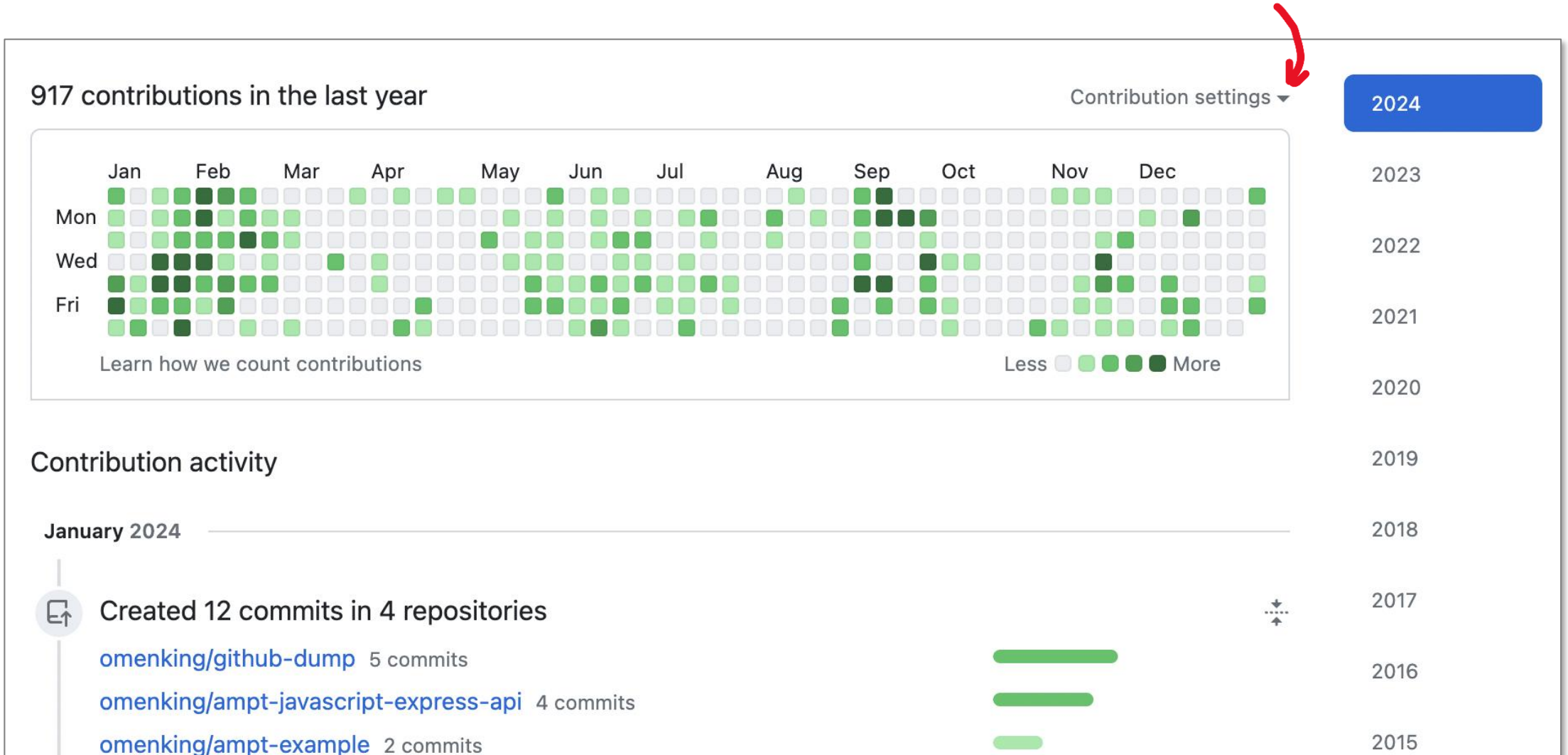
You can pin specific GitHub repos that own to best **showcase your projects**





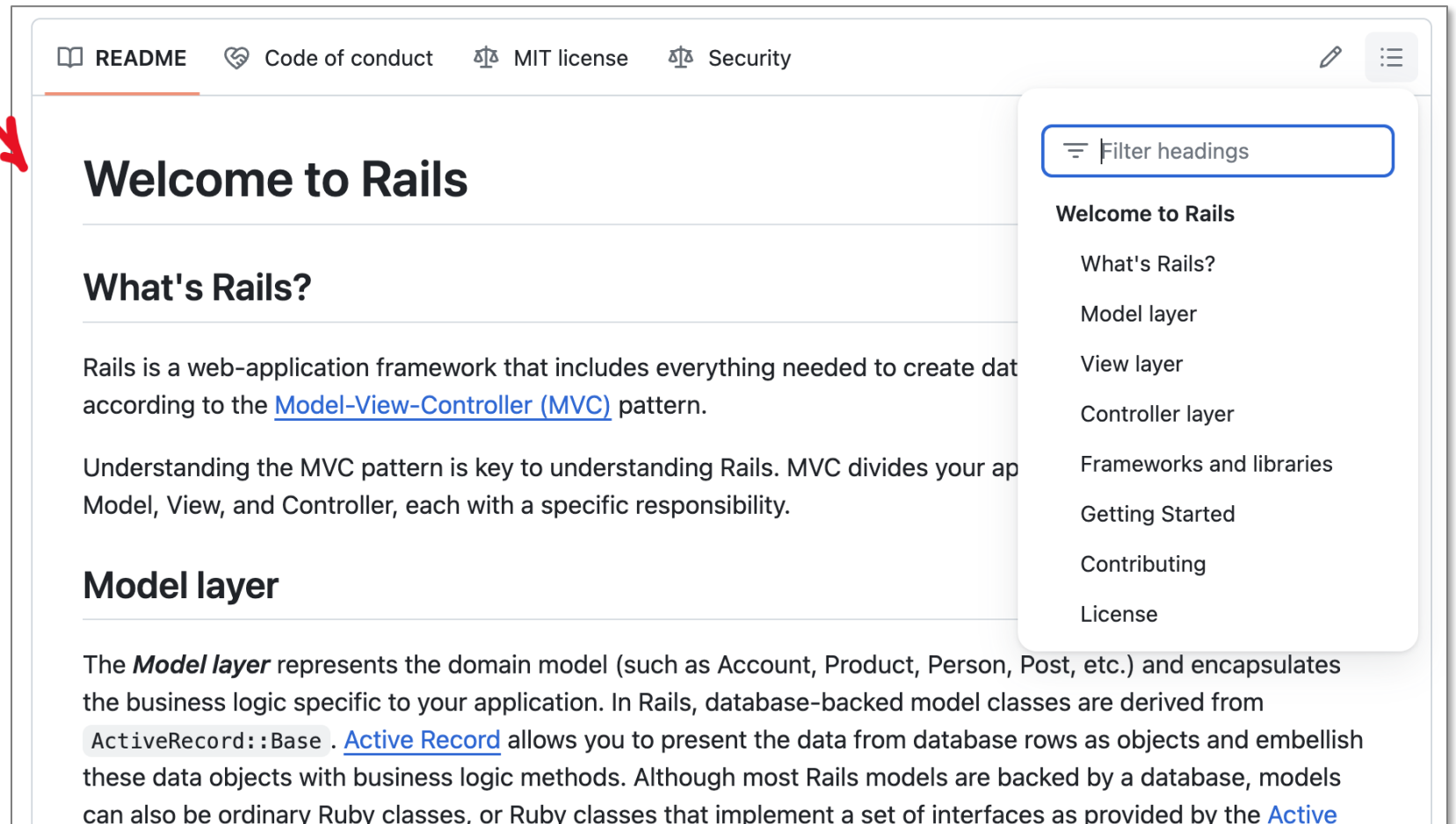
GitHub User Profile Features

Contributor Activity in your profile will show **how active (for public projects) you are on GitHub.**



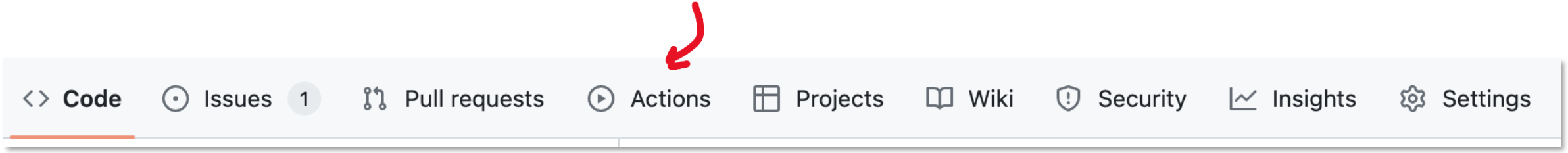
README Files

Readme files are markdown files that provided documentation/instructional information.
A repo that has a **Readme.md or README file in project's root** will be rendered on the GitHub Repo page for each access.



Basic Repo Navigation

Within a GitHub repo you'll have a **navigation bar** to the various feature of your GitHub repo:



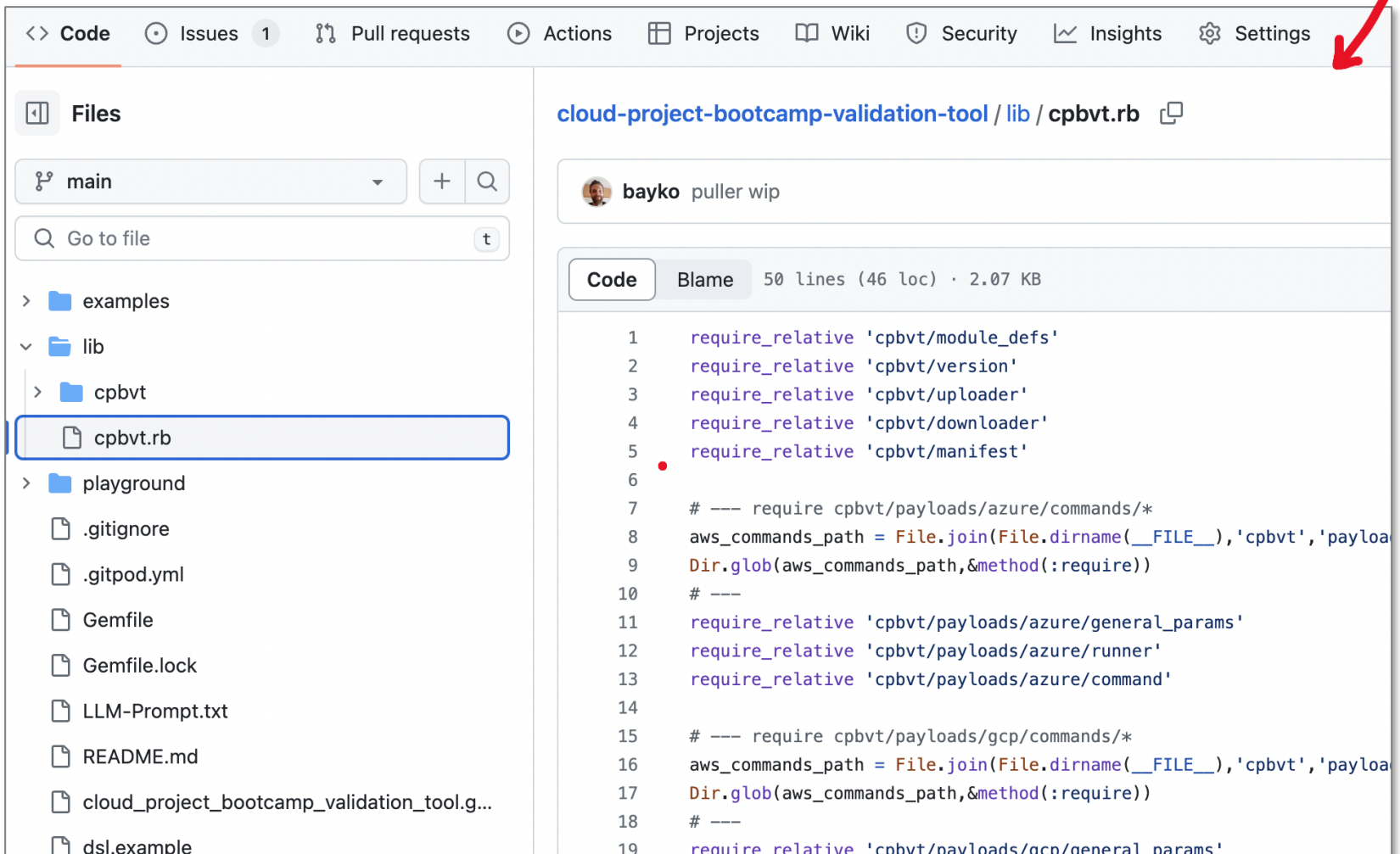
- **Code:** The main tab where the repository's source code, files, and folders are located.
- **Issues:** Tracks problems or ideas for the project, allowing collaboration and discussion.
- **Pull Requests:** Used for managing contributions from other users, enabling code review and discussion before merging changes.
- **Actions:** Manages continuous integration and continuous deployment (CI/CD) workflows.
- **Projects:** A board for organizing and prioritizing work, similar to kanban or task management boards.
- **Wiki:** A space for the project's documentation.
- **Security:** Features security-related resources, including security policies and vulnerability reports.
- **Insights:** Provides statistics and insights on the project's activity and contributions.
- **Settings:** Contains repository settings, including access controls, webhooks, and other configurations

Basic Repo Navigation

Within a GitHub Repo when you click into a file you can then easily navigate all repo files

You can see preview of the **file content**

You can jump to **specific files** using the search



Creating a new repo

When you create a repo, you choose an owner:

- Personal account
- An organization that you belong to

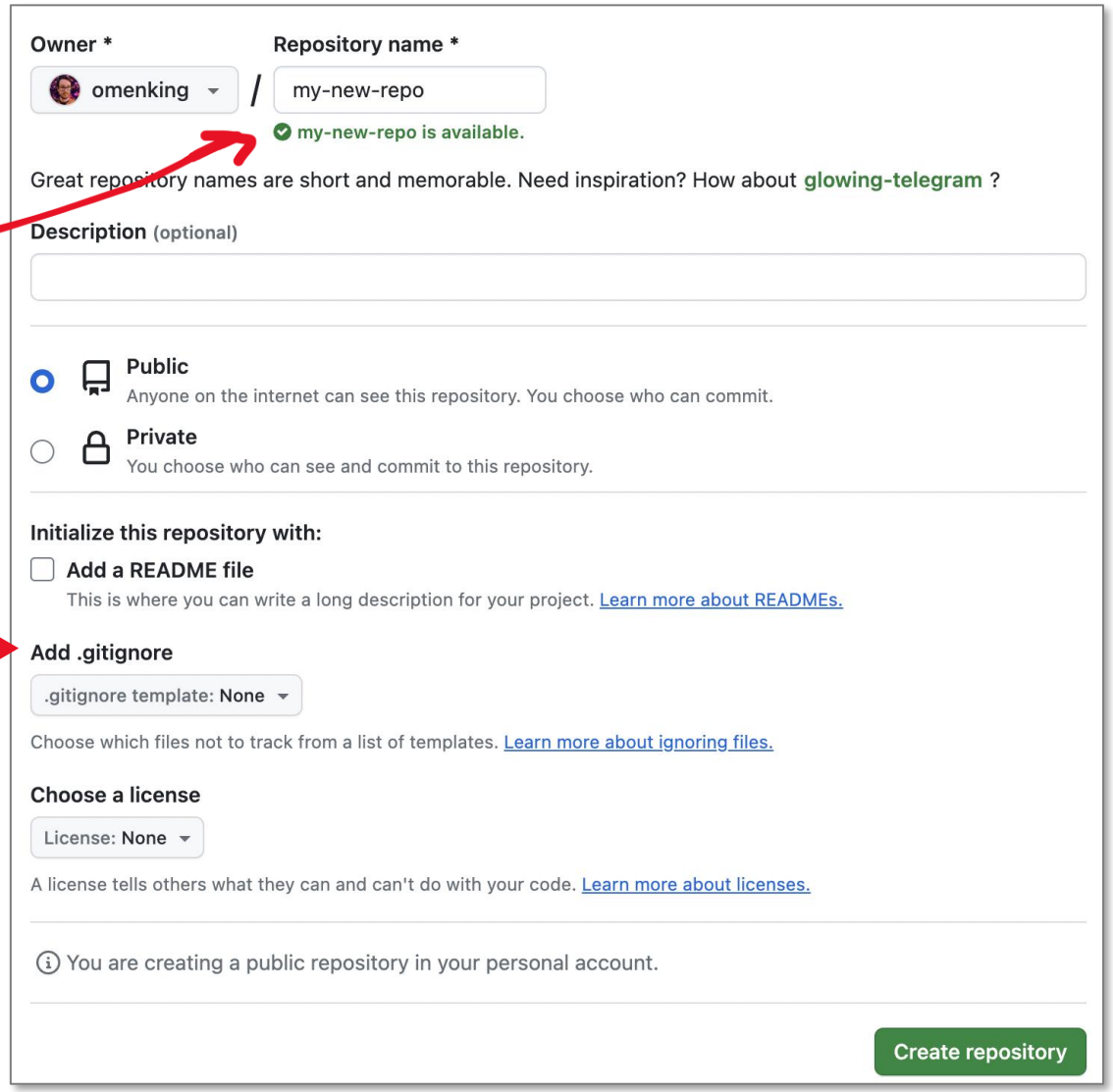
You need to choose an available github repo name

- Names are unique based on the scope of the owner

Your repo can either be public or private

You can quickly add a:

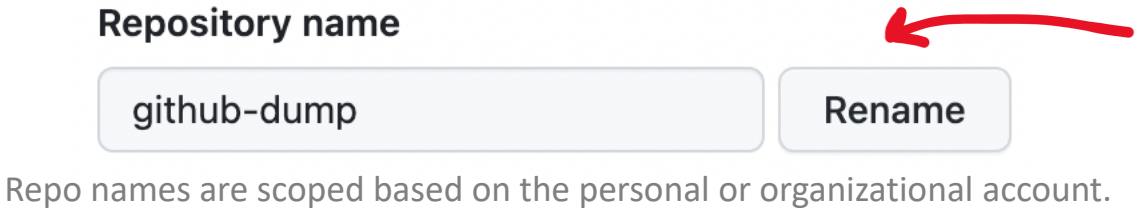
- Readme file
- .gitignore file
- License file



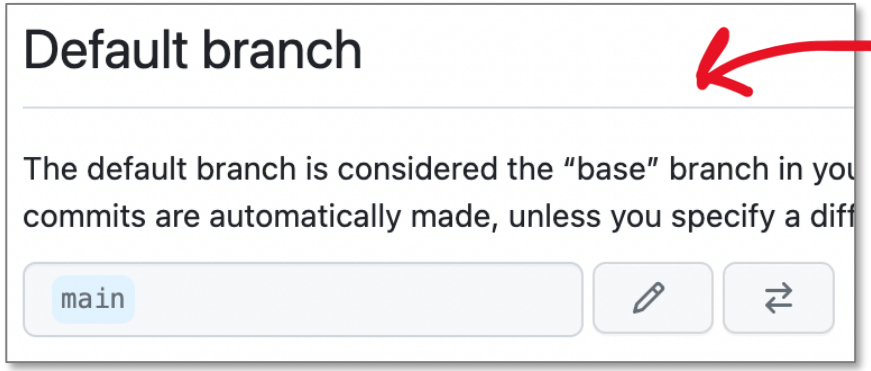
GitHub CLI can be used to create a GitHub repo

```
gh repo create my-project --public --clone
```

Maintaining a repo

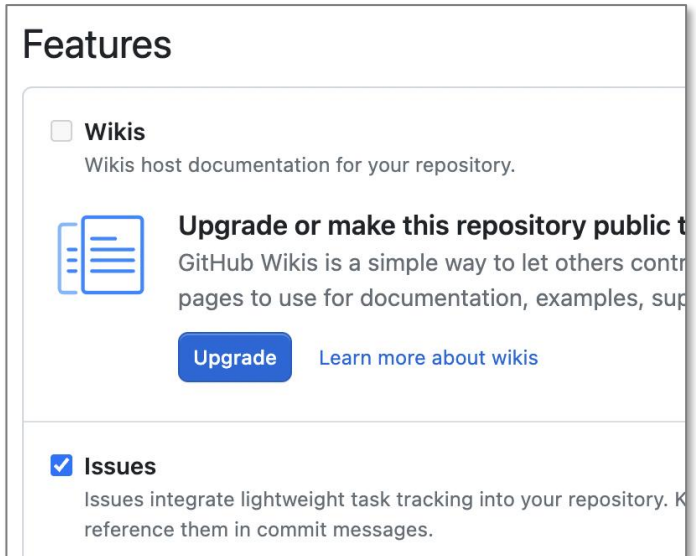


You can **change your repo name**, (but try not to because it can break external links and documentation pointing to a public repo)



You can **change the base branch** (default branch).
You can rename the default branch.
“main” is the unspoken best practice for naming your base branch

You can **opt-in-and-out of some features for your GitHub Repo**
Features will appear in the GitHub repo navigation bar



Maintaining a repo

The **Danger Zone** contains actions you need to think twice about because they cannot be undone.

Change repo visibility
When you make a private repo private to public:

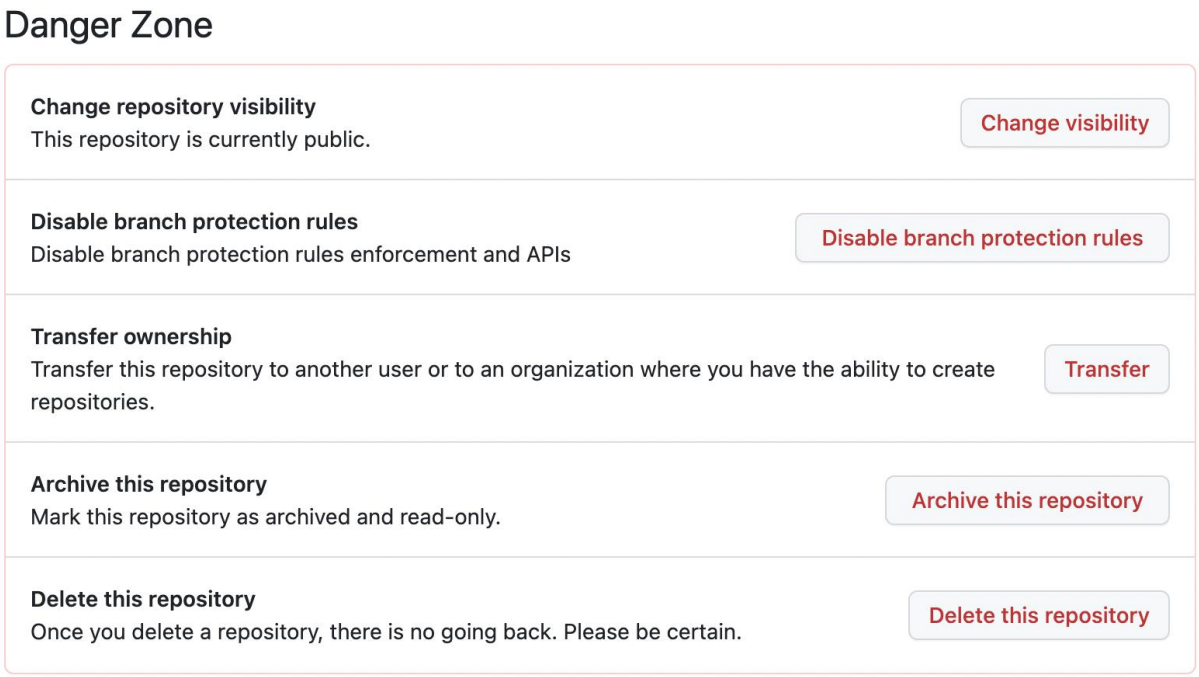
- The code will be visible to everyone who can visit https://github.com
- Anyone can fork your repository.
- All push rulesets will be disabled.
- Your changes will be published as activity.

Disable Branch Protection Rules
Branch protect rules are strict workflow rules like disallowing anyone pushing to main.

You can disable all protections temporarily eg. For quick fixes

Transfer ownership
When you want someone else to be the owner of the repo.

Archive this repository
When the repo to be read-only.

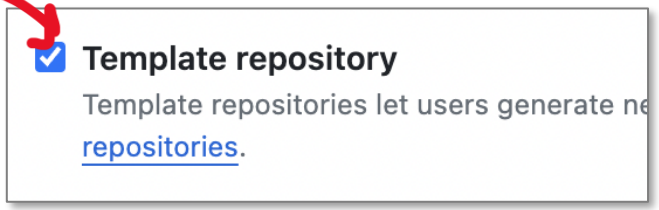


Delete this repository
When you want to delete your repo.
Once it's gone its gone, there is not trash bin you can pull it out of.

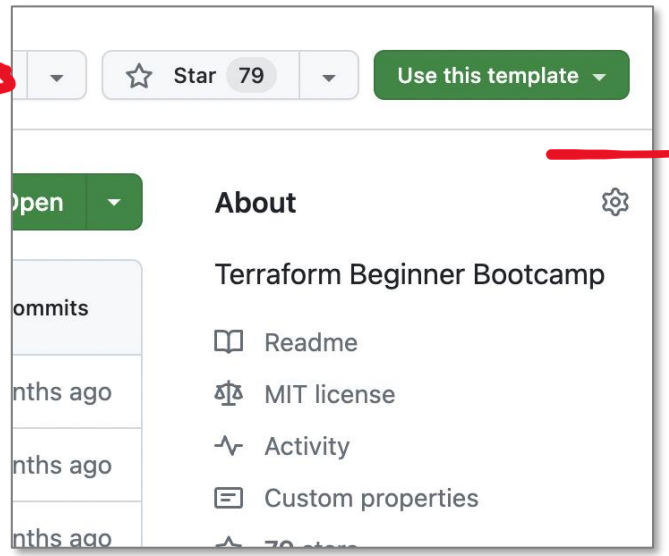
Repo Templates

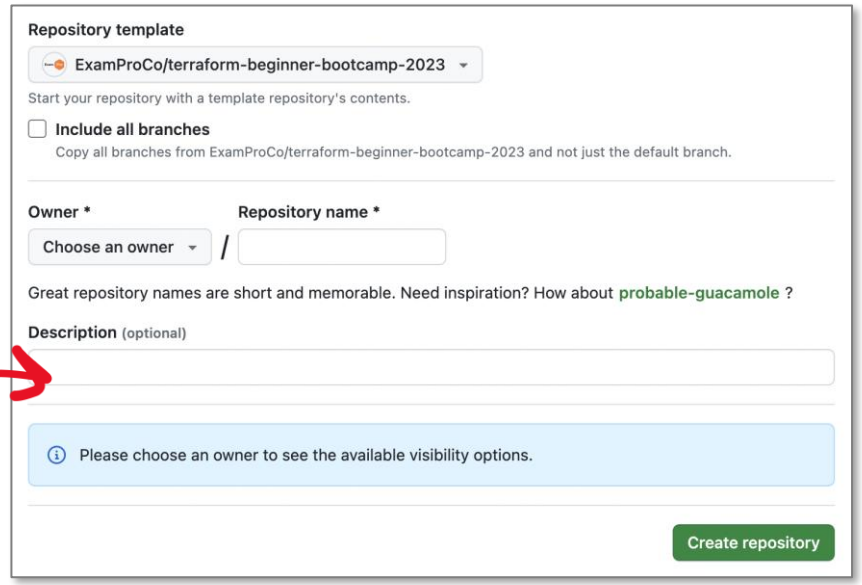
GitHub Repo Templates is a feature for public repos that **allow other GitHub users to make a copy of the contents of the template repo** to use as starting point of their own repo.

We set the repo to be a template



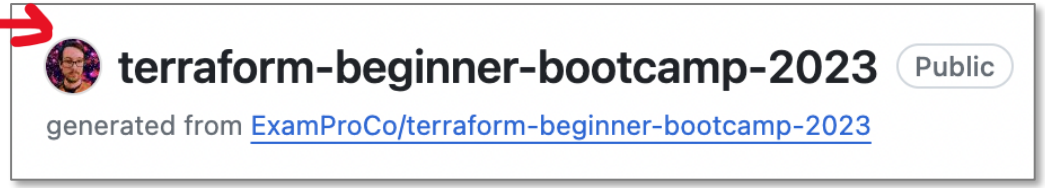
When users go the repo, they'll see a **Use this template** button to create a new repo from the template.



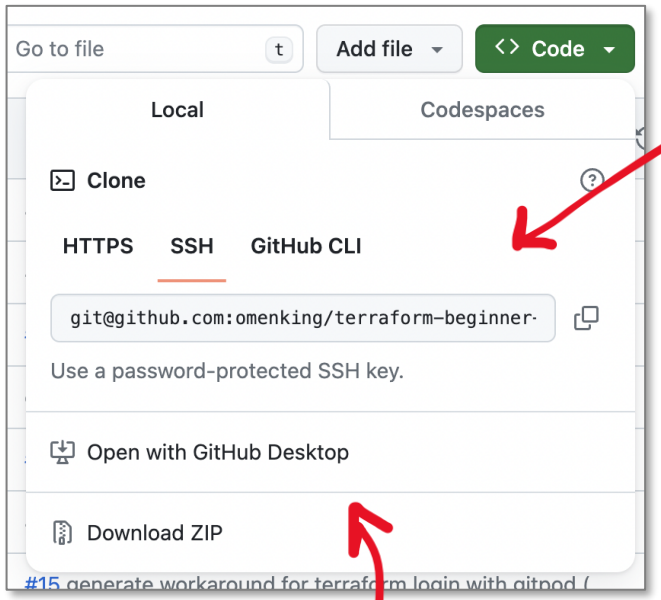


Templates are different from cloning because it's a clean repo with no previous history.

Their repo will indicate they **generated** their repo from the repo template



Cloning a repo



You can also:

- Open in GitHub Desktop which will perform the clone for you
- Download a ZIP containing the contents of the repo

You can **clone a repo *programmatically*** three different ways:

1. HTTPS

```
git clone https://github.com/omenking/terraform-beginner-bootcamp-2023.git
```

You supply GitHub username and password on clone, you need to set git to cache credentials

2. SSH

```
git clone git@github.com:omenking/terraform-beginner-bootcamp-2023.git
```

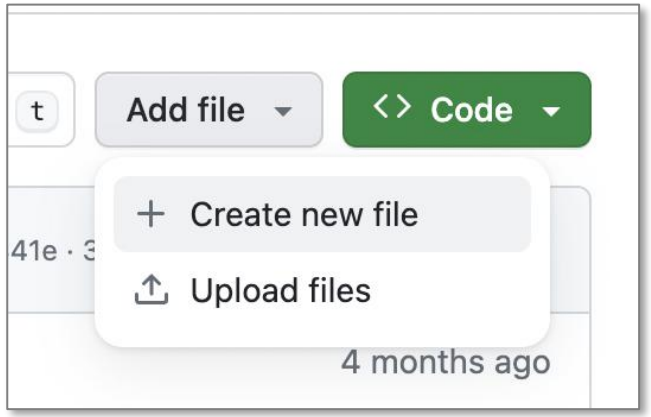
Authorization is based on SSH keypair
you have to upload the Public key to your GitHub account

3. GitHub CLI

```
gh repo clone omenking/terraform-beginner-bootcamp-2023
```

Authorization is based on the credentials when did a gh login

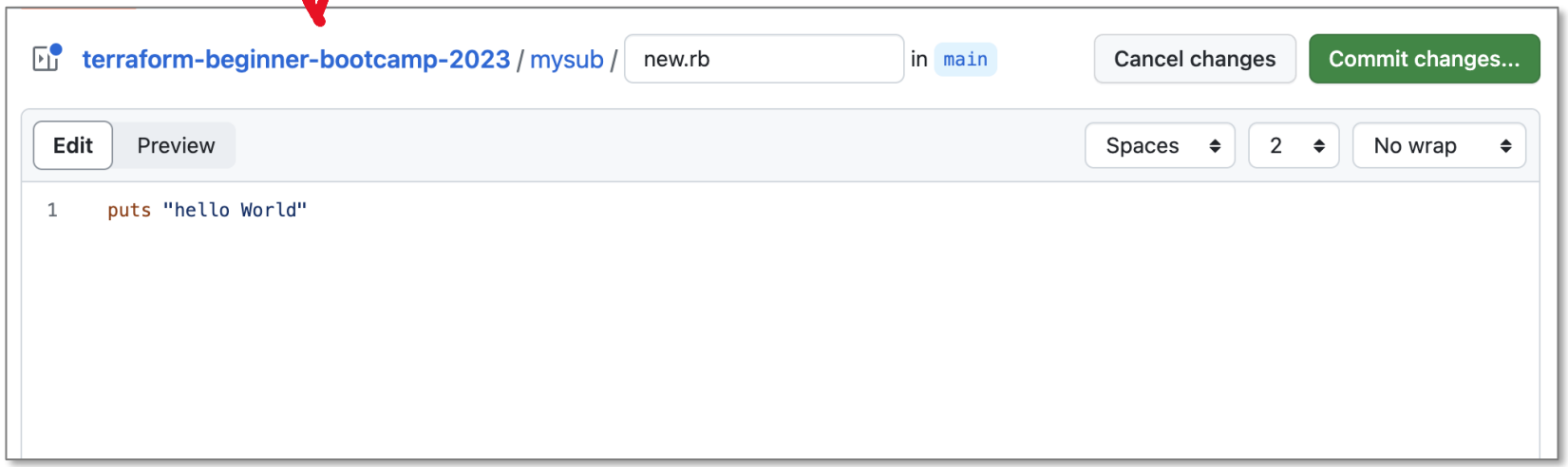
Adding files to a repo



Within the GitHub UI:

- Multiple Text and Binary files can be uploaded.
- New single text file can be **created**

When creating files in place, you have a basic text editor.
You can also **create nested folders by typing a forward slash in the filename**.



When adding multiple files that you also need to edit, you can also quickly use **Github.dev** at no cost

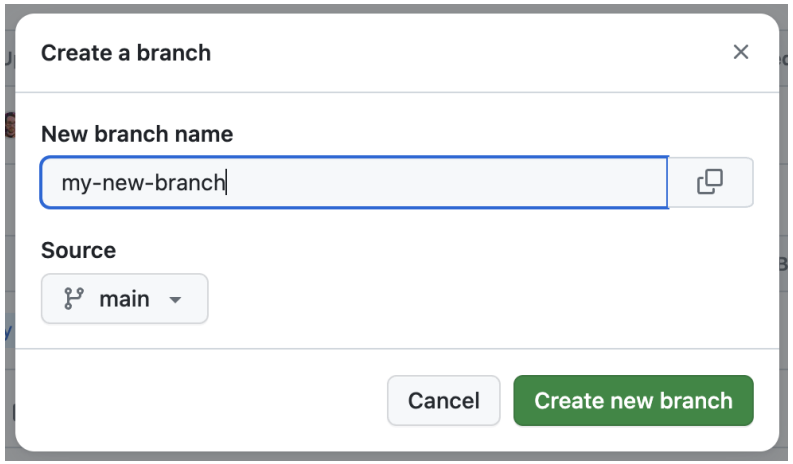
Creating branches

Using git we can create a new branch from an existing branch.
We then want to push upstream our branch to GitHub

```
git checkout -b staging
# change files, create commits
git push -u origin staging
```

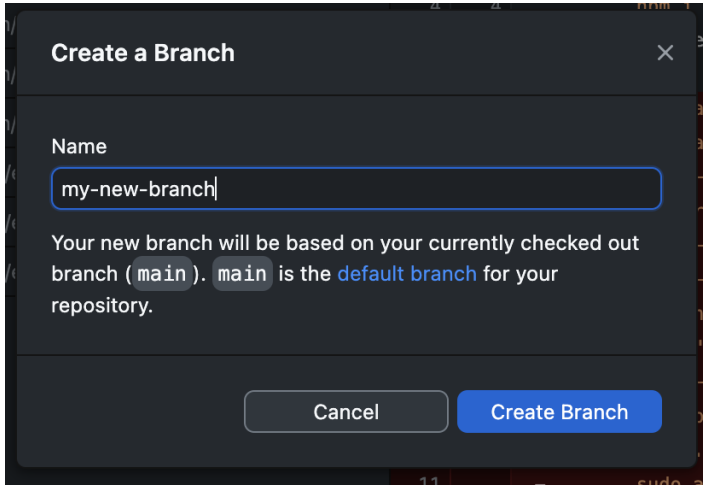
-u is the short hand flag for –set-upstream.

Branches can be directly created within GitHub UI.

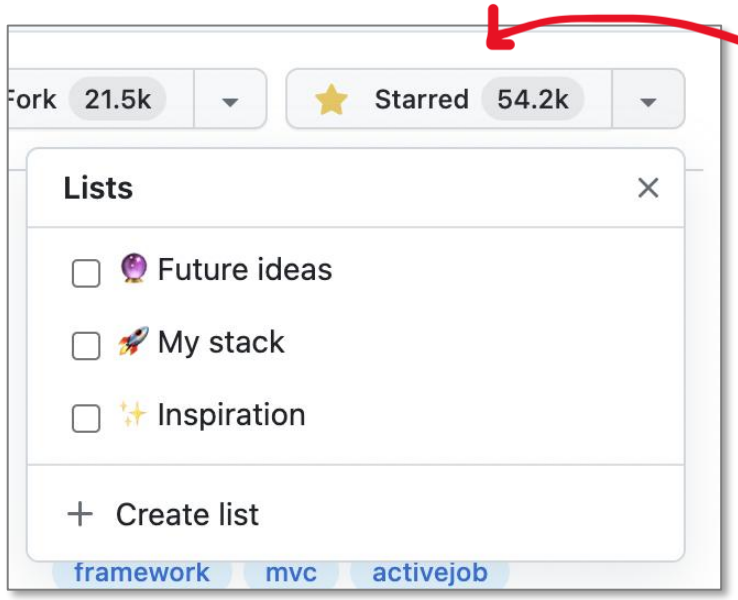


Branches can be created from Issues.
The branch and issues will be associated/link

Branches can be created in GitHub Desktop



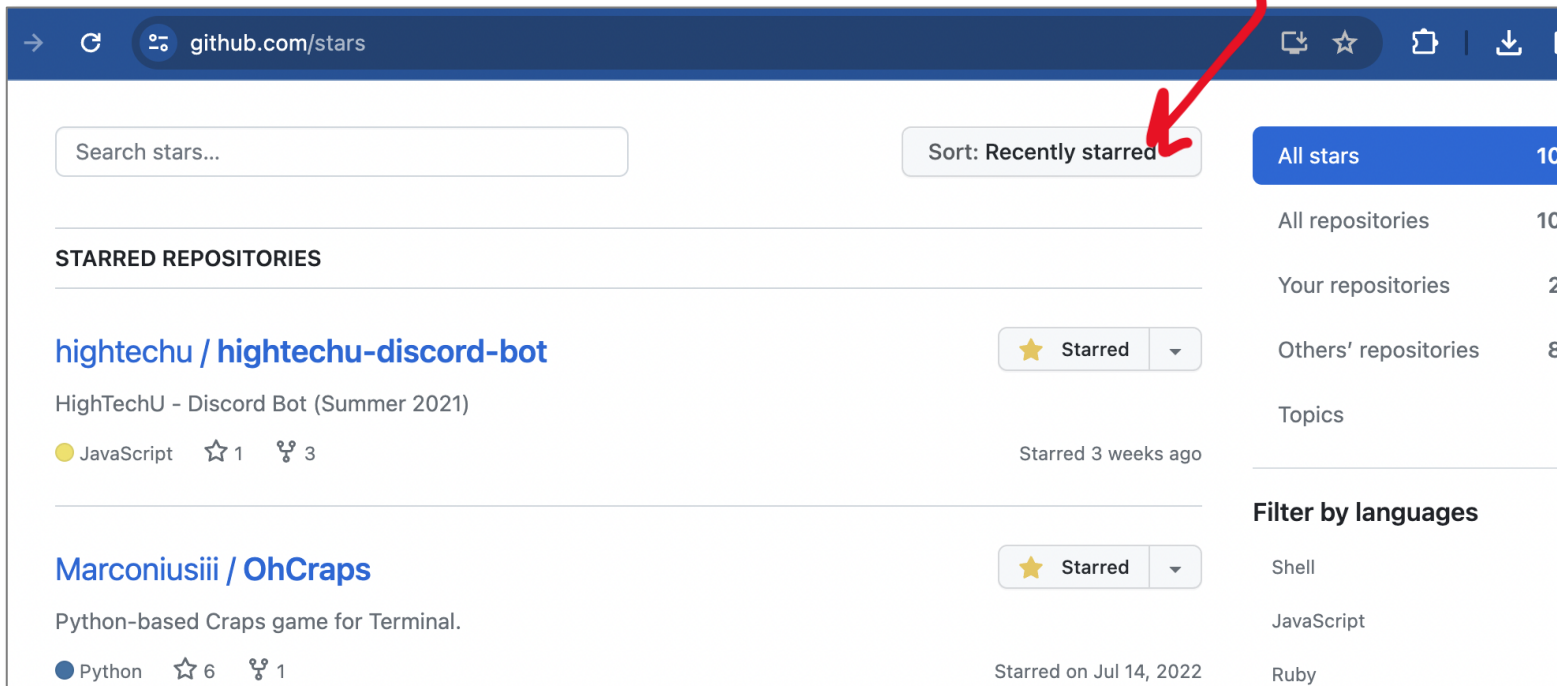
Starring repos



Similar to bookmarking, it's a way to mark a repository as interesting or to keep track of it for reference.

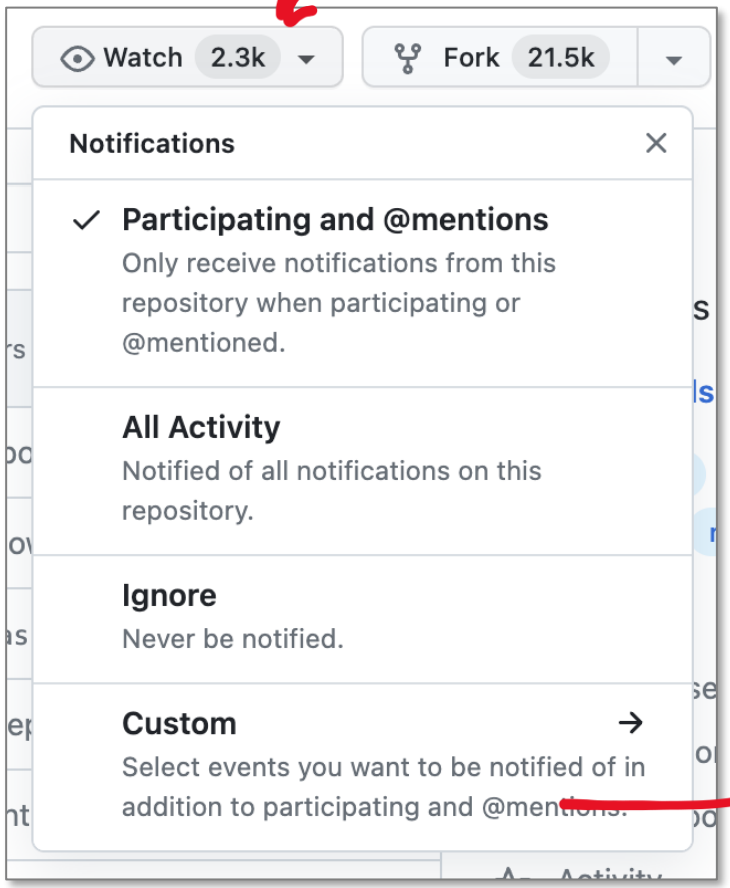
Stars are public and can be seen by everyone, often used as a **measure of a project's popularity**.

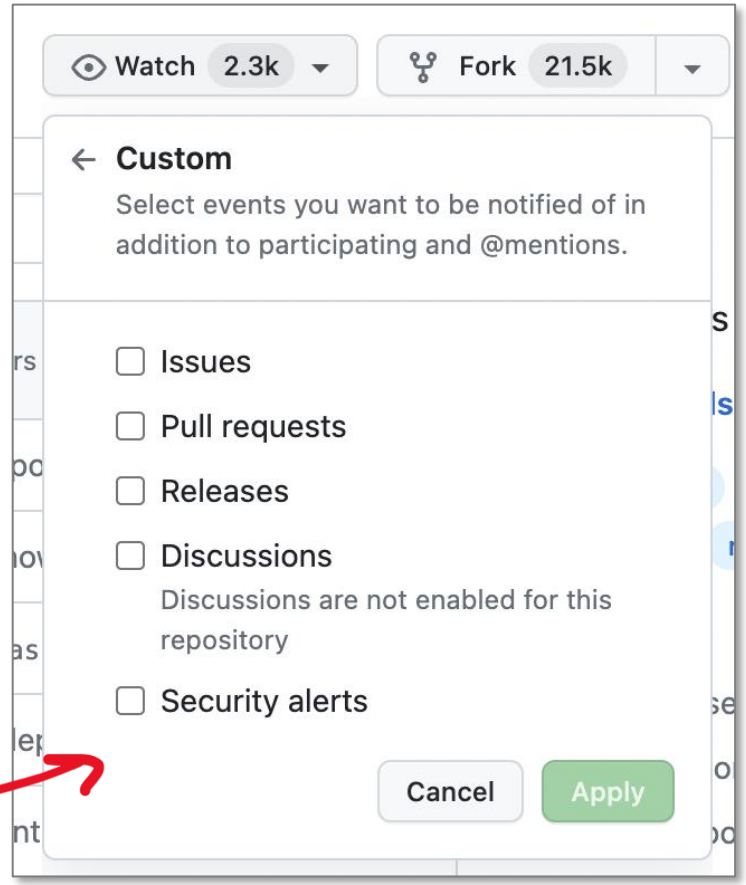
You easily find pages you starred at: https://github.com/stars



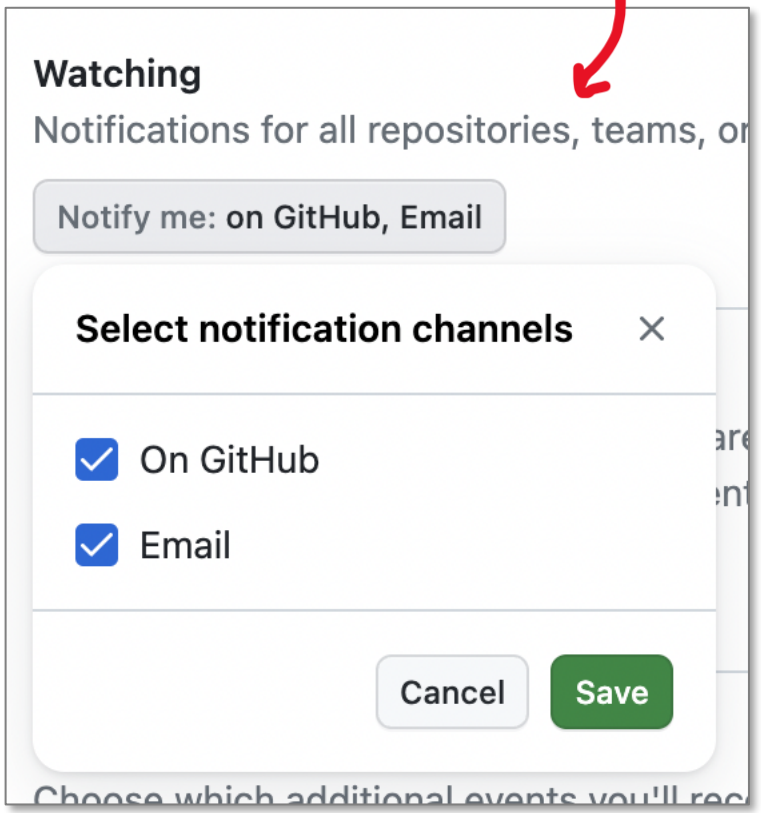
Watching Repos

Watching a repo allows you to **stay informed about activities occurring within a repo**.



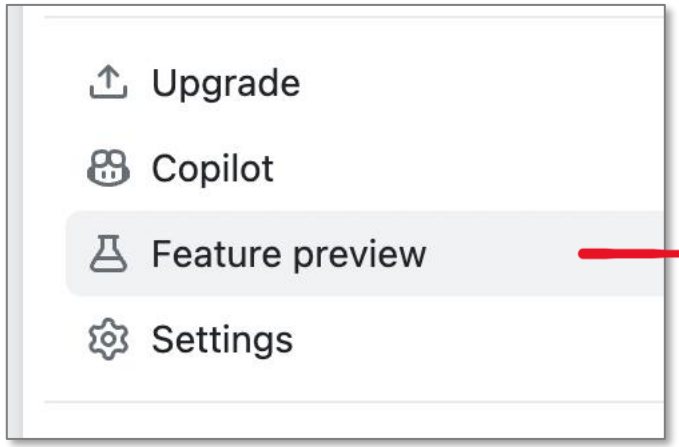


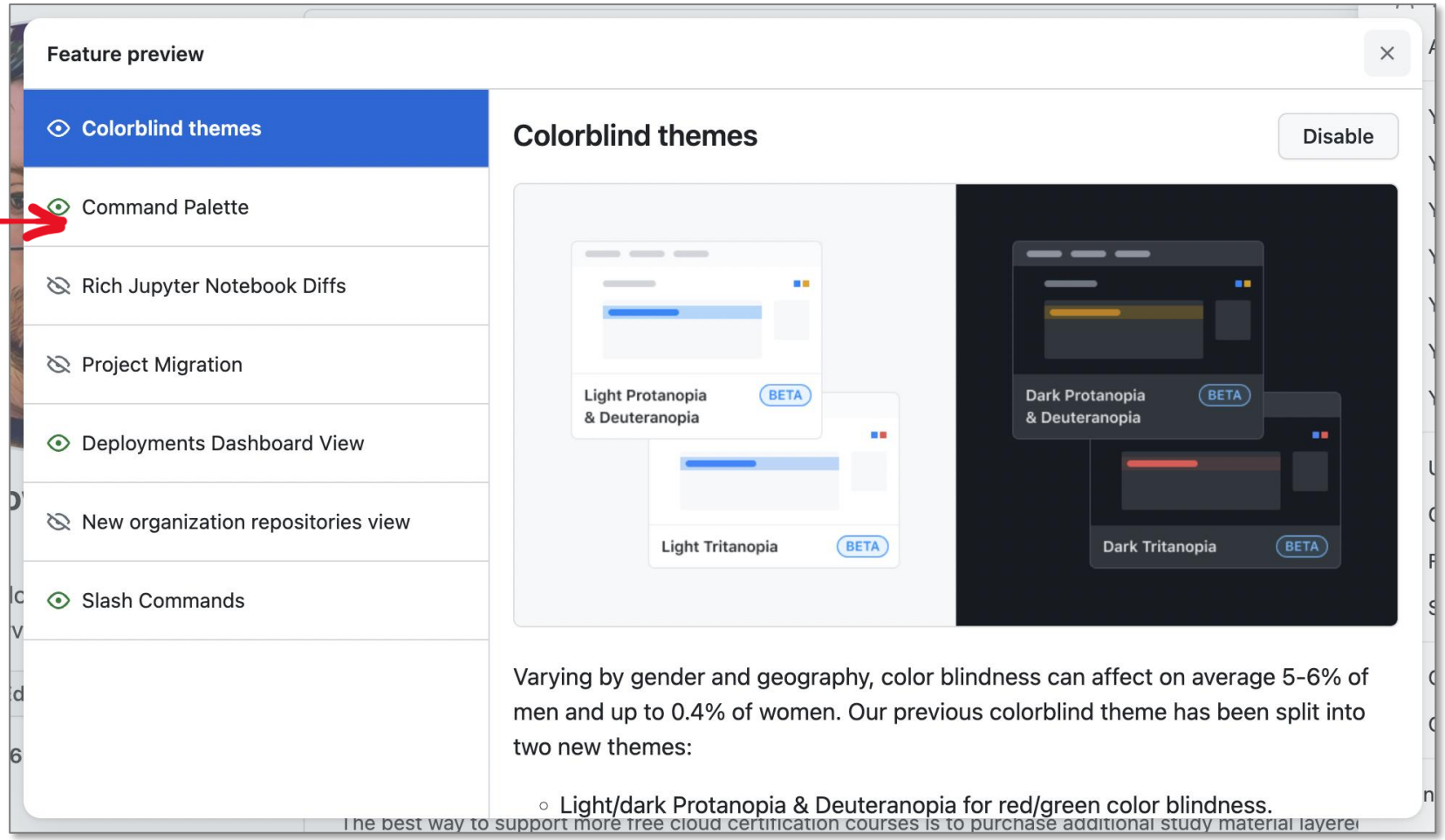
In your GitHub Account, you can specify **how you want to be notified**



Feature previews

Feature previews allows you to **enable or disable features that are in beta** in your personals account

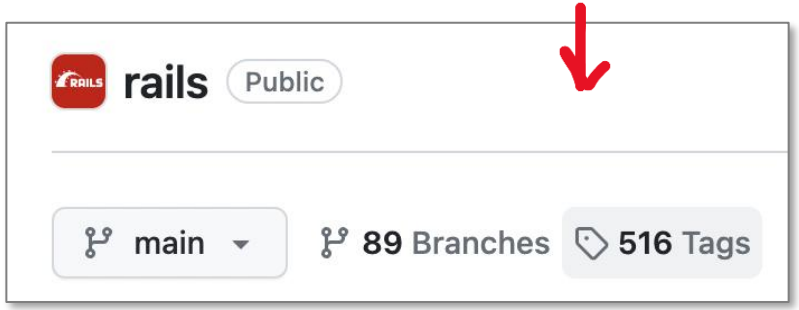


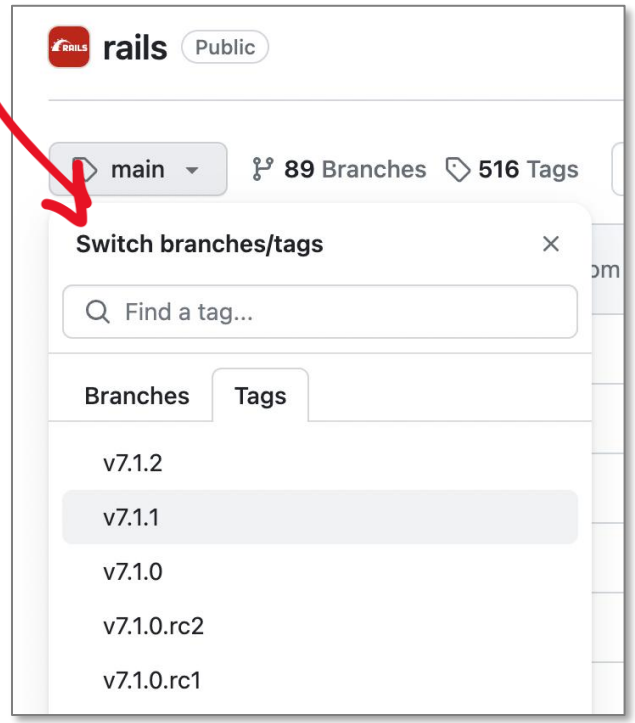


Tags

Tagging is used to capture a point in history to marked version release of your codebase

GitHub makes it **easy to explore tagged versions** of Git repo.





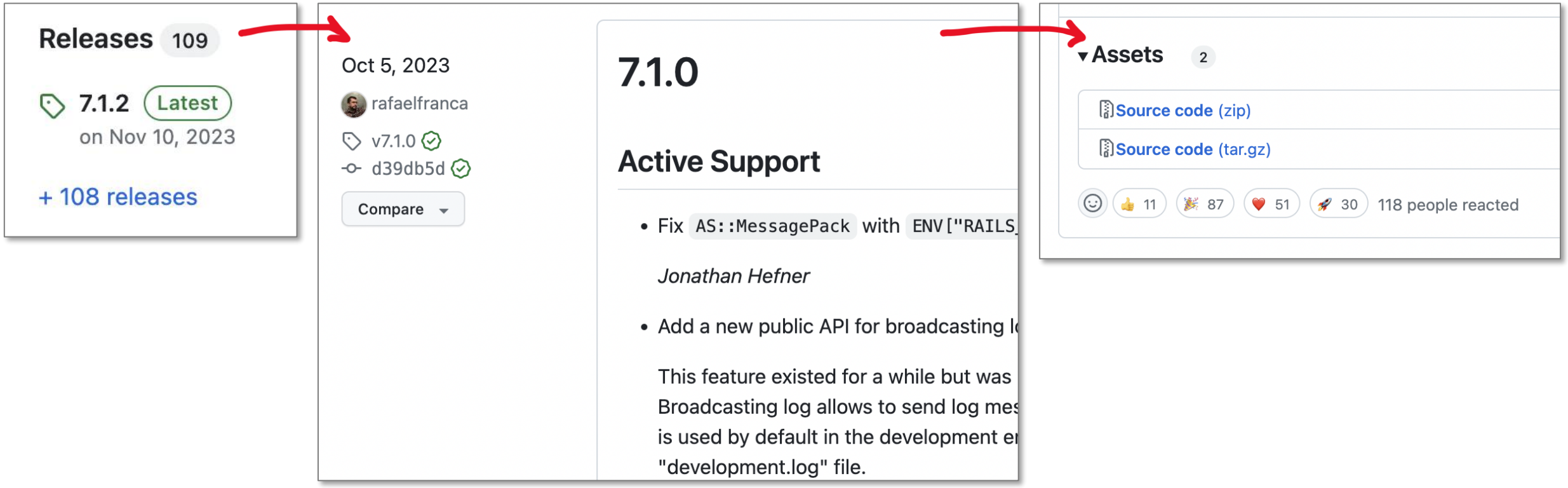
Common git tag commands:

```
git tag v1.0.0 # tag a commit
git push --tags # push tag to remote
git checkout v1.0.0 # checkout codebase at tag
git tag -d v1.0.0 # delete local tag
git push --delete origin tagname # delete remote tag
```

CI/CD pipelines may be configured to deploy on the presence of a new tag on production branch

GitHub Releases

GitHub Releases allows you to **create releases with release notes and linked assets** such as zip source or binaries for specific platforms



GitHub Packages

GitHub Packages is a platform for **hosting and managing packages, including containers** and other dependencies.

Pushing Docker container to **GitHub Packages**

```
GH_USERNAME="andrew-wc-brown"
IMAGE_NAME="hello-world"
VER="1.0.0"

# Sign-in to docker
echo $GH_TOKEN | docker login ghcr.io -u $GH_USERNAME --password-stdin
# Tag docker container
docker tag $IMAGE_NAME:$VER ghcr.io/$GH_USERNAME/$IMAGE_NAME:$VER
# Push to Docker Packages
docker push ghcr.io/$GH_USERNAME/$IMAGE_NAME:$VER
```

Supported package registry:

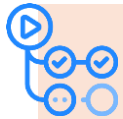
- JavaScript (npm package)
- Ruby (gems)
- Java (Maven and Gradle packages)
- .NET (NuGet packages)
- Docker images

```
FROM alpine:latest
CMD echo "Hello World!"
```

Dockerfile

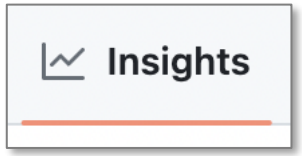
```
docker build -t hello-world .
docker run hello-world
```

Build and run Docker image



GitHub Actions could be used to build and then publish packages to GitHub Packages

Repo Insights



For a GitHub Repo under the Insights tab, you can gain lots of statistical graphs about the repo.

Insights contains:

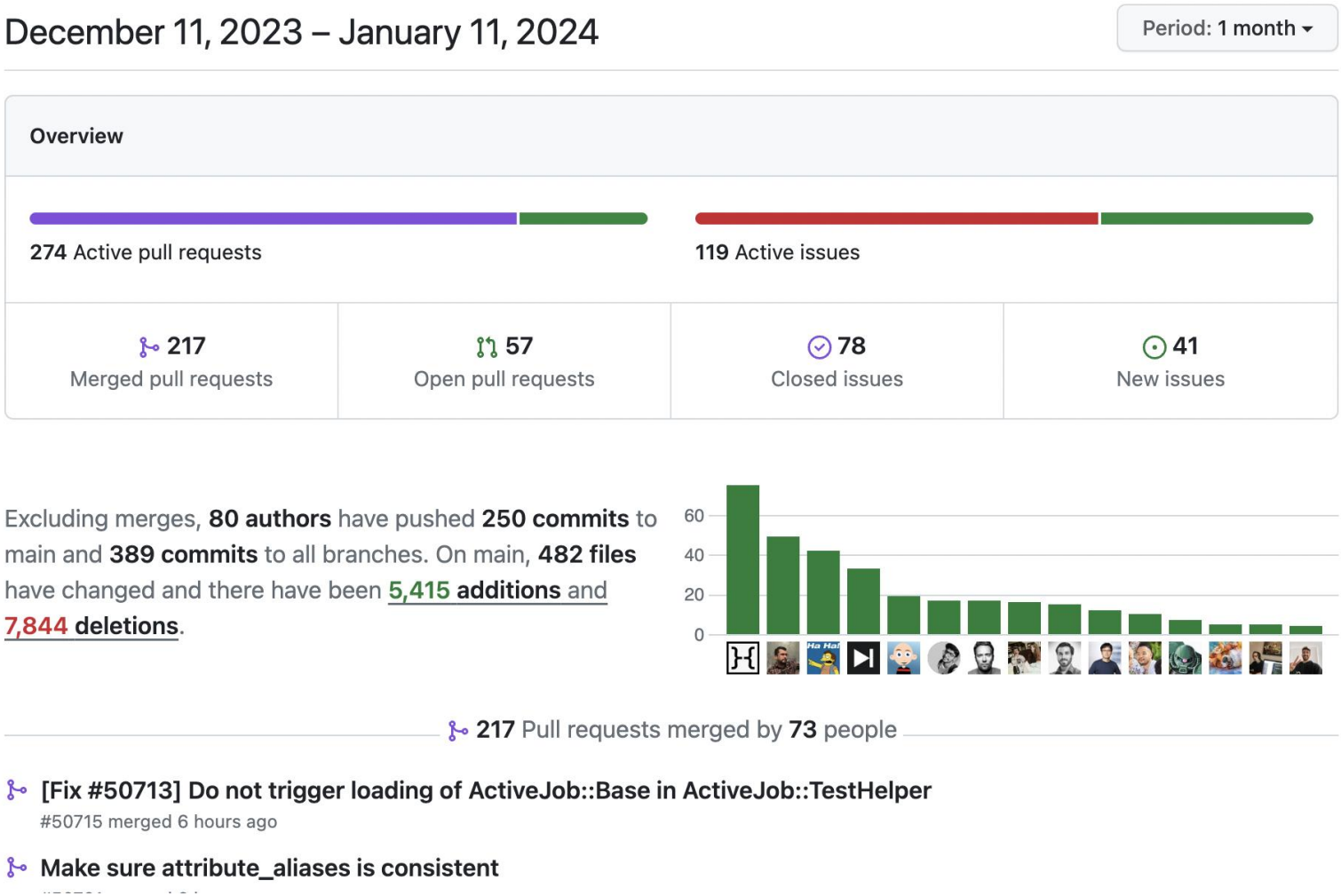
- **Pulse:** Overview of recent activity (issues, pull requests).
- **Contributors:** Lists contributors and their activity stats.
- **Community Standards:** Checks for essential community health files.
- **Commits:** History of all commits in the repo.
- **Code Frequency:** Graph of code additions and deletions over time.
- **Dependency Graph:** Visualizes code dependencies.
- **Network:** Shows fork relationships and variations.
- **Forks:** Number and links to repository forks.

Repository insights is not listed as an available feature in the GitHub free tier?

Repo Insights

Pulse contains:

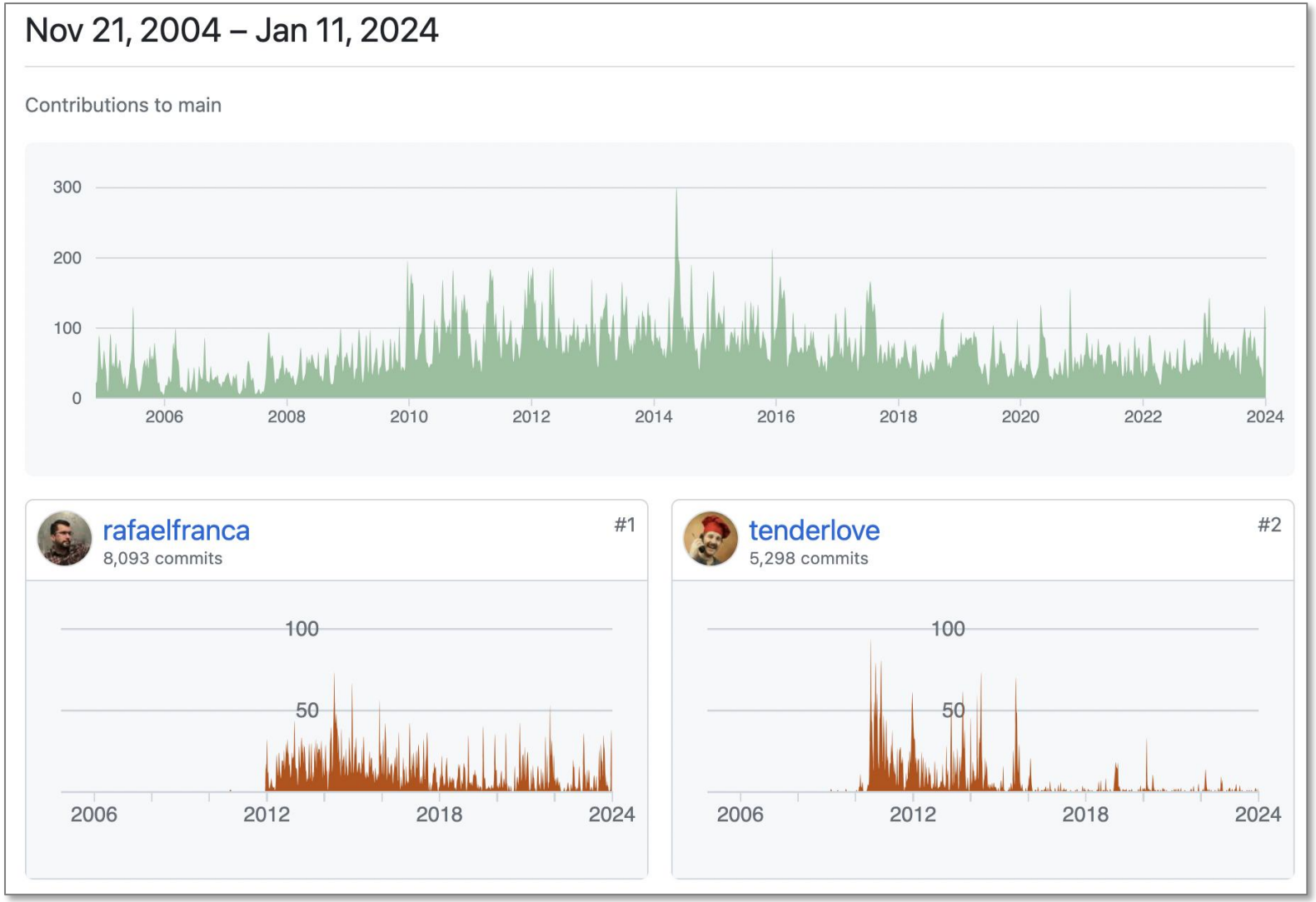
- Pull requests open/close ratio
- Issues open/close ratio
- Summary of activity
- Graph of top contributors
- List of merged pull requests
- List of issues closed
- List of issues opened
- List of unresolved conversations



Repo Insights

Contributors contains:

- Graph of all commits over a time period
- Graph of specific contributors commits over a time period



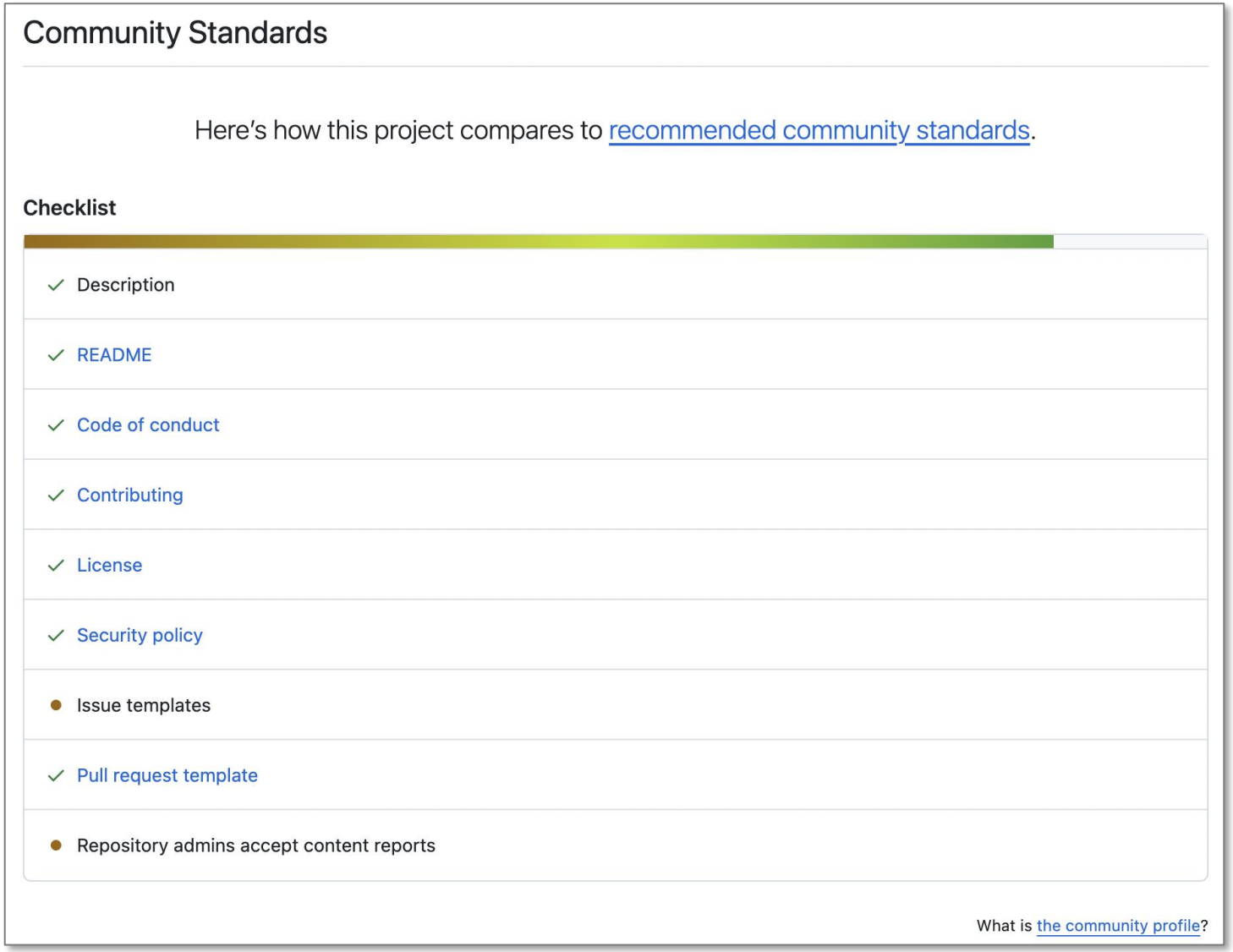
Repo Insights

Community Standards contains:

- A checklist of recommend community standards and how much a community profile has completed

This graph will only show for repos that have community profiles.

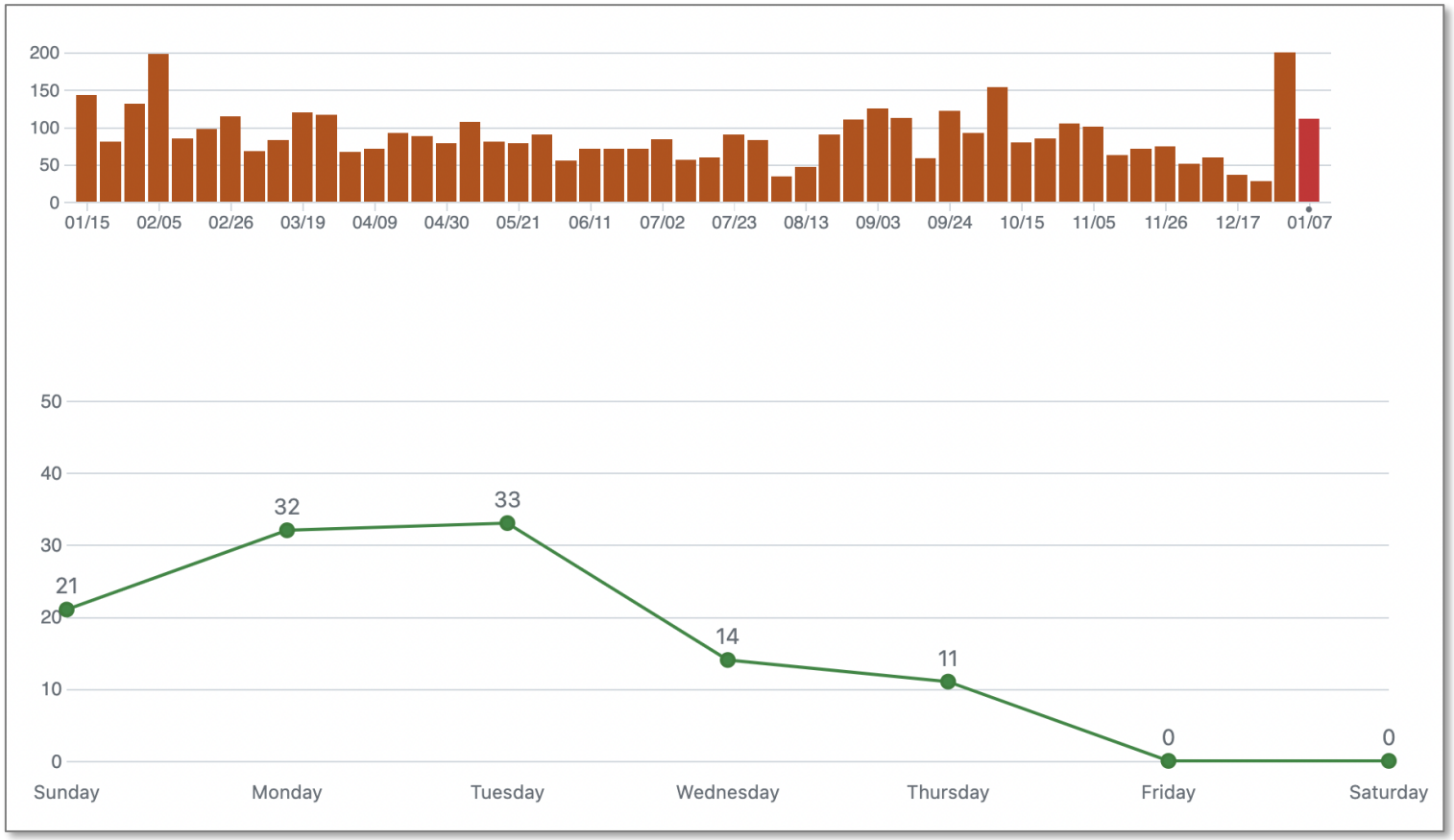
Community profiles are for open-source projects on GitHub.



Repo Insights

Commits contains:

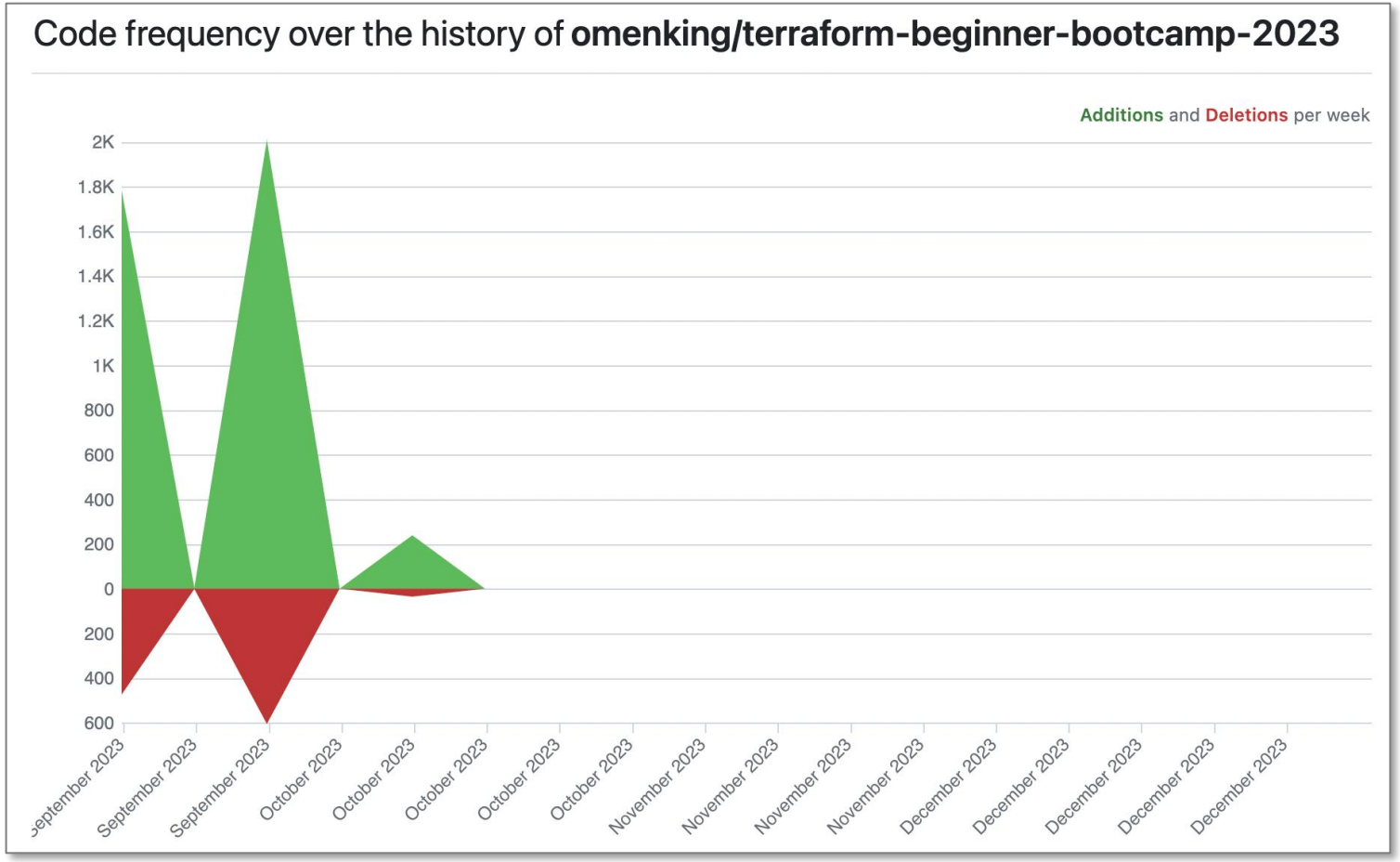
- Number of commits per week for the last 52 weeks. (for a year)
- The average number of commits by day of the week for the selected week



Repo Insights

Code frequency contains:

- The number of additions and deletions of code to a report per month.

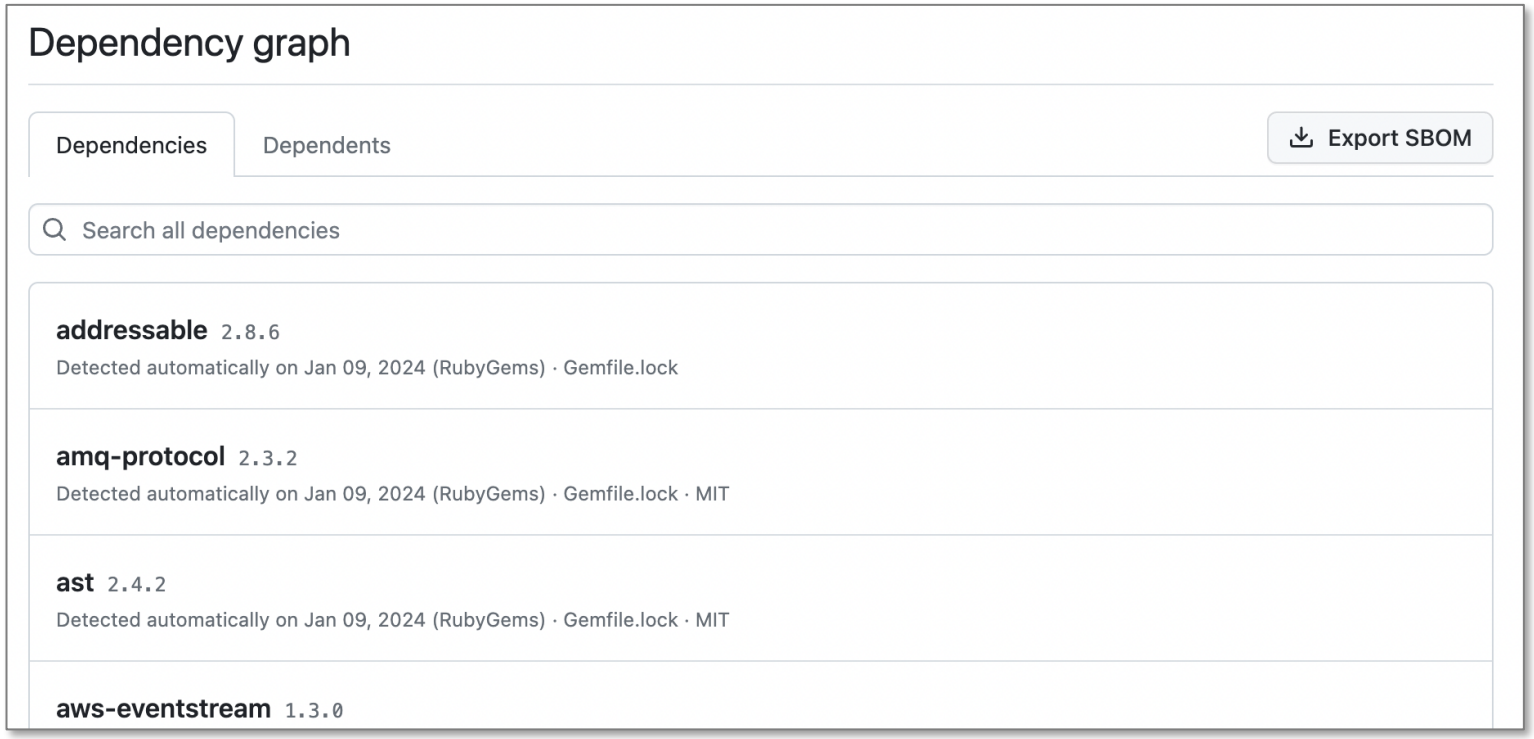


If the additions and deletions is too large the information will be able to graph.

Repo Insights

Dependency graph contains:

- List of dependencies
- List of dependents
- Export Software Bills of Materials (SBOMs)

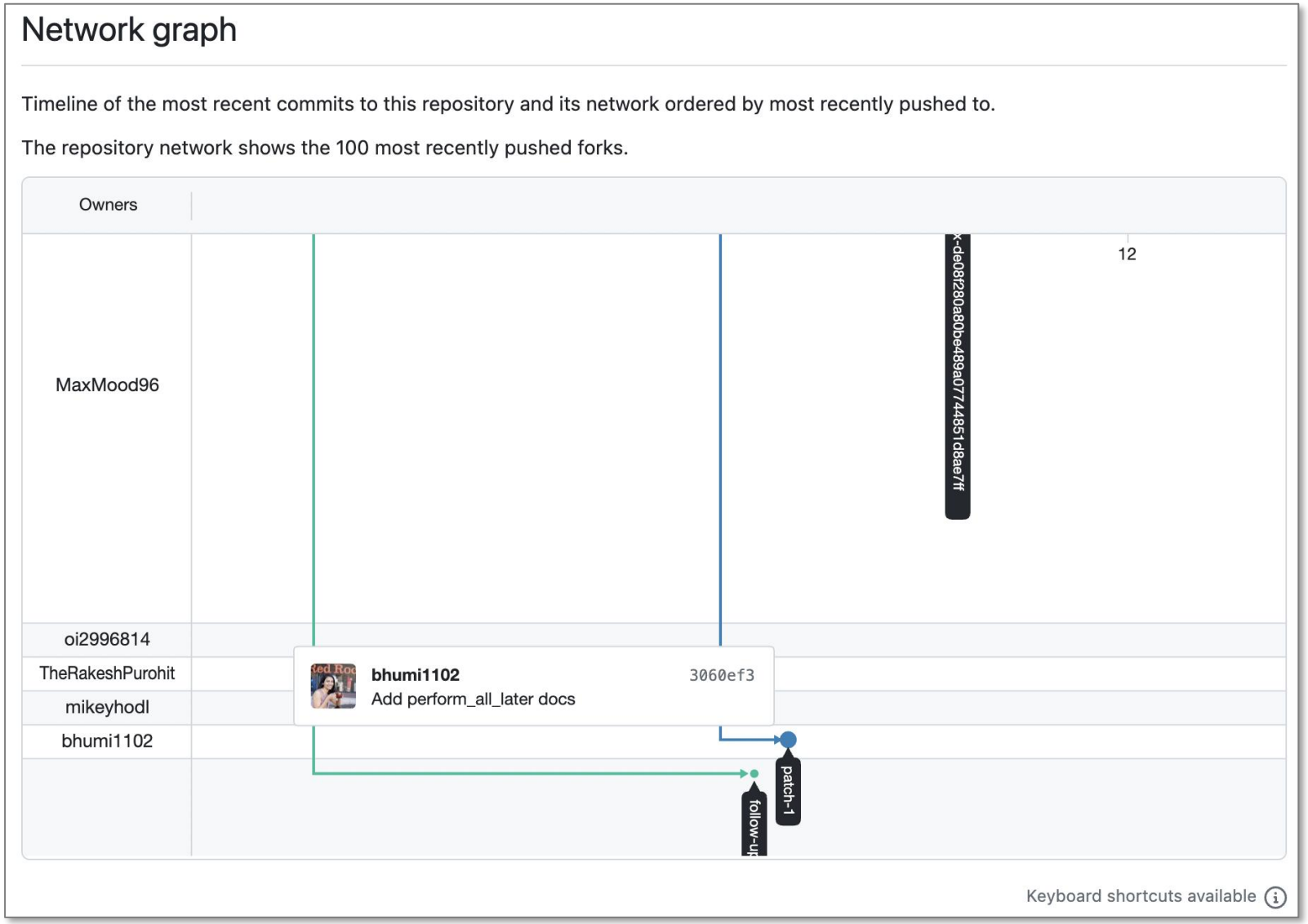


Security and compliance teams increasingly request software bills of materials (SBOMs) to identify the open-source components of their software projects, assess their vulnerability to emerging threats, and verify alignment with license policies.

Repo Insights

Network contains:

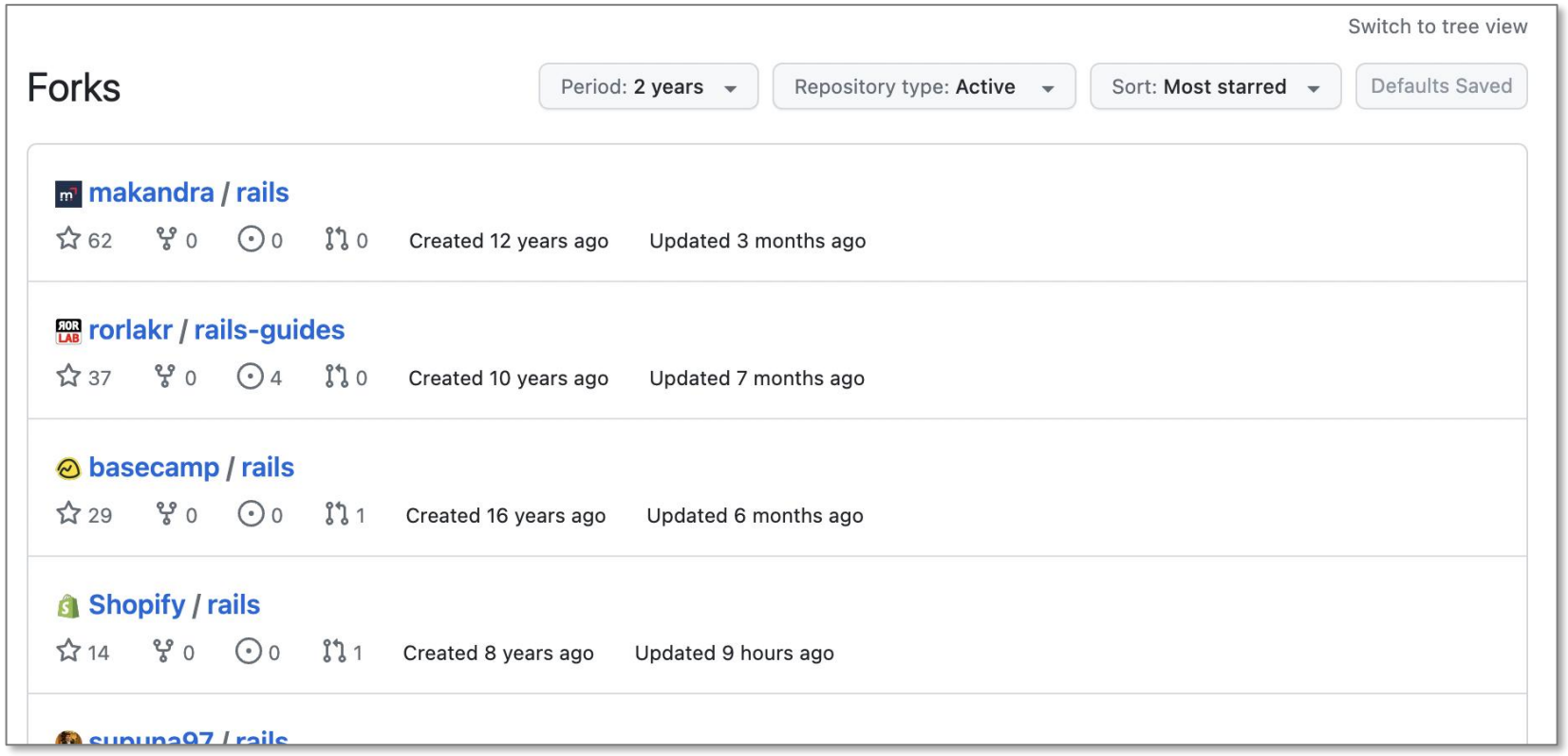
- 100 most recently pushed forks
  - You can read the commits to determine the difference of these forks.



Repo Insights

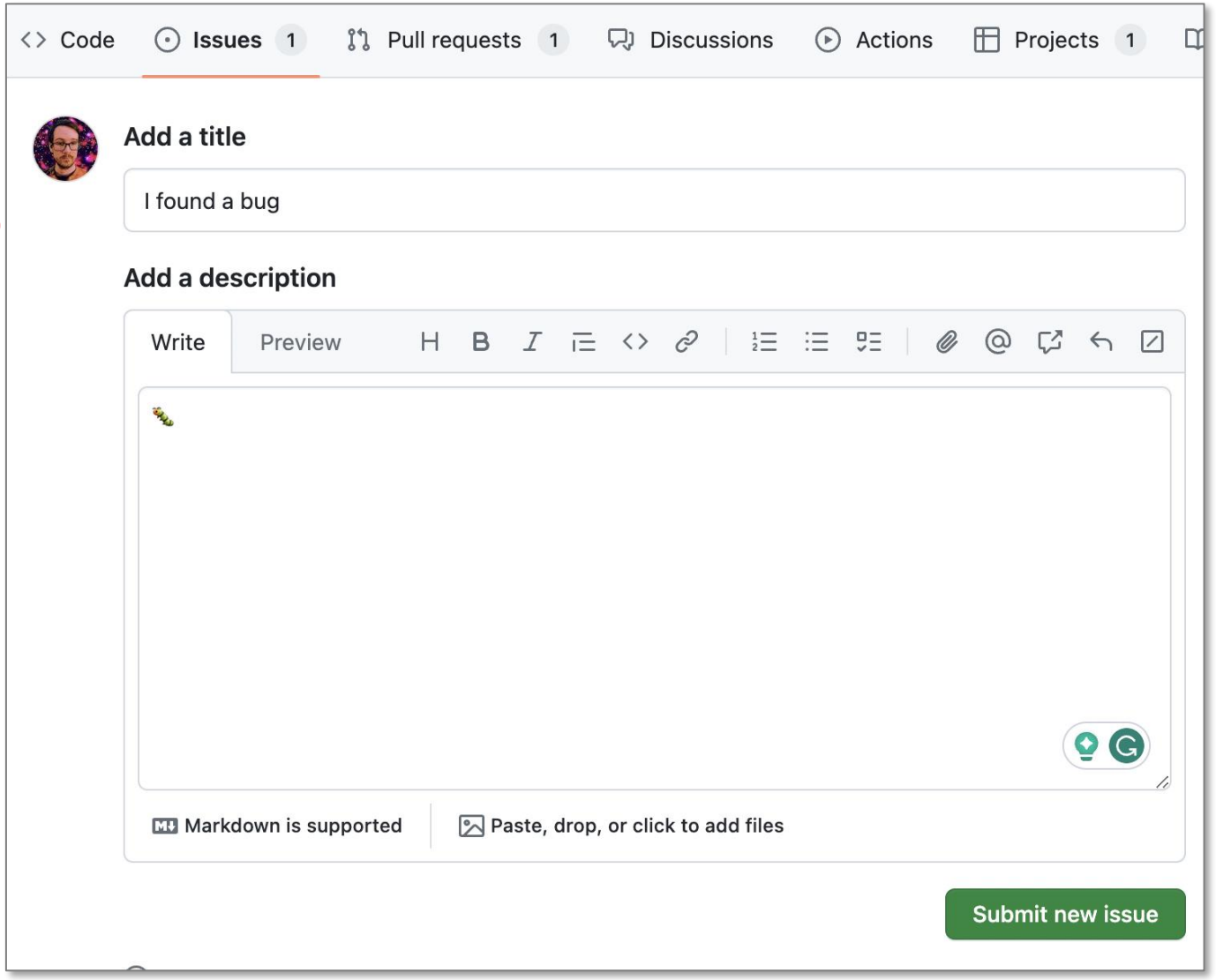
Forks contains:

- Contains a list of filterable forks



Creating Issues

Opening an Issue is straight forward.
You enter a **title** and a markdown-supported **description**



You can use the **GitHub CLI** to create Issues.

```
# Create an issue interactively
$ gh issue create
Creating issue in owner/repo
? Title My new issue
? Body [(e) to launch nano, enter to skip]
http://github.com/owner/repo/issues/1
$
```

Issues vs Discussions vs Pull Requests

Issues

Tracking tasks, bugs, enhancements, and other actionable items.

- Often linked to code changes
- Can be linked to **PRs**

Discussions

Facilitating conversations and Q&As about a wide range of topics related to the project.

- Categorized by topics
- Can be “converted” to issue
  - Really just creates an issue with a soft link to the issue
- **Not directly linked to code changes**

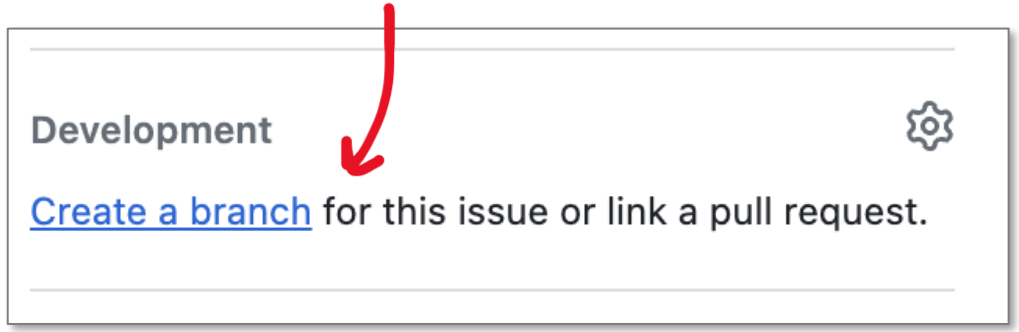
Pull Requests

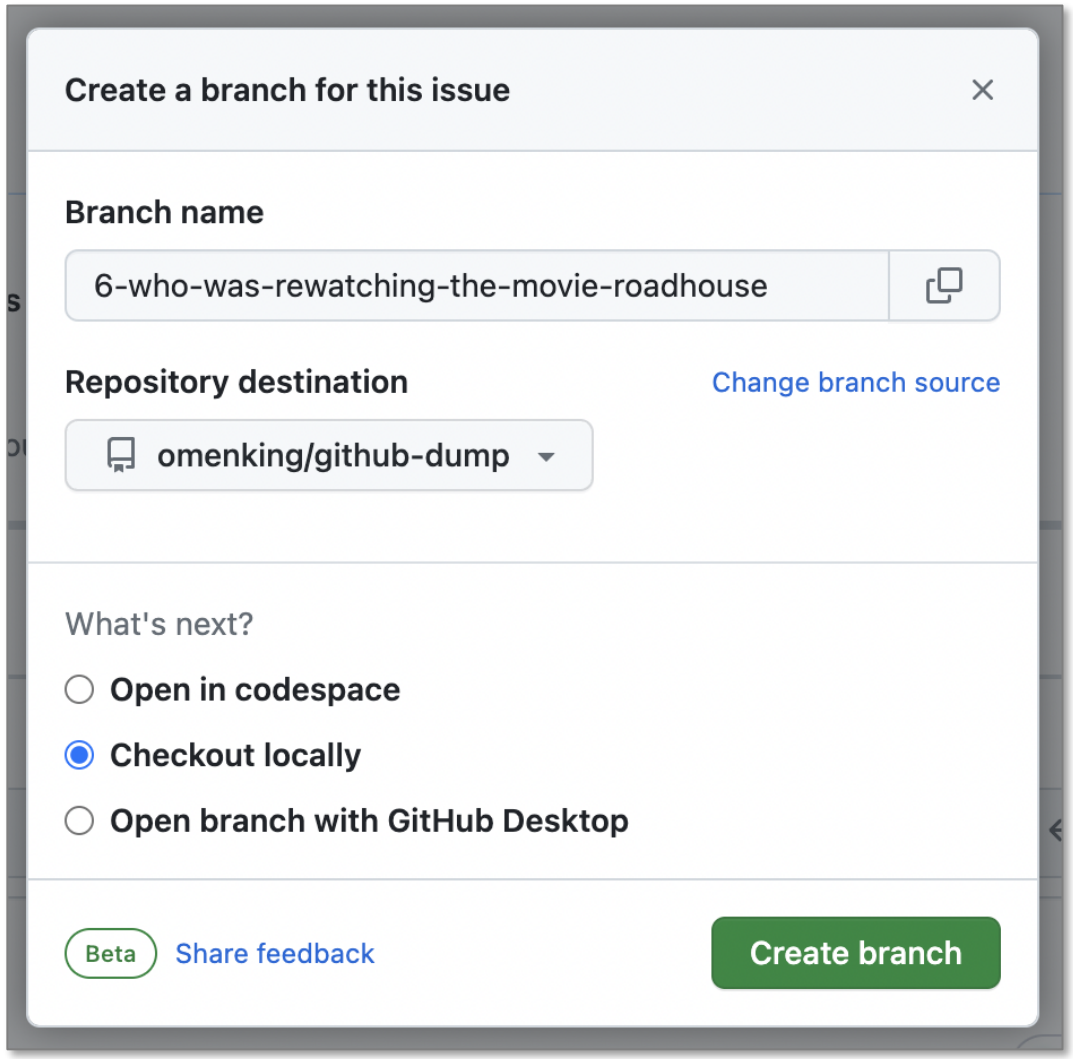
Proposing, reviewing, and merging code changes into the codebase.

- Directly linked to code changes
- Can be linked to **Issues**

Create a Branch from an Issue

From an Issue you can create
an **associated branch**.





A common workflow is creating Feature or Bug branches.
By defining your Issues upfront, you quickly create these branches

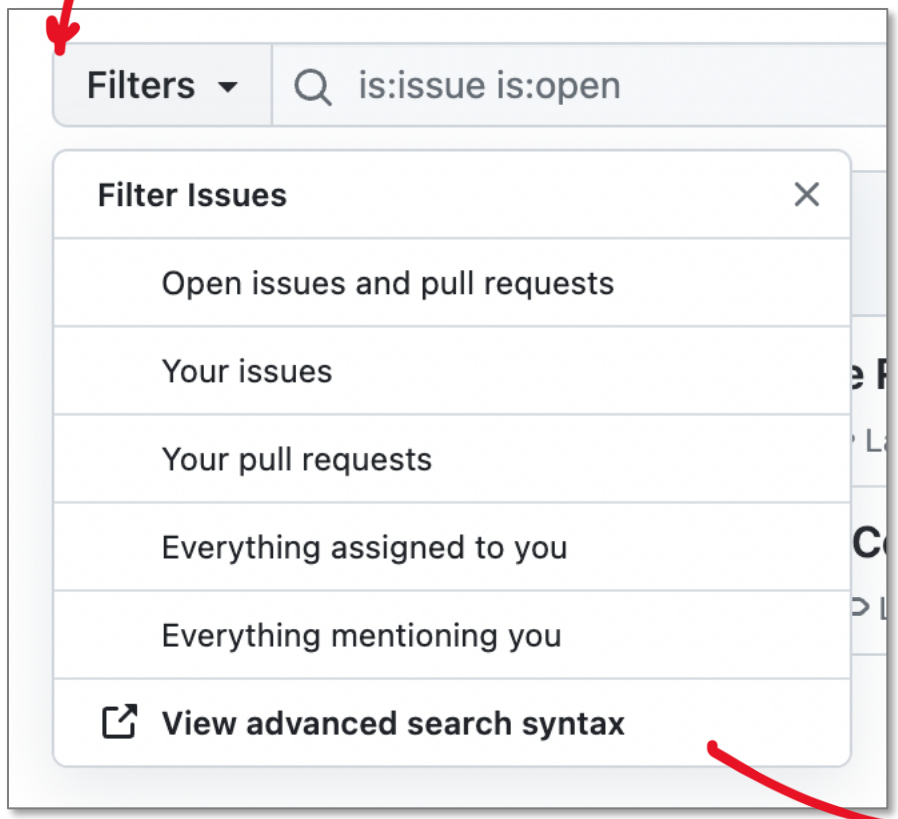
Define Feature or Bug as Issue

↓

Create Feature/Bug Branch

Search and Filter Issues

You can apply filter rules to quickly sort though lots of Issues

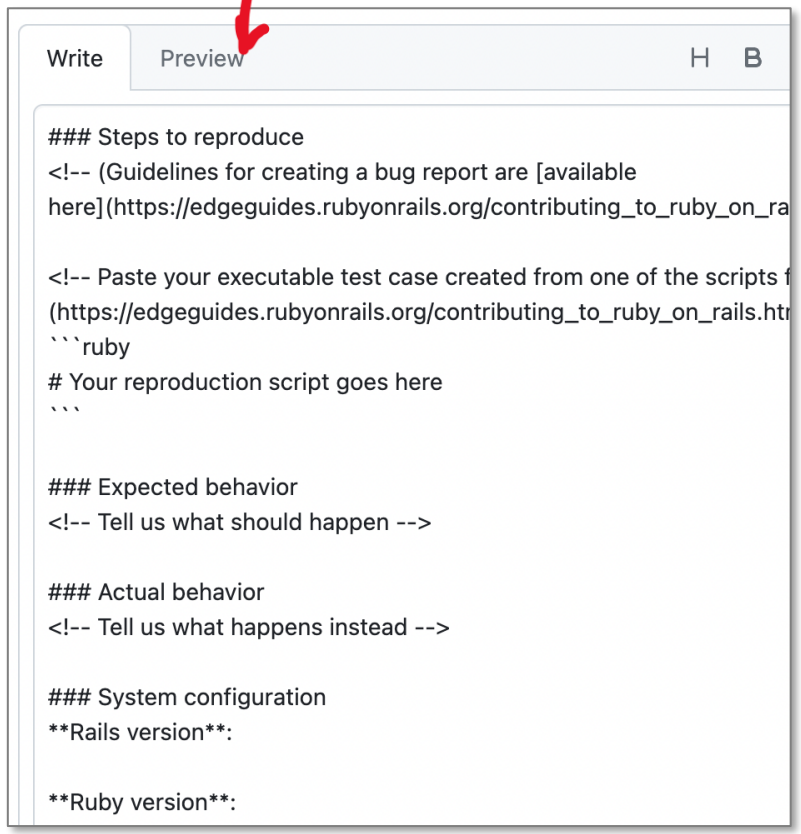



```
is:issue author:username label:bug
is:issue label:bug label:urgent state:open
is:issue assignee:username milestone:"v1.0"
is:issue no:assignee repo:username/repository comments:>=10
is:issue created:>=2022-01-01 created:<=2022-12-31
is:issue involves:username comments:0
is:issue language:python updated:>2022-01-01
```

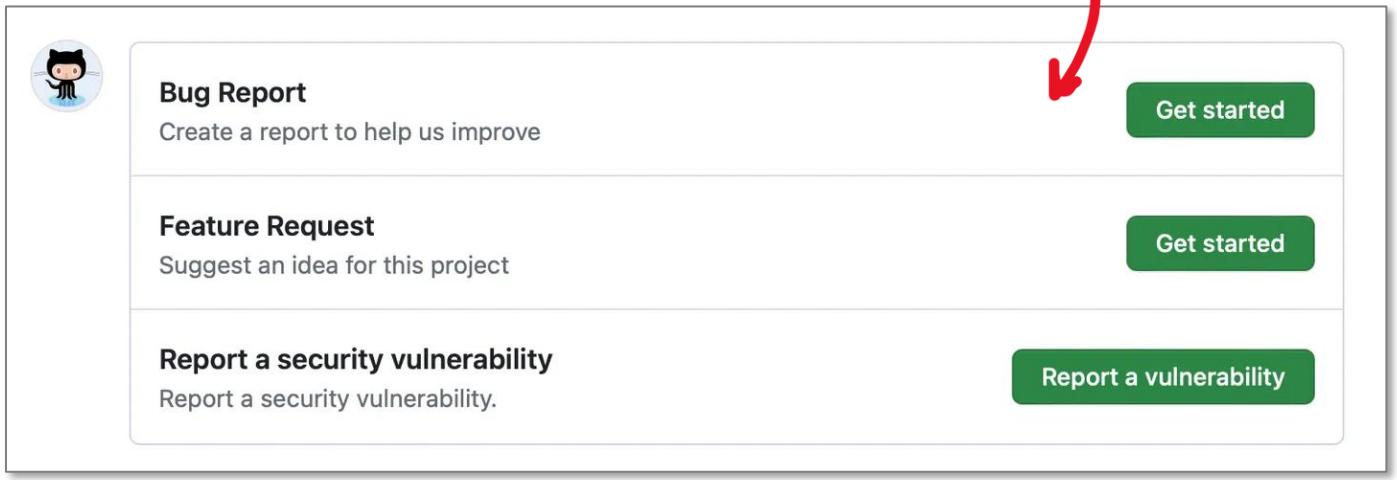
You can use **advanced search syntax**.

Issues Templates

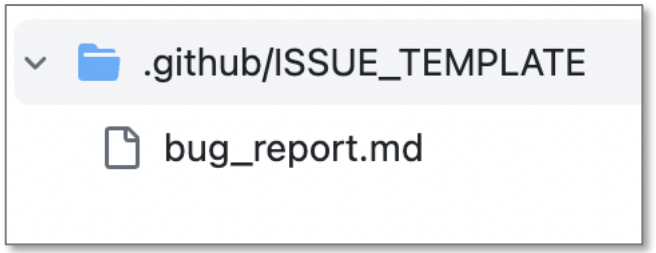
Issues Templates are **markdown templates that are preloaded for new Issues**.
They help ensure users creating issues provide all the relevant and expected information



You can **create multiple issues templates** to better improve the context of issues:



Issue Templates are stored as markdown files in **.github/ISSUE_TEMPLATES** folder in your repo

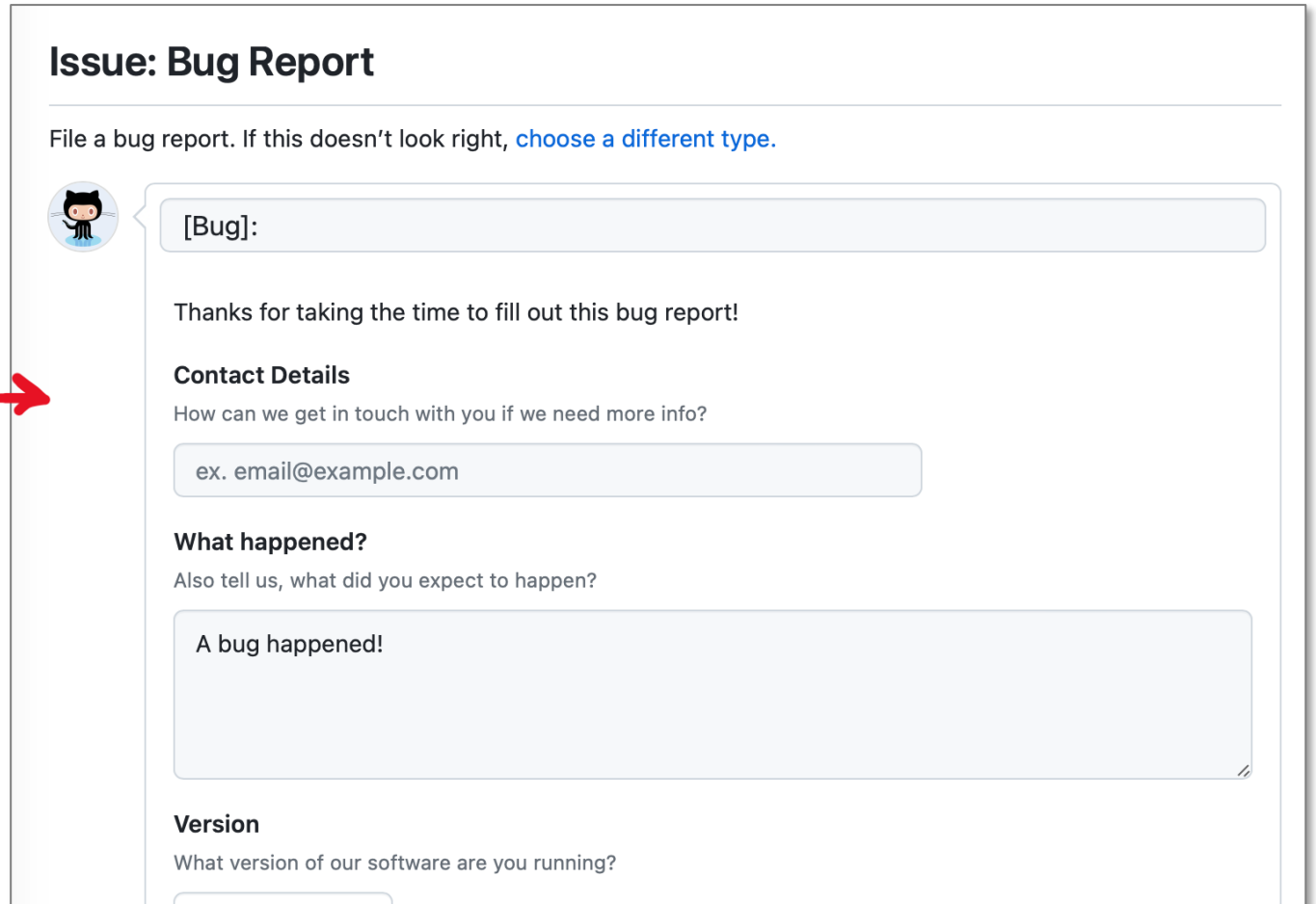


GitHub has a wizard GUI to easily create Issue Templates

Issues Forms

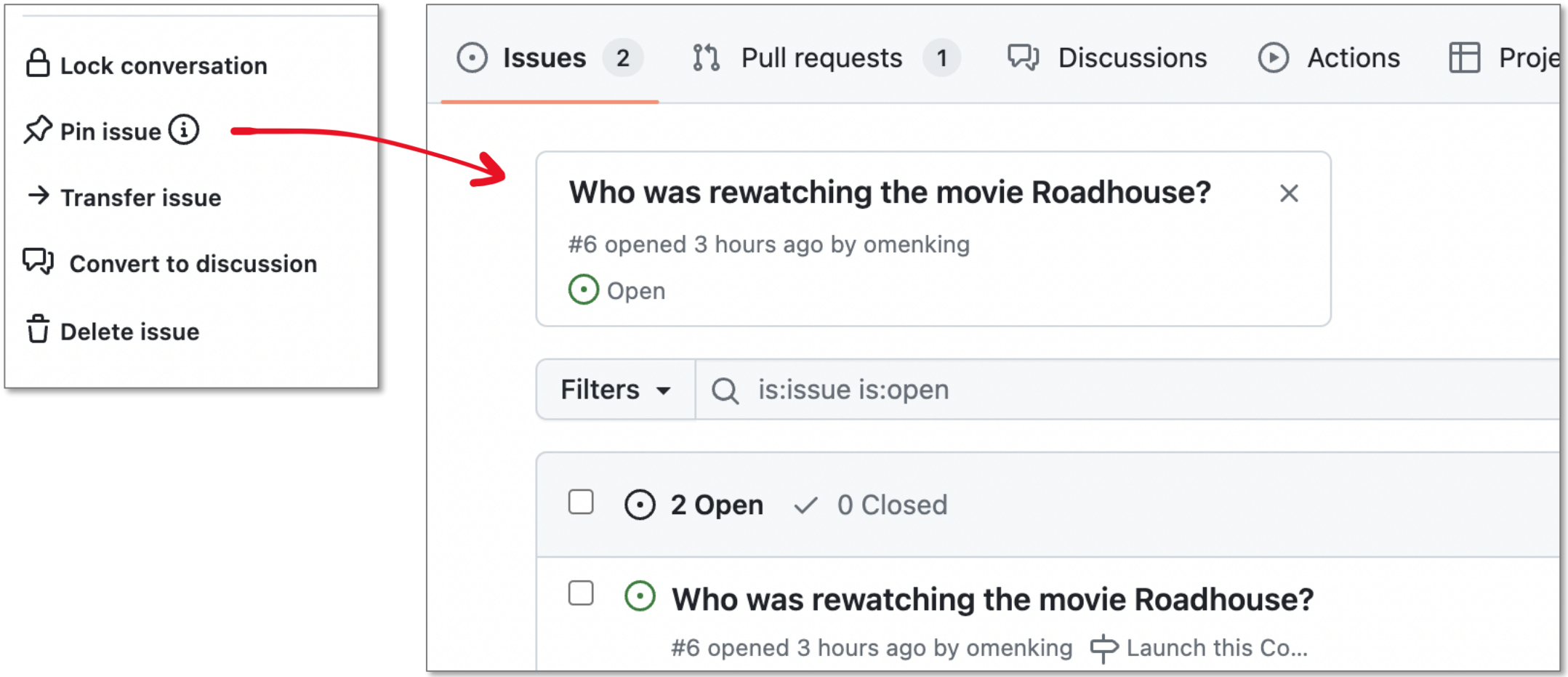
Issues Forms is the evolution of Issues Templates.
You use a **YAML formatted file** to create Issue forms
for stricter entry of issue information


```yaml
name: Bug Report
description: File a bug report
title: "[Bug]: "
labels: ["bug", "triage"]
projects: ["octo-org/1", "octo-org/44"]
assignees:
  - octocat
body:
  - type: markdown
    attributes:
      value: |
        Thanks for taking the time to fill out
  - type: input
    id: contact
    attributes:
      label: Contact Details
      description: How can we get in touch with
      placeholder: ex. email@example.com
    validations:
```



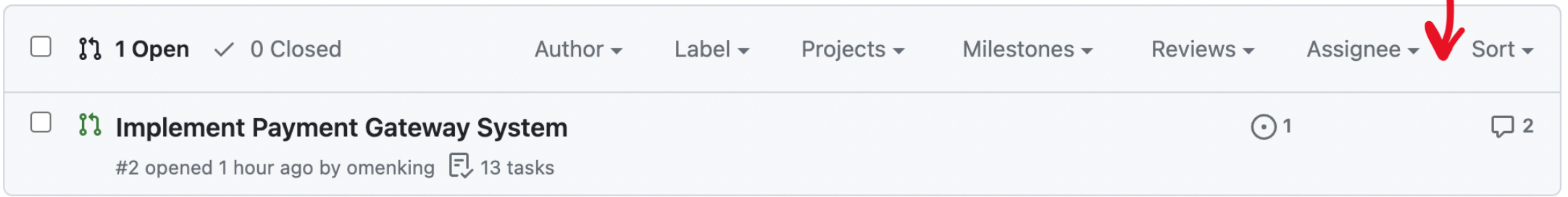
Pinning Issues

Up to 3 issues can be pinned so the appear at the top of the Issue page.



Pull Requests

A **Pull Request (PR)** is a **formal process to put forth changes**, that can be manually or automatically reviewed before its accepted into your base (main) branch.



Benefits of a Pull Request

- **Collaborative Review**: Enhances code quality through team discussions and peer feedback.
- **Change Tracking**: Provides a record of code changes and related discussions.
- **Automated Testing**: Enables integration with tools for automated checks and tests.
- **Controlled Integration**: Manages safe and reviewed merging of code changes.
- **Open Source Friendly**: Simplifies contributions and collaboration in open-source projects.

A pull request **is not a feature of git**, but a workflow. Services like GitHub can automate the Pull Request workflow

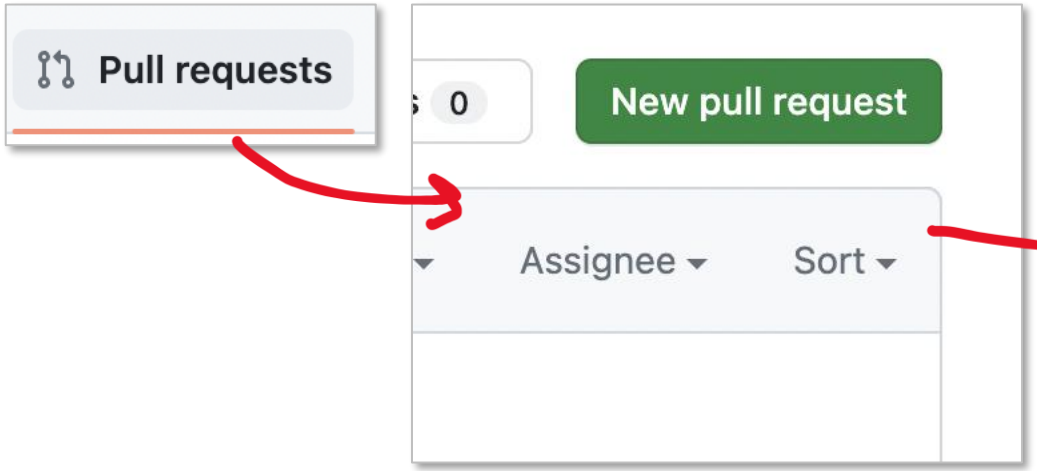
Creating Pull Requests

We set the:
Base – who we are going to merge into
Head – the changes to pull in

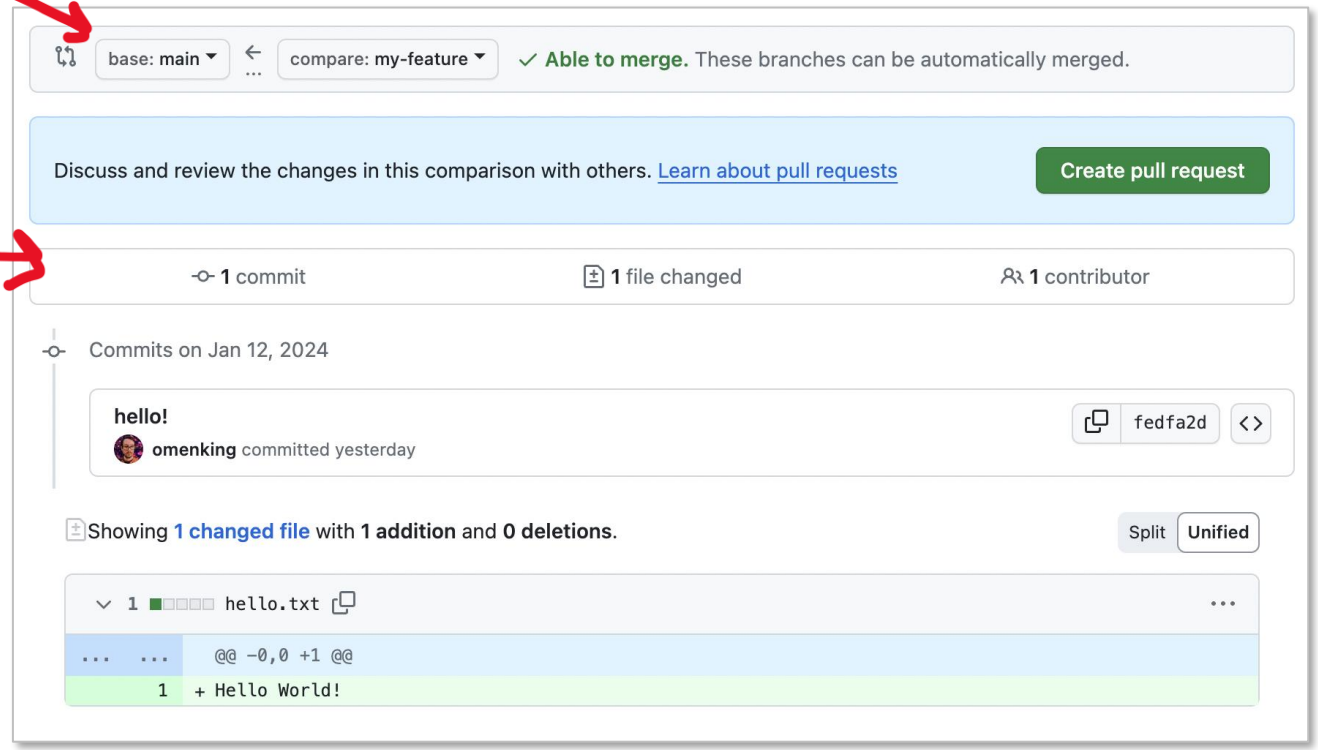
We can use the **Github CLI** to create a pull request from a branch.

```
git checkout -b cool-feature-branch
git commit -am "Your commit message"
git push -u origin cool-feature-branch
gh pr create --base main --head cool-feature-branch
```

In the GitHub repo we navigation to **Pull Requests**



We create a **new pull requests**



Pull Requests - Base and Compare

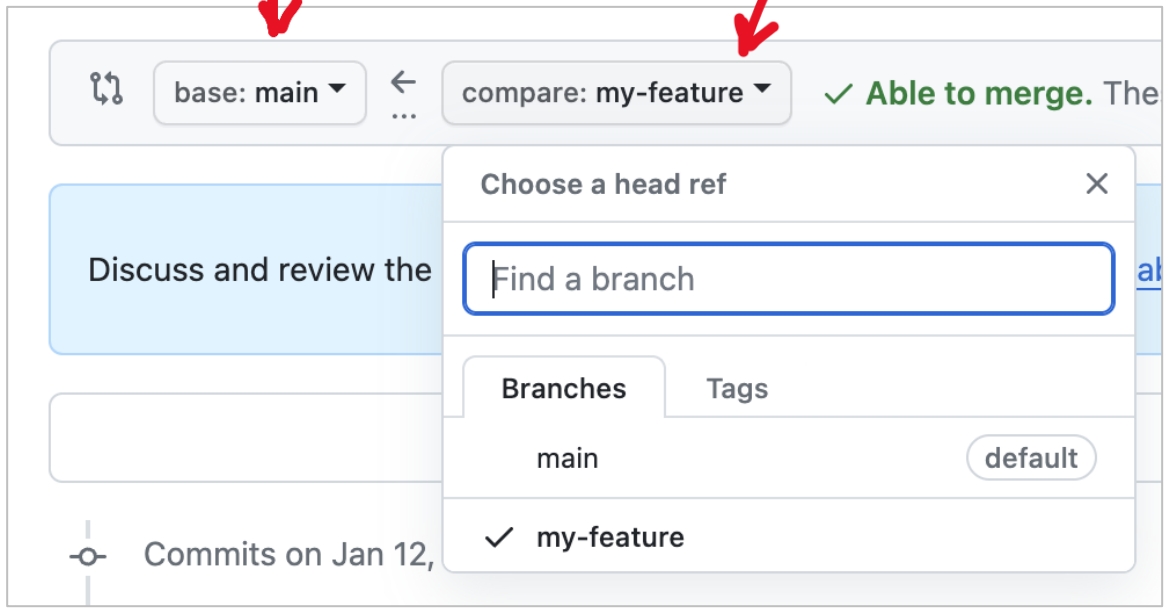
Base and Compare determines **the direction of the merge** for a pull request

Base is who you want to merge into

- This is usually main branch or an environment specific branch

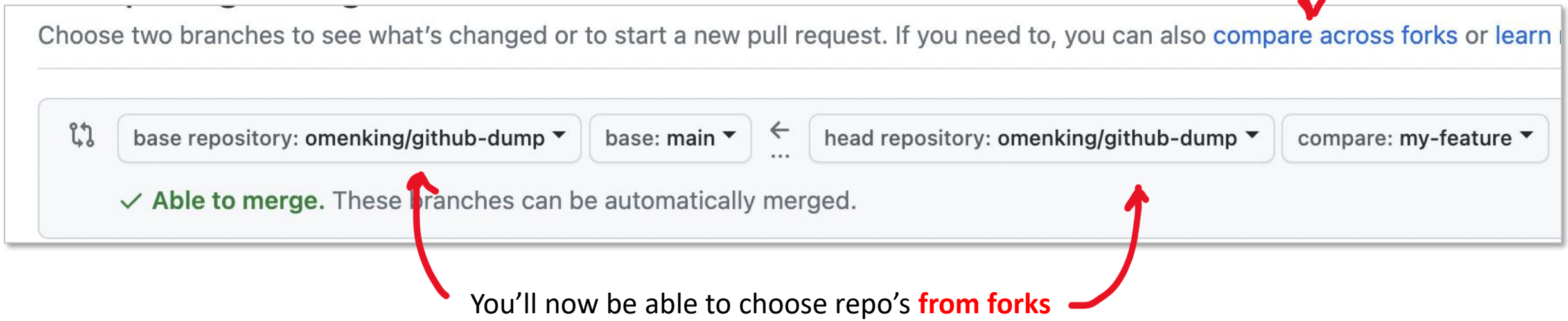
Compare is what will be merged into Base

- Compare is choosing a Head ref
- This usually a bug or feature branch



Pull Requests - Base and Compare

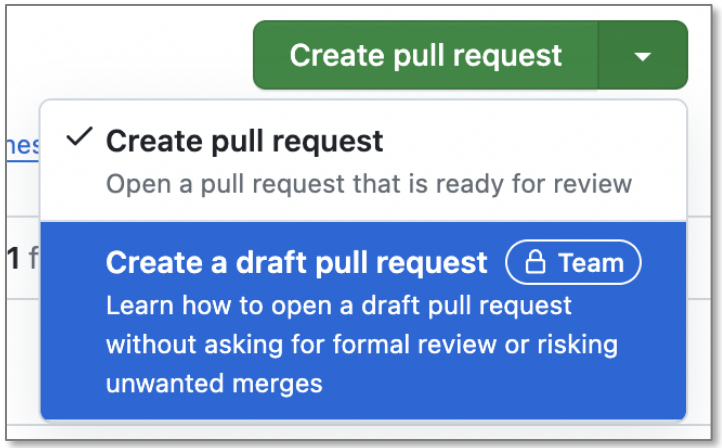
You can also **compare across forks** this is how a fork stays up to date, or how forks can suggest you to accept their changes.



You'll now be able to choose repo's **from forks**

Draft Pull Requests

A **Draft Pull Request** on GitHub is a feature that **allows you to open a pull request but mark it as a work-in-progress (WIP).**



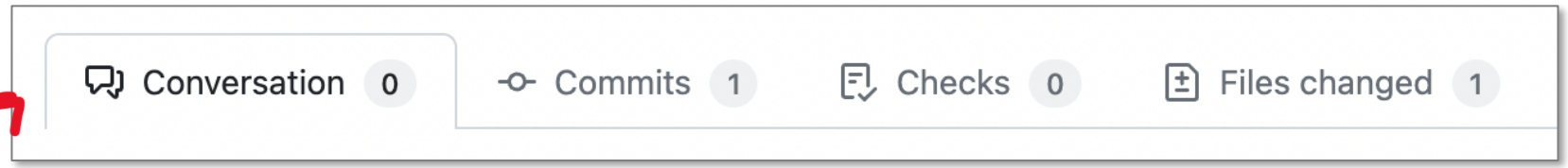
- Draft pull requests **cannot be merged**
- code owners are **not automatically requested to review draft pull requests**

Use case for Draft Pull Requests

- **Indicating Work-in-Progress:** Communicates that the pull request is not ready for review or merging.
- **Preventing Premature Merging:** Ensures incomplete work is not accidentally merged.
- **Facilitating Early Feedback and Collaboration:** Allows for early sharing and discussion of code changes.
- **Continuous Integration Testing:** Enables CI tests during the development process.
- **Transitioning to a Ready State:** Easy switch from draft to ready for final review and merging.
- **Organizing Work and Priorities:** Helps in managing and tracking ongoing work in large projects.

Draft Pull Requests is a feature only for **GitHub Organizations Teams**

Pull Requests Tabs

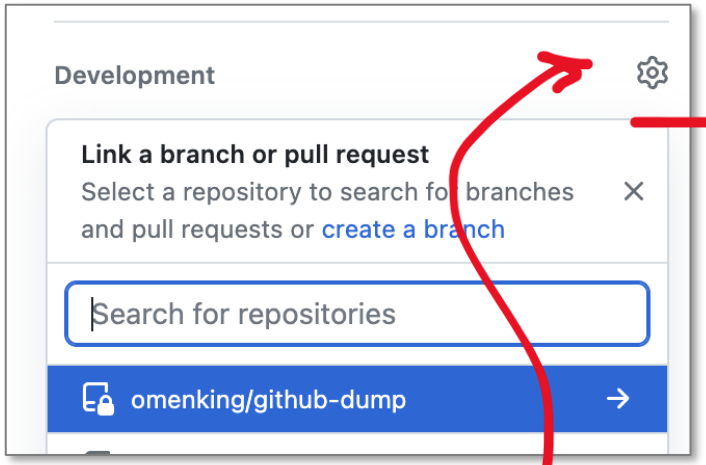


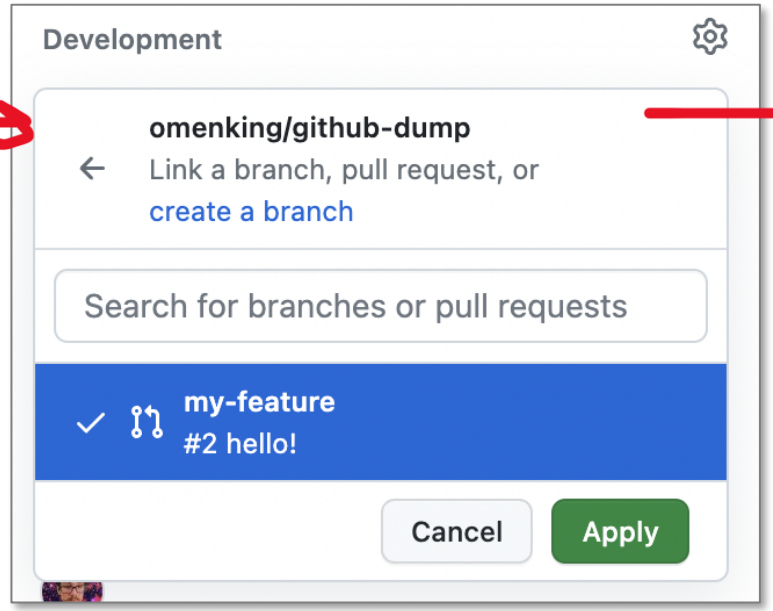
When you've created a Pull Request in GitHub you have the following tabs:

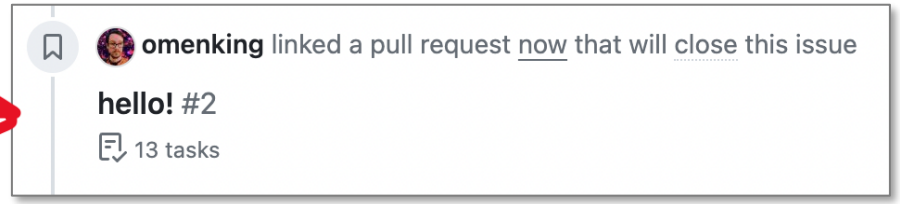
- **Conversation** — Threaded conversation where you can discuss the pull request
- **Commits** — Commits that are being merged into base
- **Checks** — Checks that need to be passed to merge PR.
  - could be apps installed or GitHub PR rules eg. Past test code suites
- **Files changes** — A list of files to be changed showing additions and deletions

Linked Activity within a Pull Request

You can Link Issues to Pull Requests so that the state of the pull request will automatically close the Issue



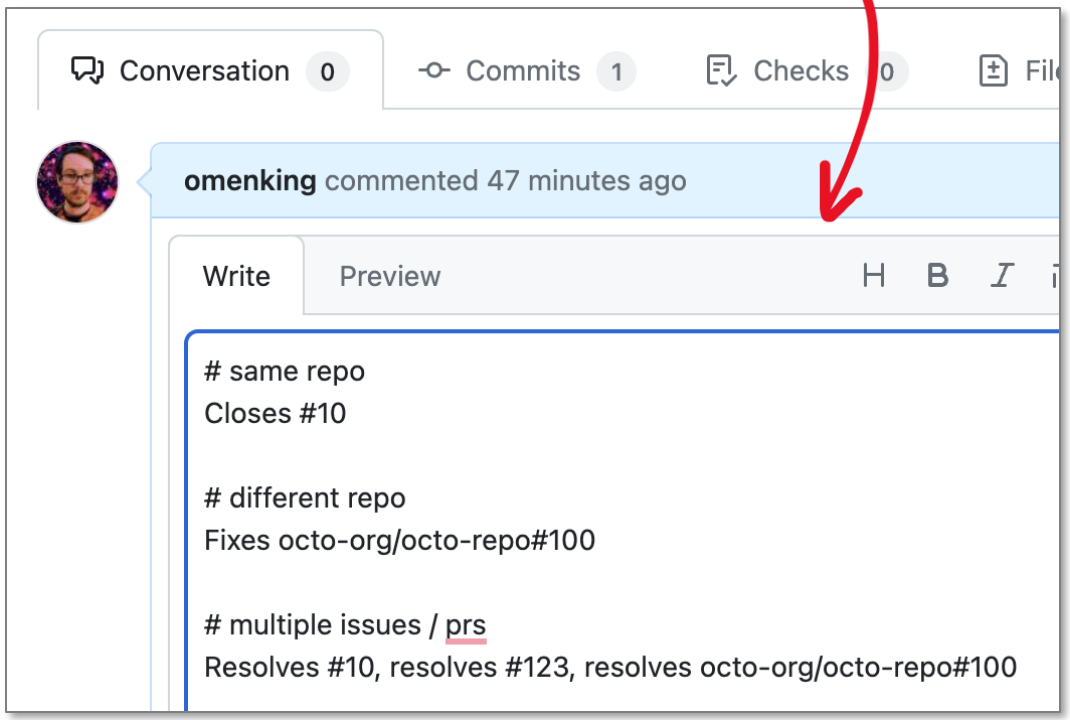




Within the Issue go to the right hand toolbar and click the **cog**

Linked Activity within a Pull Request

You can link a pull request to an issue by using a supported keyword in the **pull request's description** or in a commit message.



Supported Keywords

- close
- closes
- closed
- fix
- fixes
- fixed
- resolve
- resolves
- resolved

⚠ The pull request **must be** on the default branch.

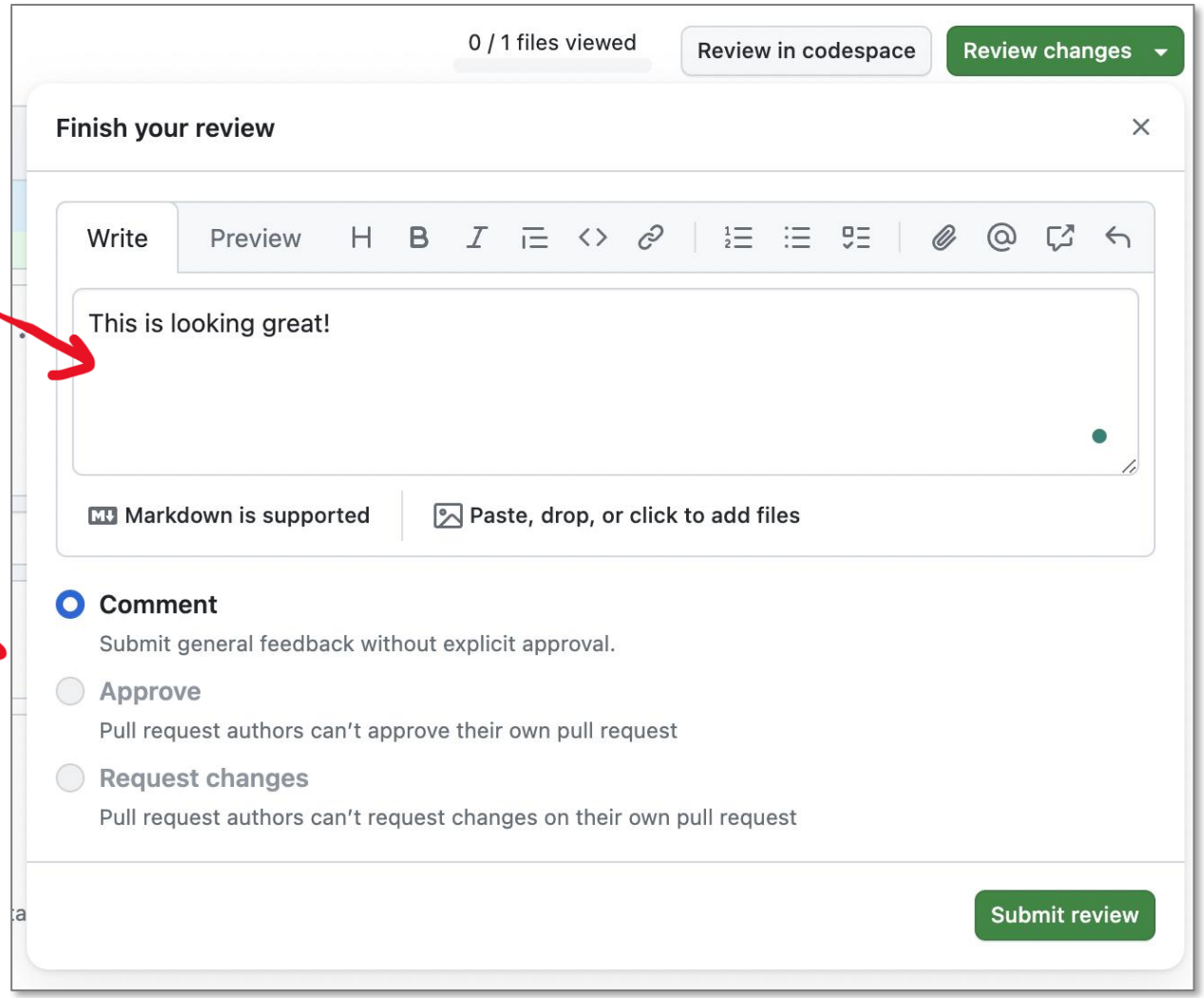
Pull Request Statuses

Open	The default status when a pull request is created. It's open for discussion and review.
Draft	Indicates the pull request is a work-in-progress and not yet ready for review.
Closed	The pull request has been closed without being merged. This status is used when the proposed changes are no longer needed or if the branch has been rejected.
Merged	The pull request's changes have been merged into the target branch. This status indicates a successful conclusion of the pull request process.
Changes Requested	This status is used during the review process when a reviewer requests changes before the pull request can be merged.
Review Required	Indicates that the pull request requires a review before it can be merged. This status is common in repositories where reviews are a mandatory part of the workflow.
Approved	The pull request has been reviewed and approved for merging by the required number of reviewers.
Conflict	Indicates that there are conflicts between the pull request's branch and the target branch that need to be resolved before merging.
Ready for Review	A pull request initially marked as draft can be changed to this status once it's ready for review.

Commenting in Pull Requests

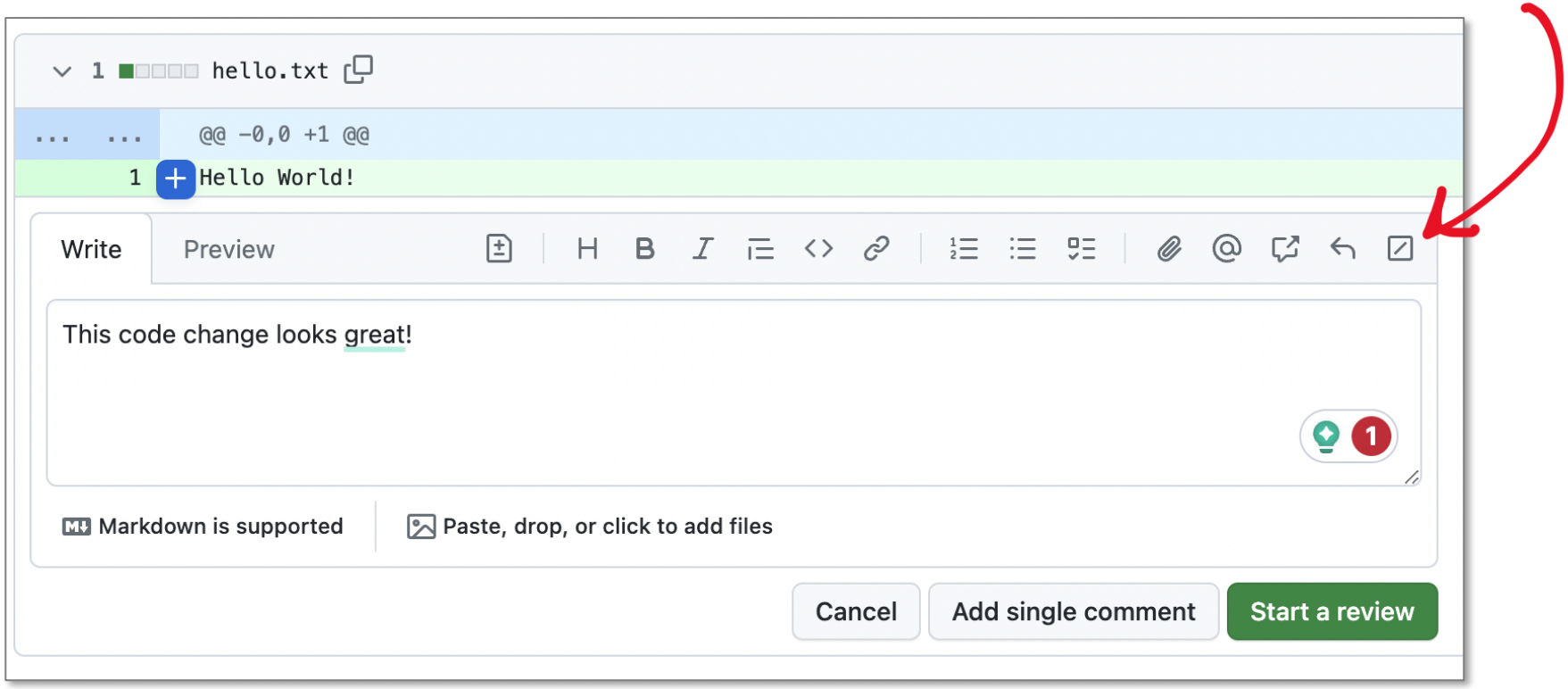
When you comment you are submitting a review to be considered before a Pull Request is being accepted.

If you are not the author, but also a designated reviewer you can **request changes** or mark the Pull Request as **Approved** so it can be considered for merging.



Commenting in Pull Requests

Under **Files Changed** you can also apply a review on **a very specific line of code.**



Code Reviews with CODEOWNERS file

CODEOWNERS GitHub repo specific file to define individuals or teams that are responsible for **specific code** in a repository

```
*       @global-owner1 @global-owner2
*.js    @js-owner
*.go docs@example.com
*.txt @octo-org/octocats
/build/logs/ @doctocat
docs/*  docs@example.com
apps/ @octocat
/docs/ @doctocat
/scripts/ @doctocat @octocat
**/logs @octocat
/apps/ @octocat
/apps/github
/apps/ @octocat
/apps/github @doctocat
```

CODEOWNER files uses a **similar syntax to .gitignore.**

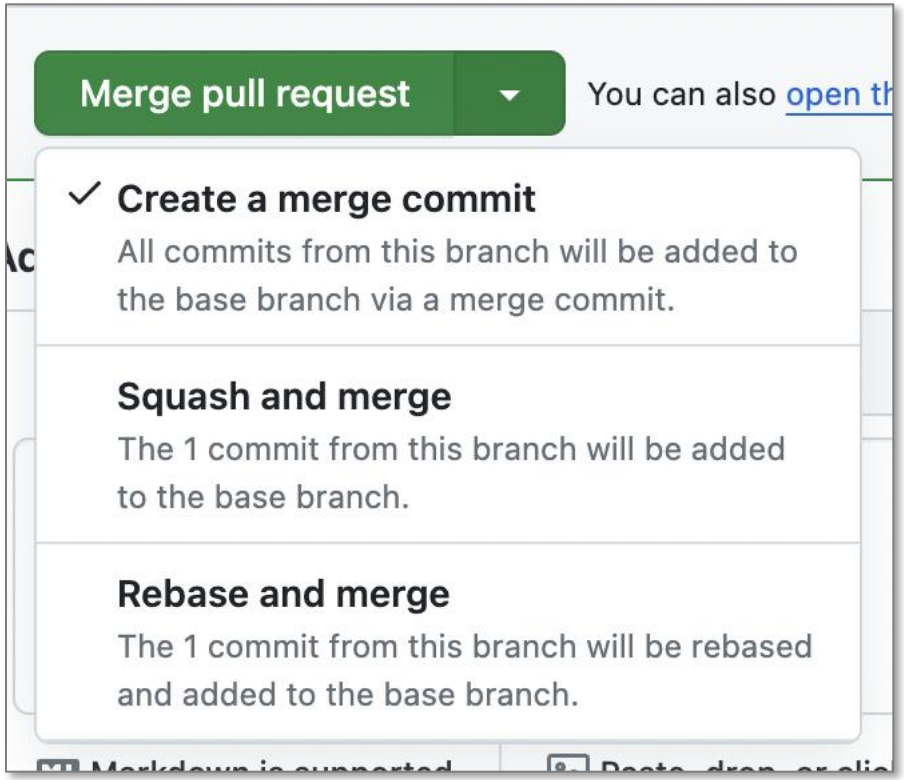
When a pull request is opened that modifies any files matching a pattern in the CODEOWNERS file, GitHub **automatically requests a review from the specified code owners.**

The CODEOWNERS files goes in either the project root, .github or docs directory.

Pull Request Options

When merging a pull request there are a few options

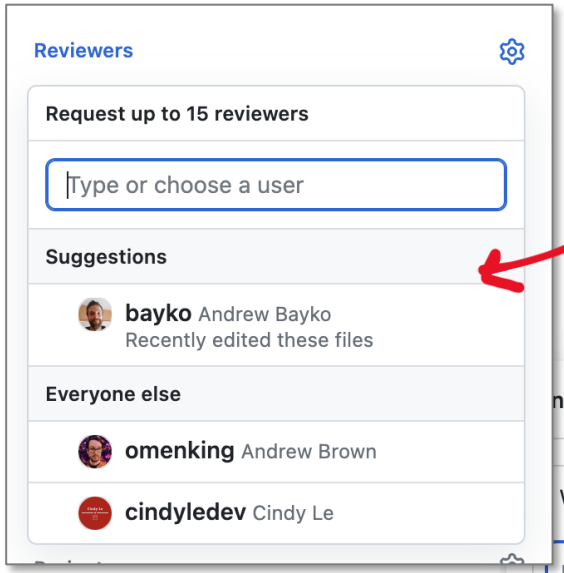
1. **Create a merge commit**
   - All commits will be added
2. **Squash and merge**
   - 1 commit will be added
3. **Rebase and merge**
   - 1 commit will be added and rebased

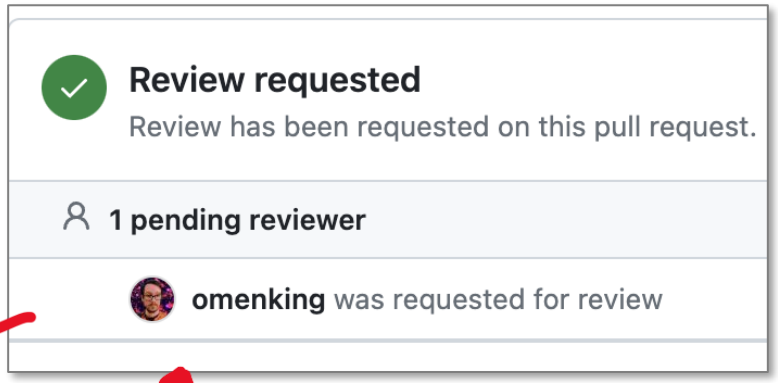


The use-case depends on your team's workflow. They may prefer only a single commit is added to keep the base branch git tree clean and readable.

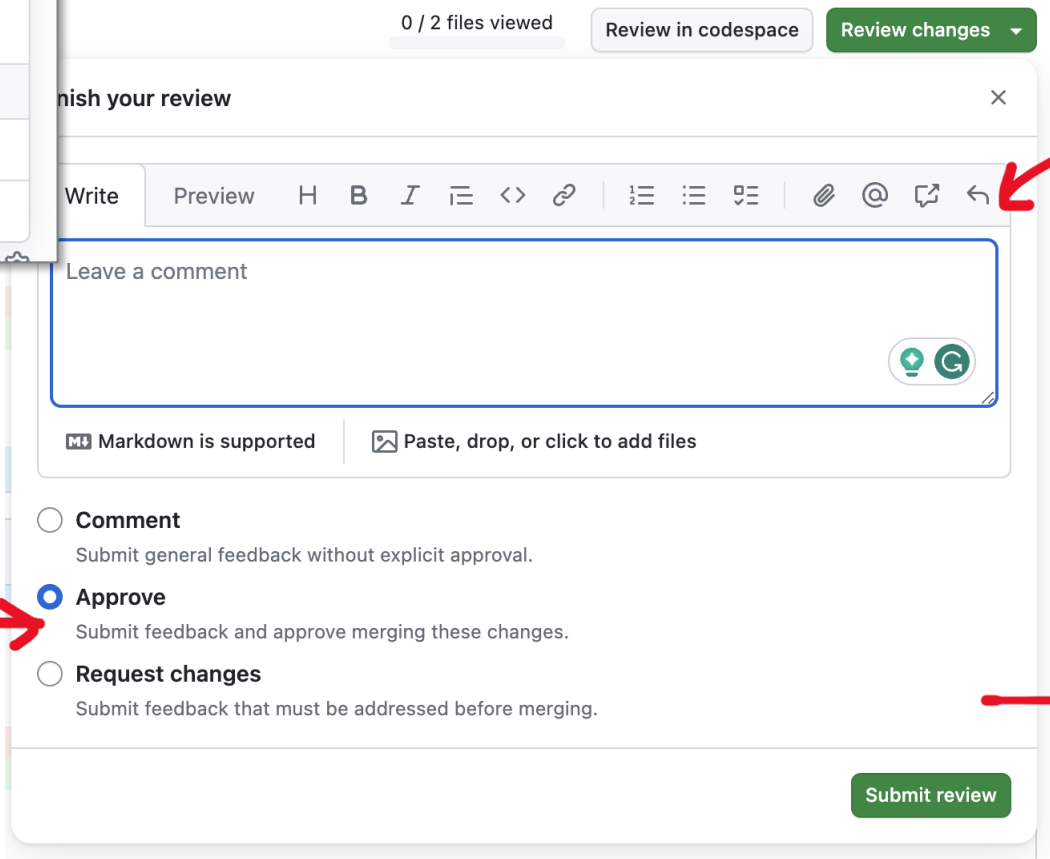
Required Reviewers

Pull Request can have **multiple required reviewers** who must write a review that approves the changes.



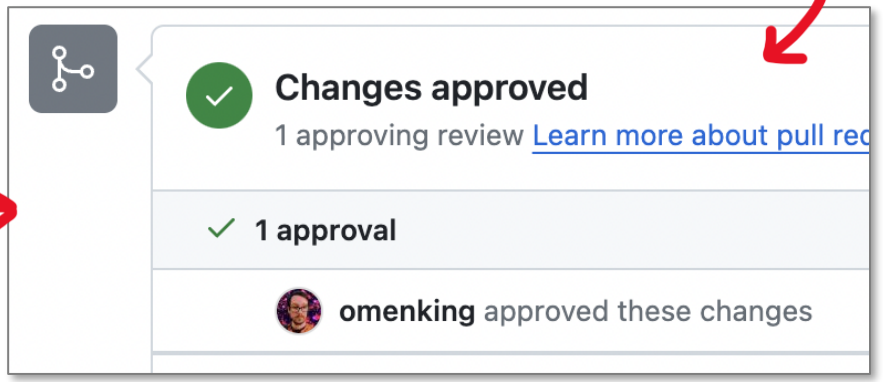


The review is expected



The review with **Approve** is provided

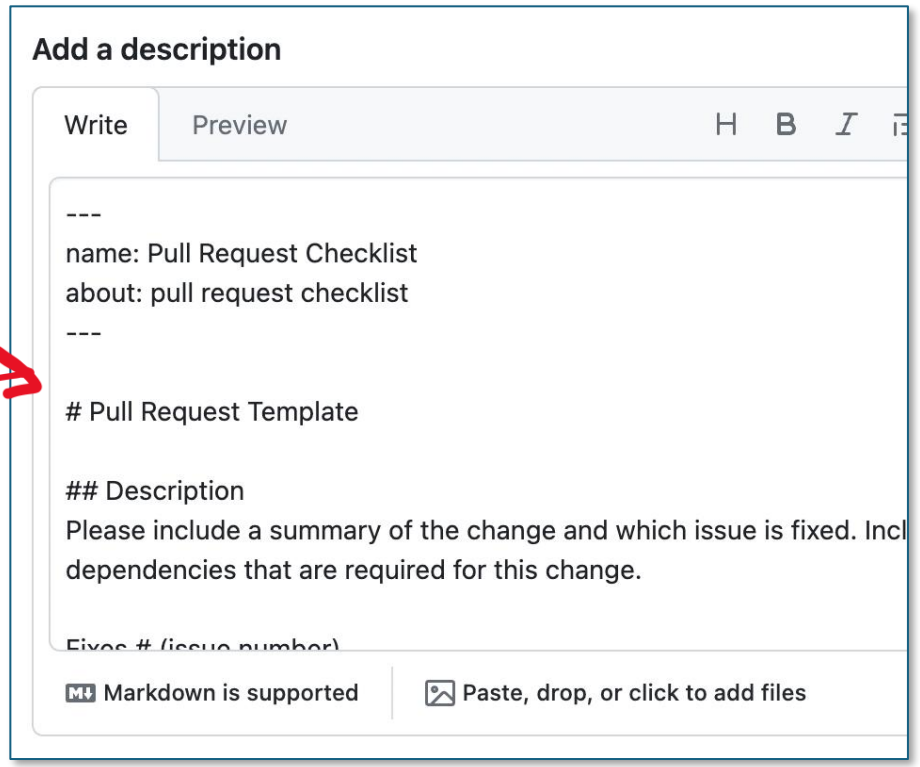
The changes are approved **passing the check** for the repo



Pull Request Templates

Pull Request Templates are similar to Issue Templates
They will populate the pull request textarea with the specified template

You create a a file in the
.github/pull_request_template.md



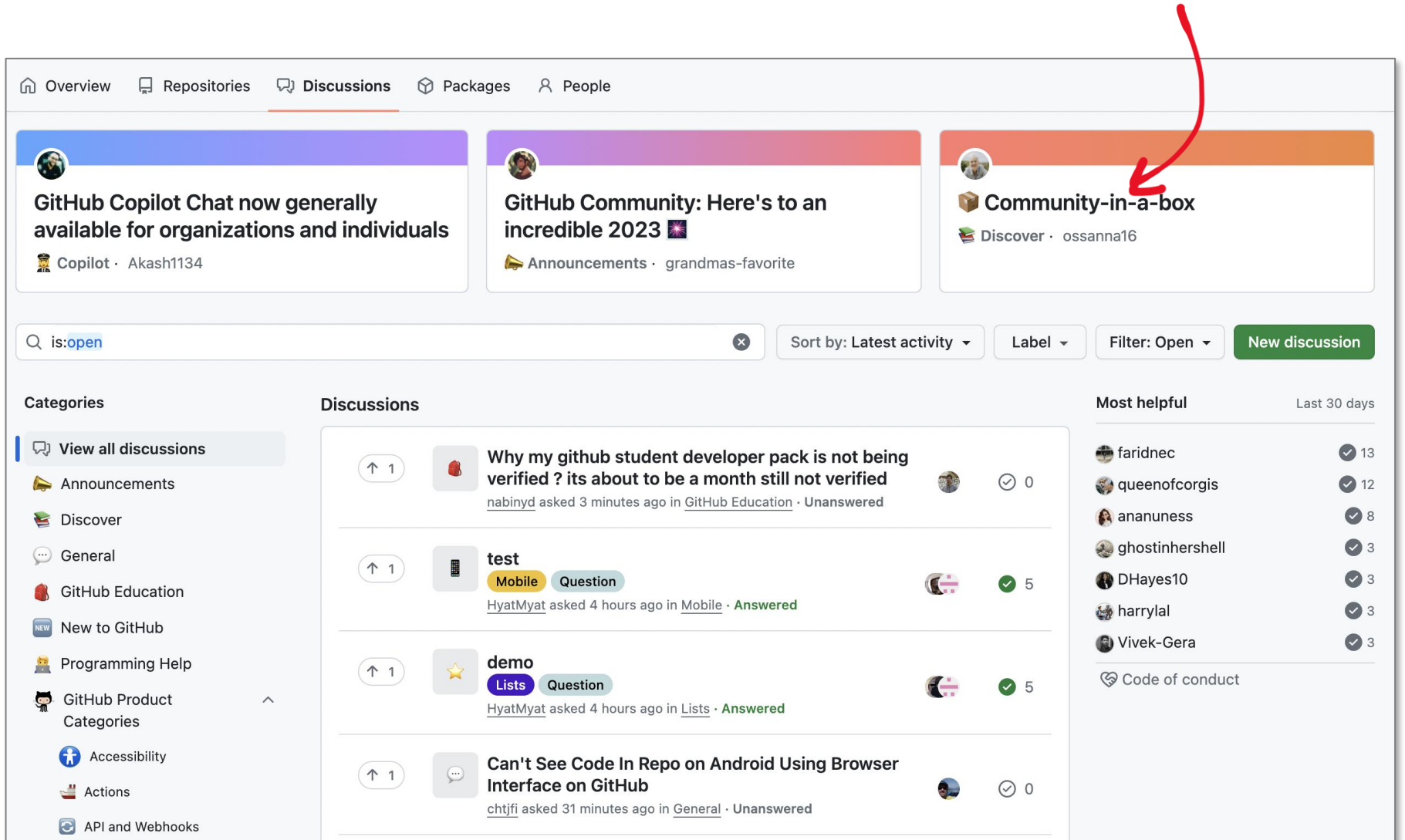
Technically you can create multiple Pull Request Templates in a folder called .github/PULL_REQUEST_TEMPLATE

GitHub kind-of supports multiple Pull Requests Templates but you have to assemble your own URL with a query strings, so **its not as convenient to use as multiple Issue templates**.

GitHub Discussions

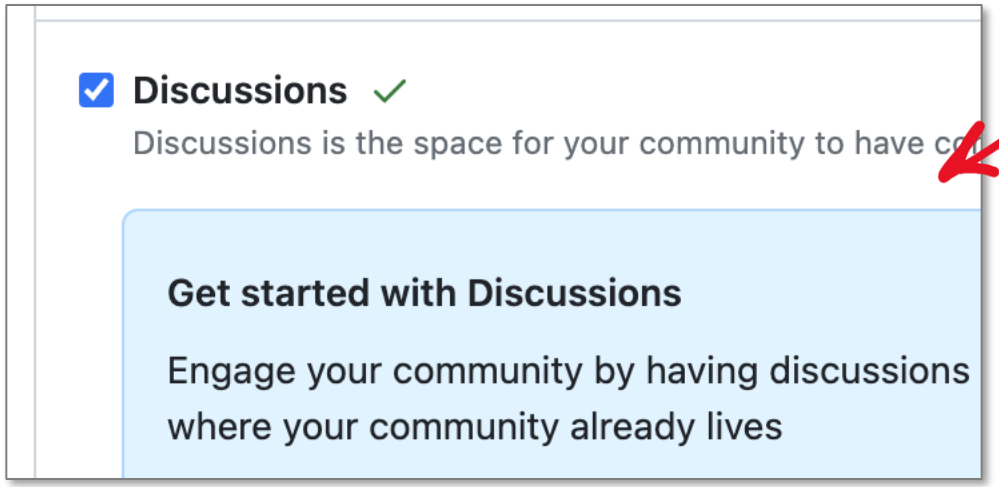
GitHub Discussions is **community communication tool** for your public or private GitHub Repos.

- Threaded Conversations
- Categorization
- Community Interaction
- Markdown Support
- Pin Discussions
- Conversation Polls
- Discussion Voting
- Conversion to Issues
- Notifications
- Searchable and Linkable
- GitHub Integration

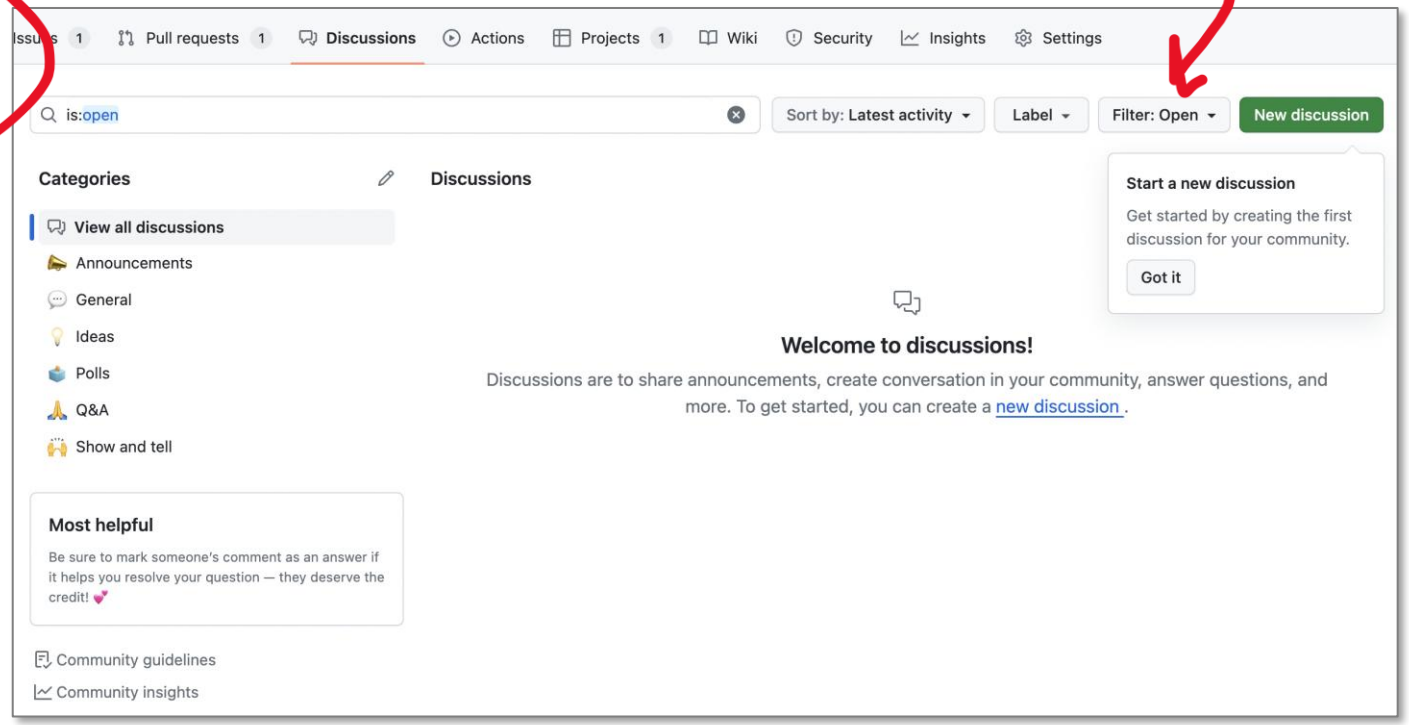


Discussions

Turn on Discussions under your **GitHub Repo Features**

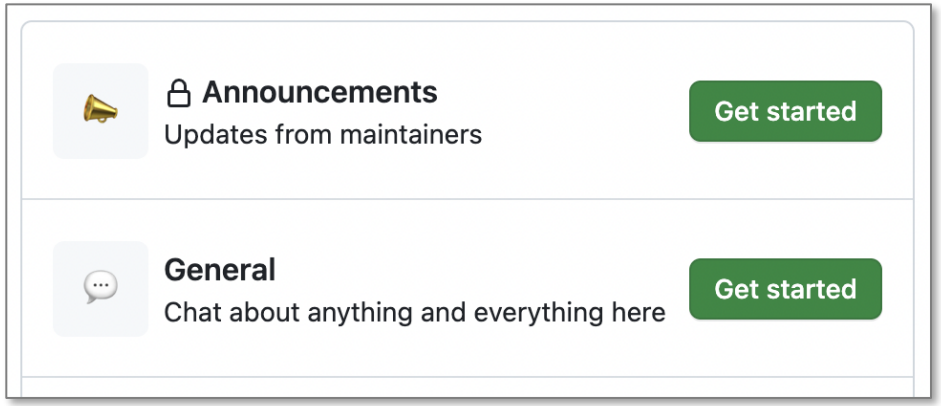


A **Discussions tab** will appear in your GitHub Repo.
You'll have some default categories available.

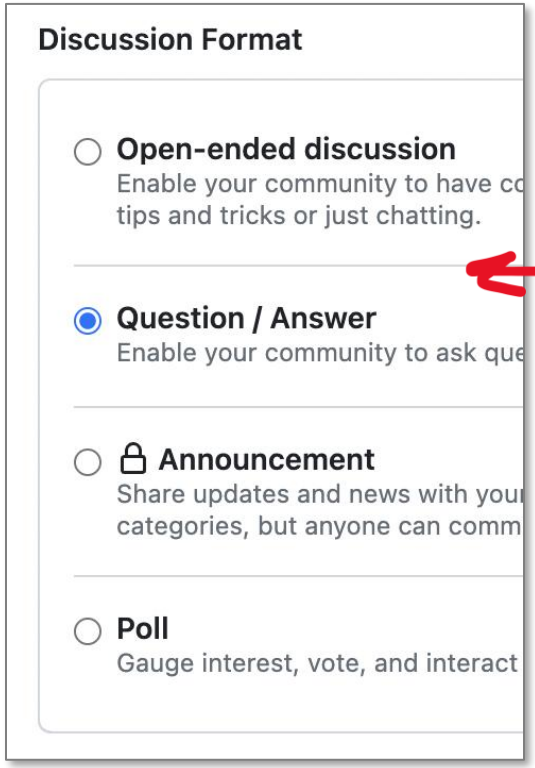


Discussion Options

When someone creates a Discussion, they **need to choose a Category.**



GitHub provides some Categories, but GitHub repo owners **can create their own.**



When you create a new category. you will choose a **Discussion Format**:

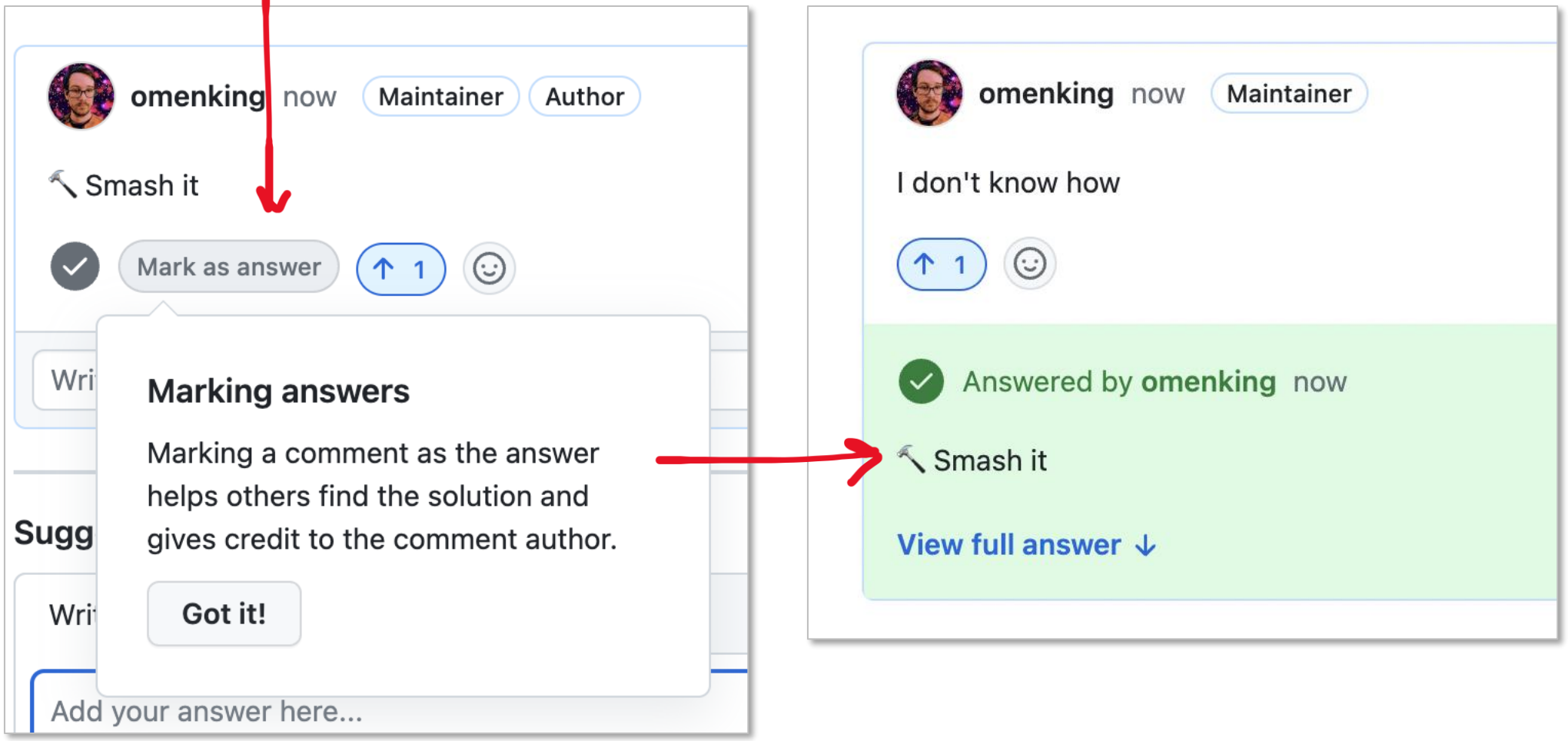
- **Open-ended discussion** — no voting of best answer, just open discussions
- **Question / Answer** — vote on the best suggested answer.
- **Announcement** — admins or maintainers can only post, anyone can reply
- **Poll** — Create Poll and let members vote

1
2

Categories can be grouped into **sections**

Marking Answers in a Discussion

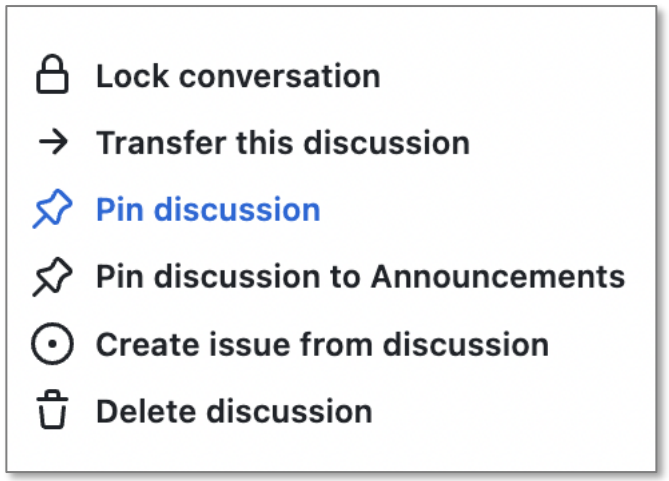
For **Q&A Discussion Formats** you can **mark an answer** as the recommended answer regardless of upvotes to help direct which question you think is the most credible.



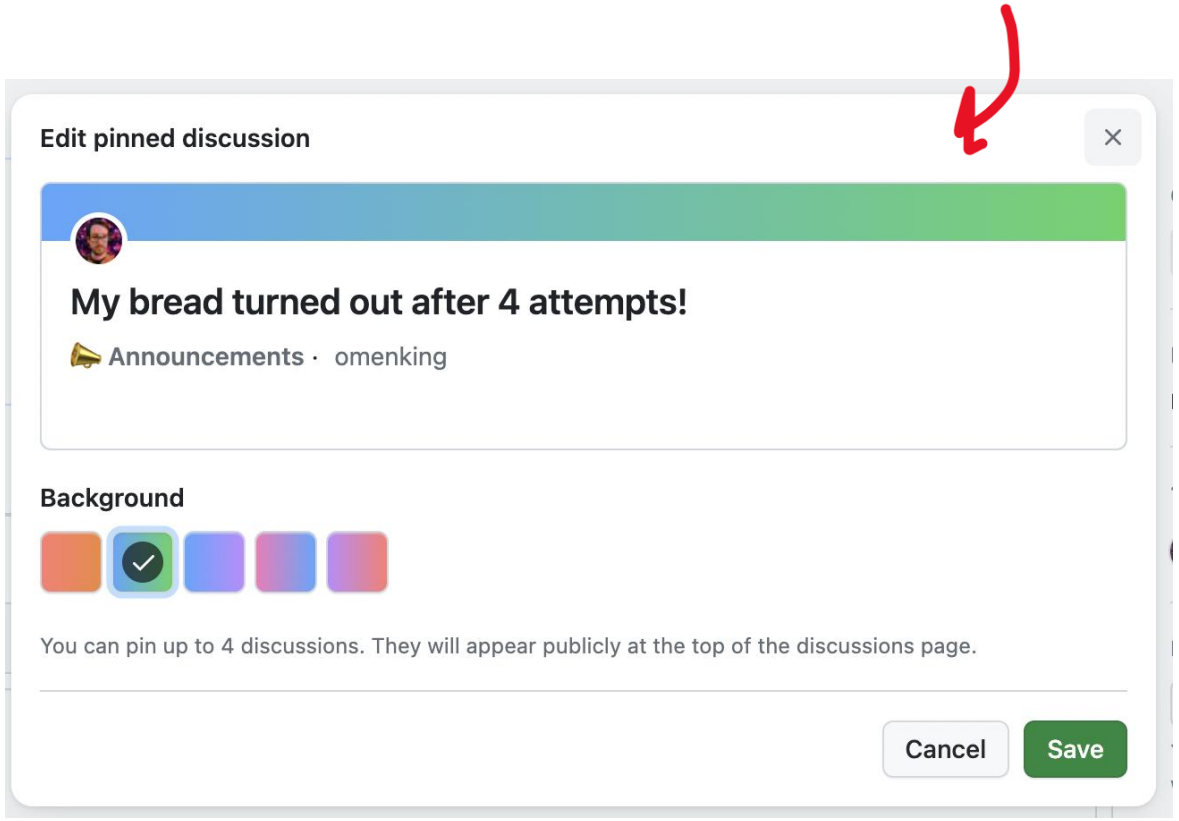
Pinning Discussions

Pinning Discussions puts discussions fixed at the top of the community page.

You can **pin** a discussion the Discussions home page or for a specific category



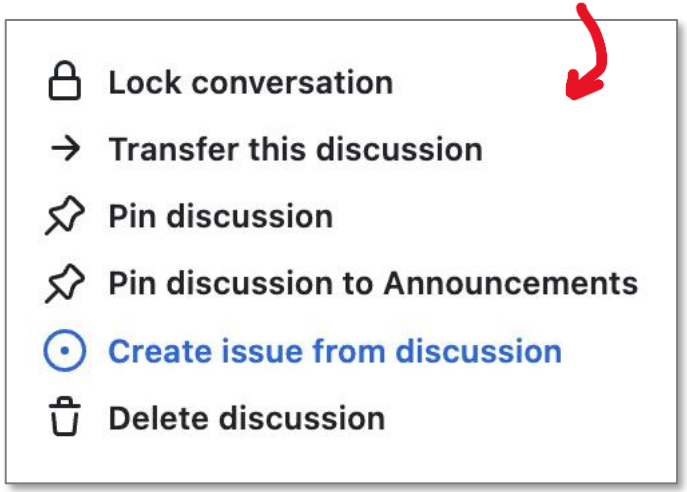
You can **customize** the look of a pinned discussion.



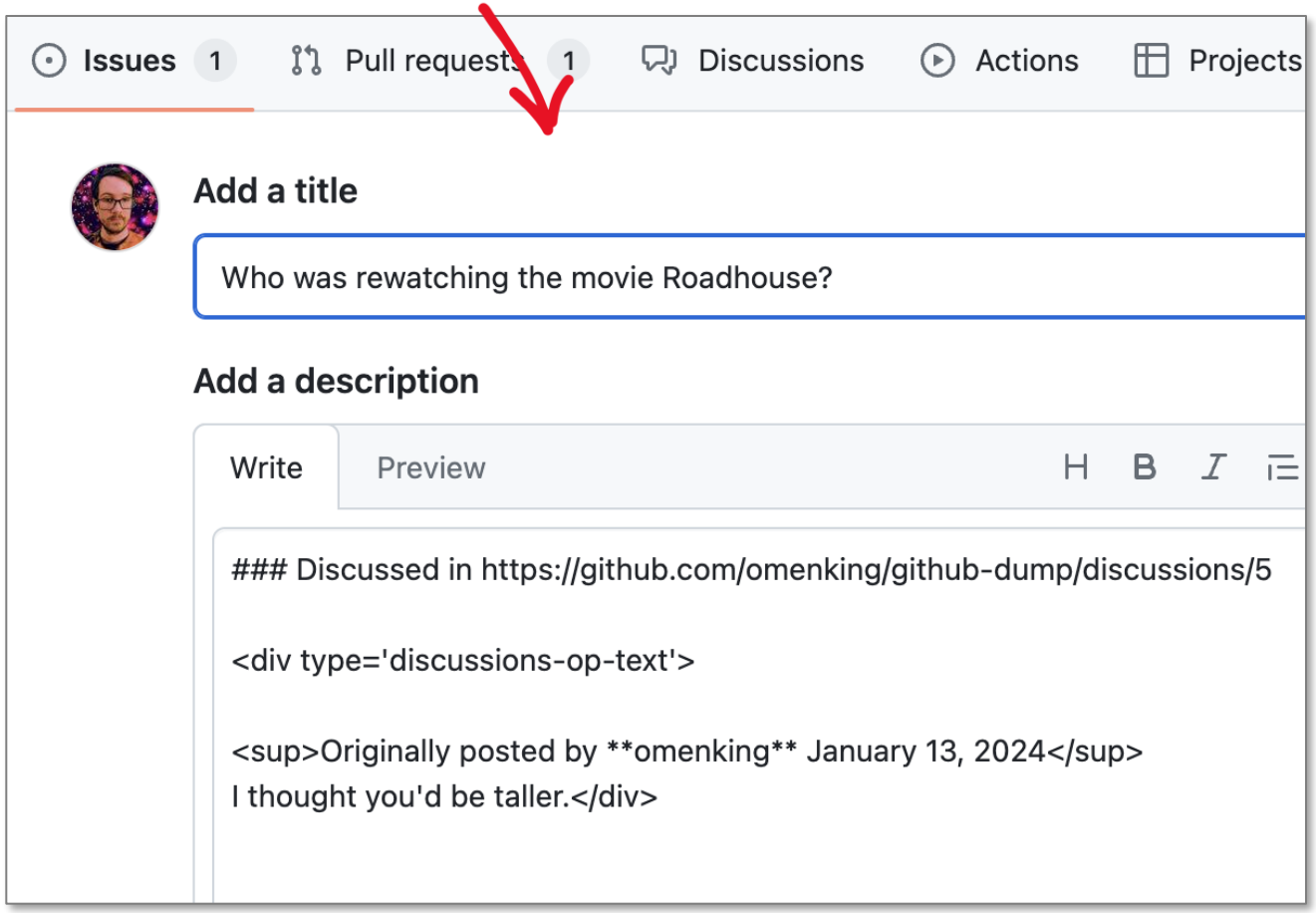
Convert Discussion to an Issue

You can **"convert"** a Discussion to an Issue

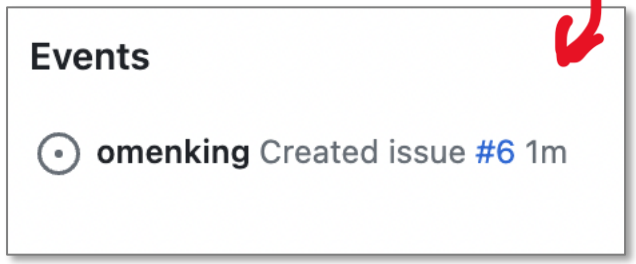
In the discussion choose to **create issue**.



It will **prepopulate an Issue** and place a Discussion link in the description

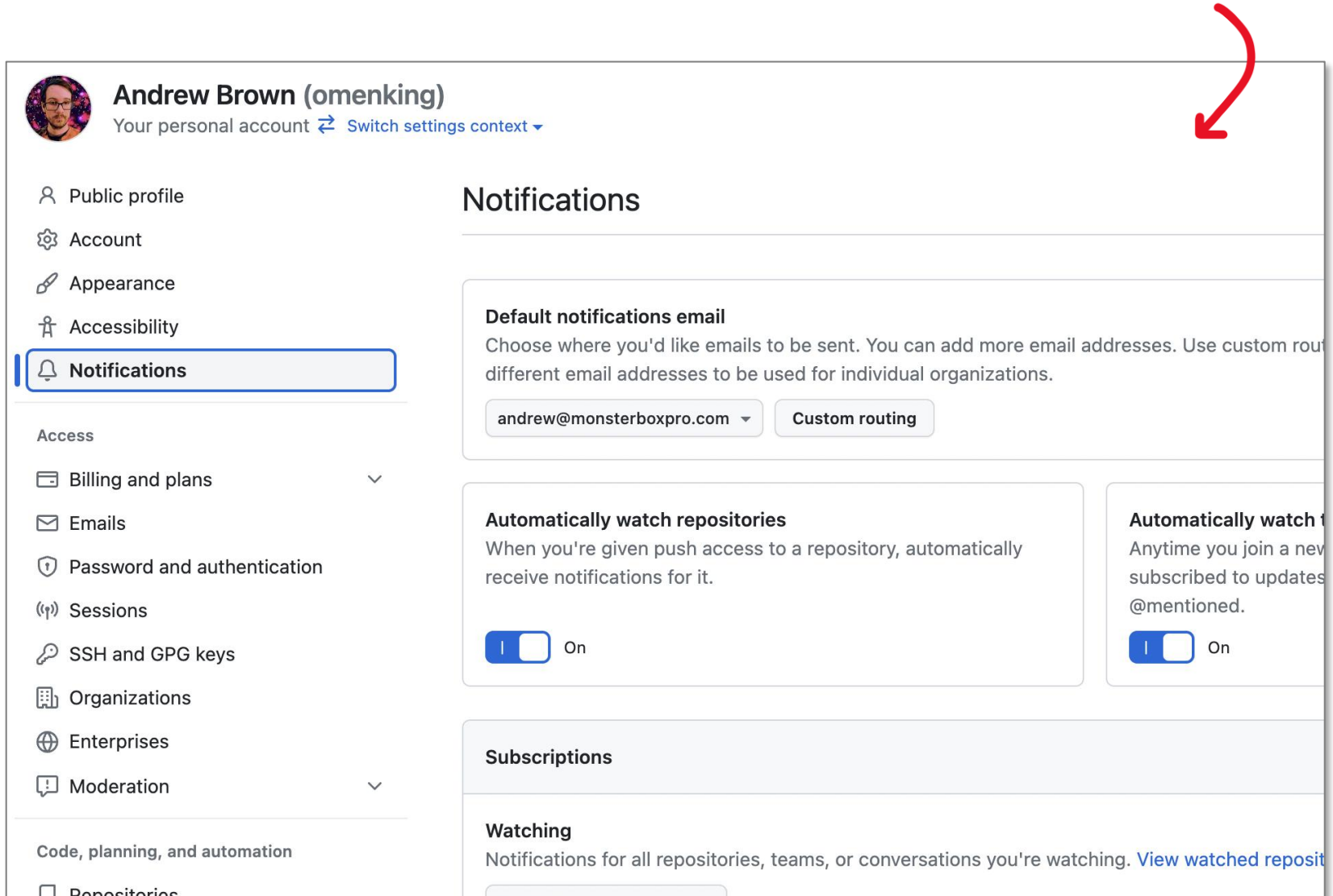


Under the **Discussion's Events** it will have a link to the issue



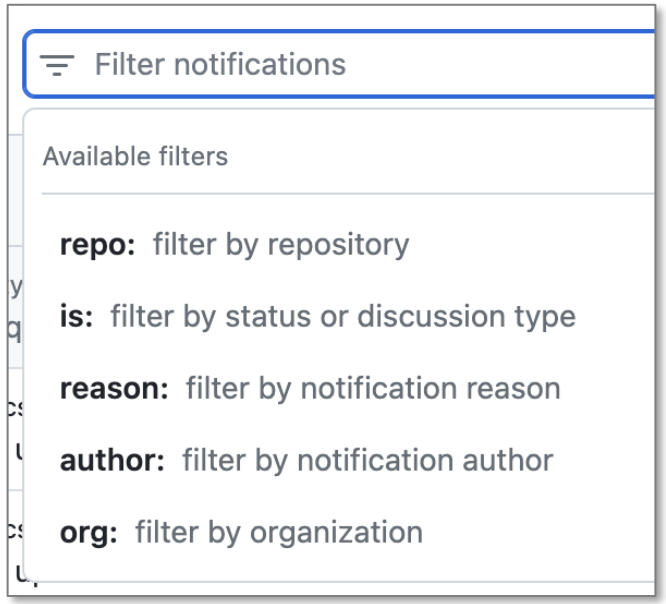
Notification Configuration Options

In your account settings you have a Notifications tab to configure **notifications options**.

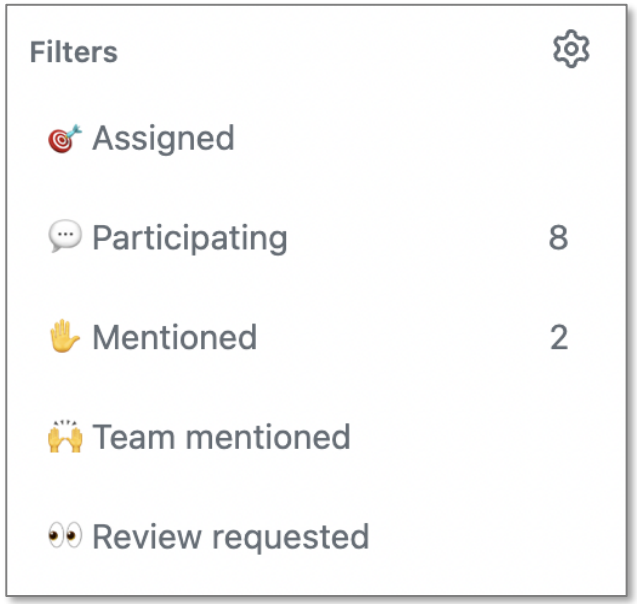


Notification Filters Options

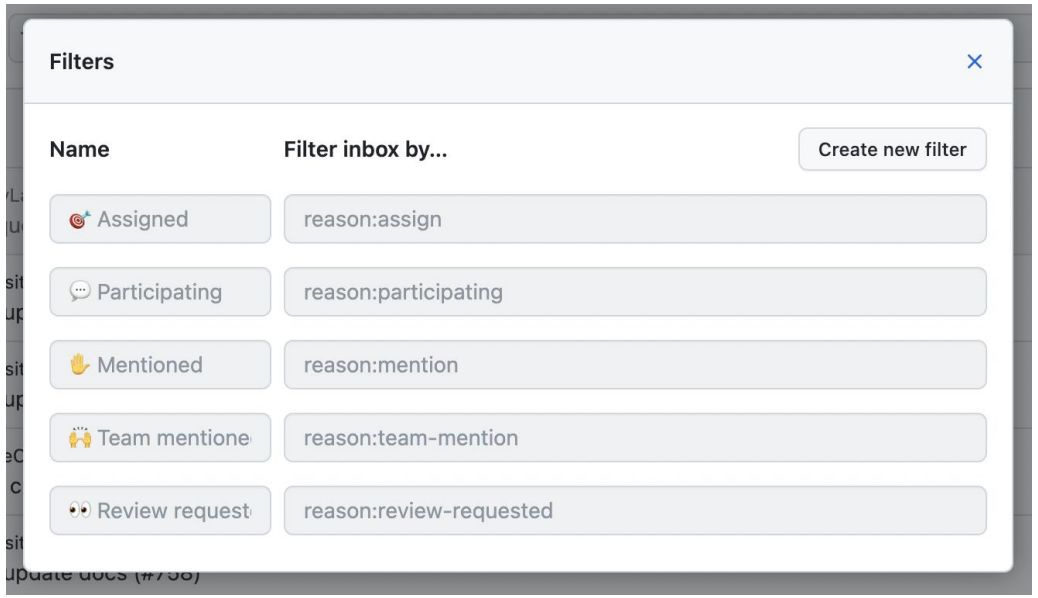
You can filter notifications using a **filter syntax:**

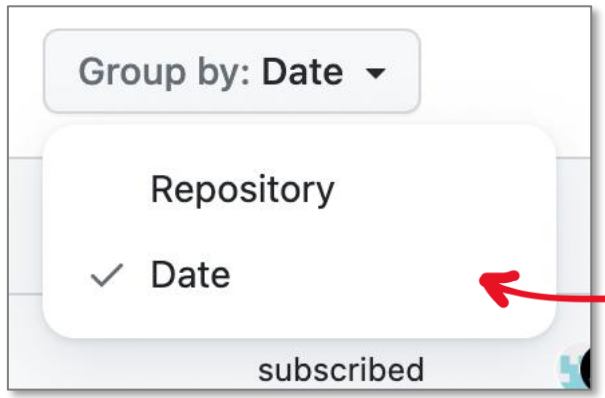


There are **predefined saved filters**



You can **create more saved filters** using the filter syntax



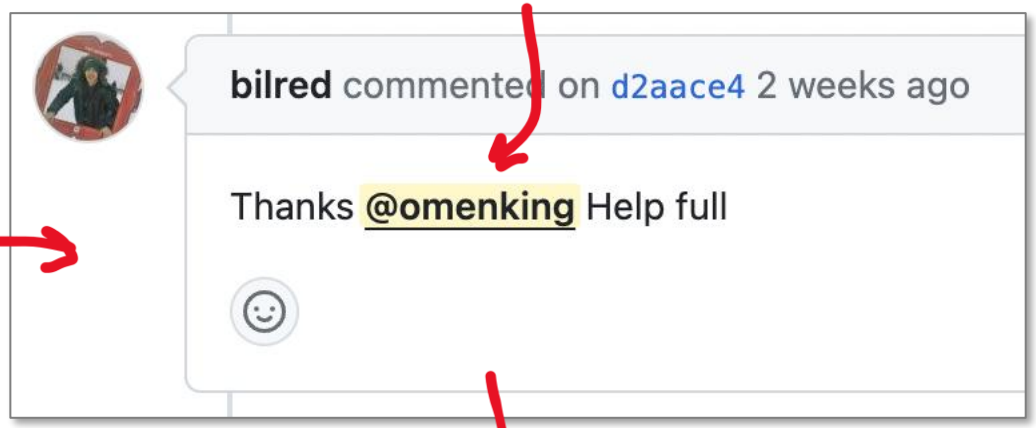


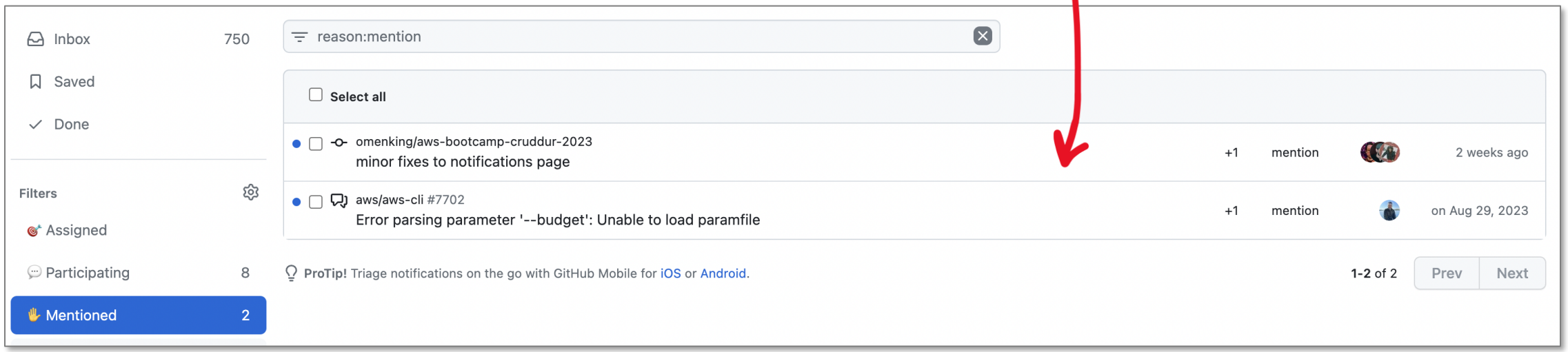
You can **group** Notification Threads by Repo or by Date.

Mentions in Threads

A mention is when someone using a @ followed by your username

If someone mentions you in a thread than you can find these mention under the **Mentioned Filters.**

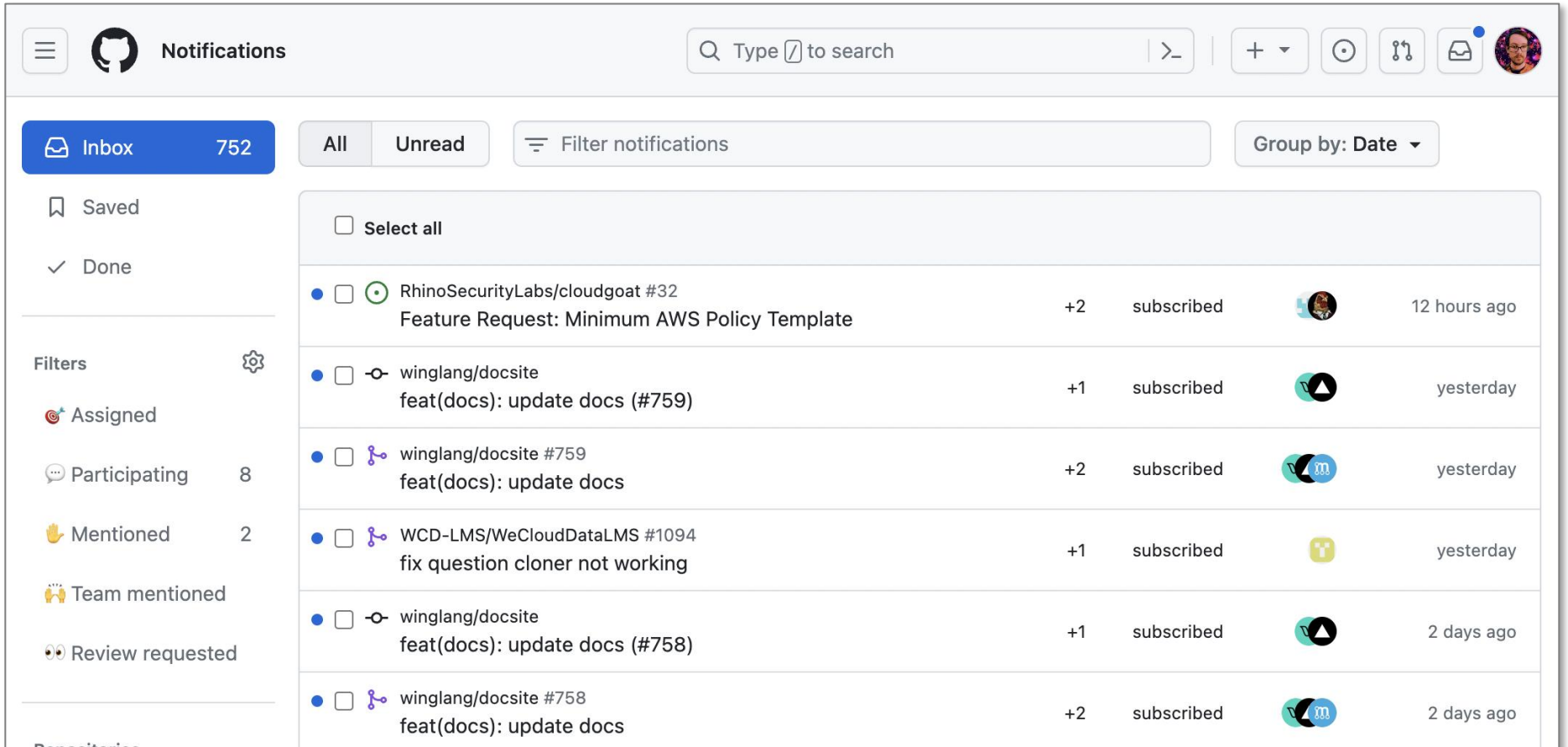




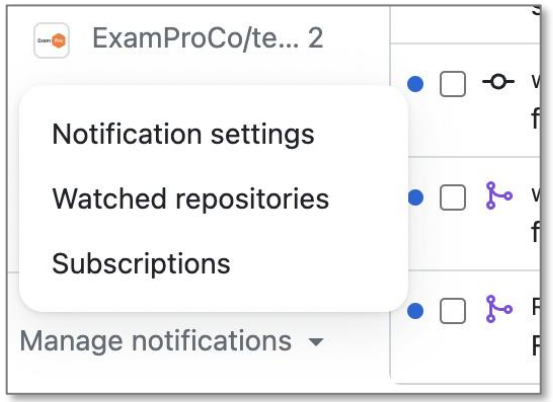
Notifications Inbox

The Notifications Inbox provides a chronological list of notifications threads.
This allows you to keep on top of all subscribed activities with associated GitHub repos.

https://github.com/notifications

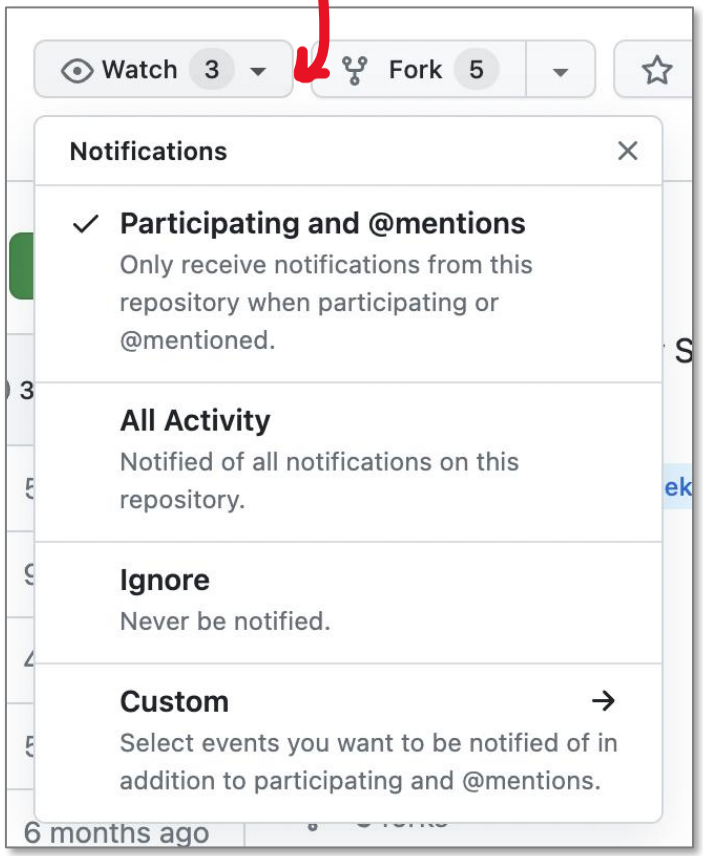


In the far bottom right, we can access other **Notifications views:**



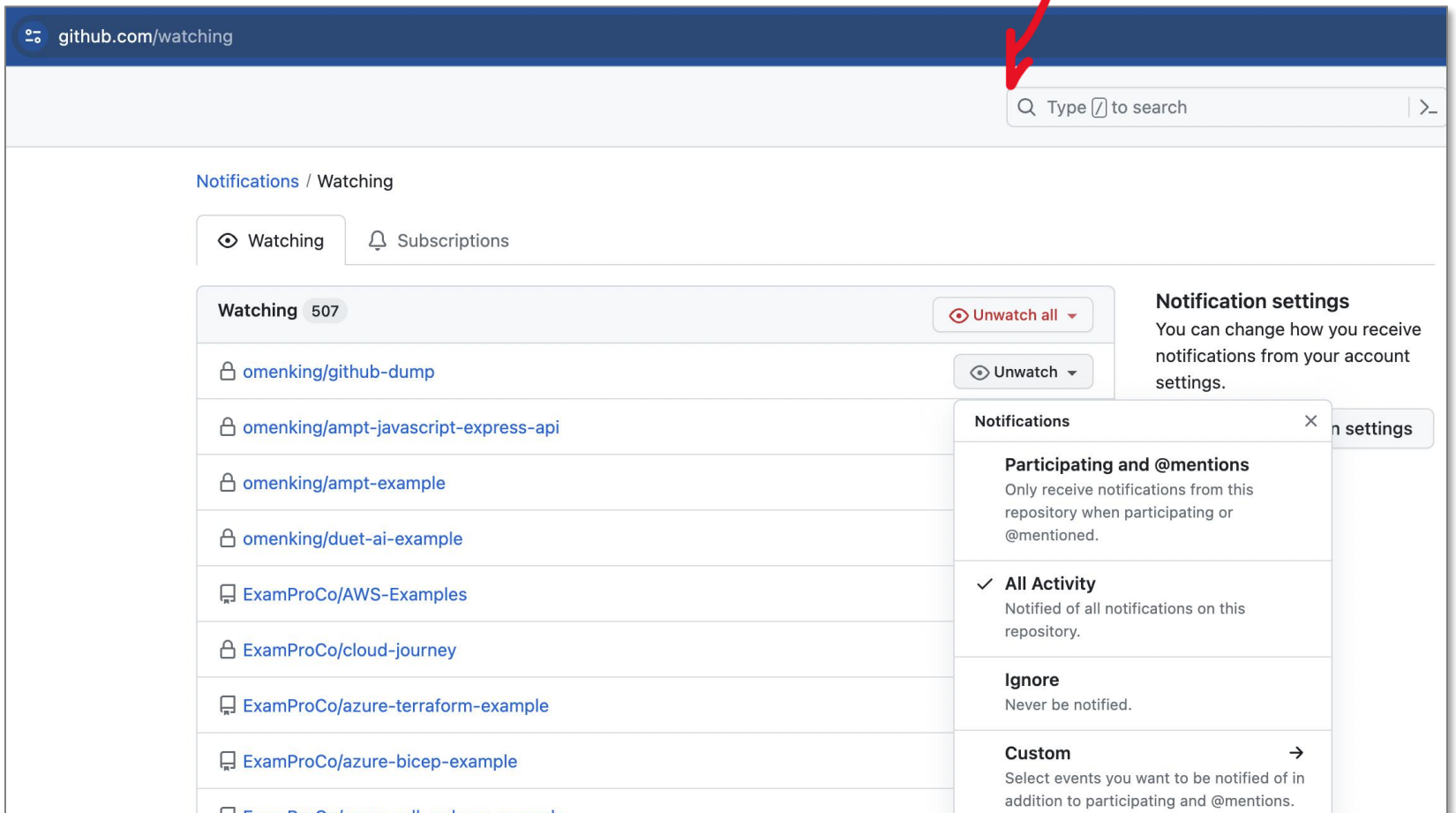
Notification Watching

You can “watch” repos on GitHub to get notified of activities.
Your own repos you already set as watching.



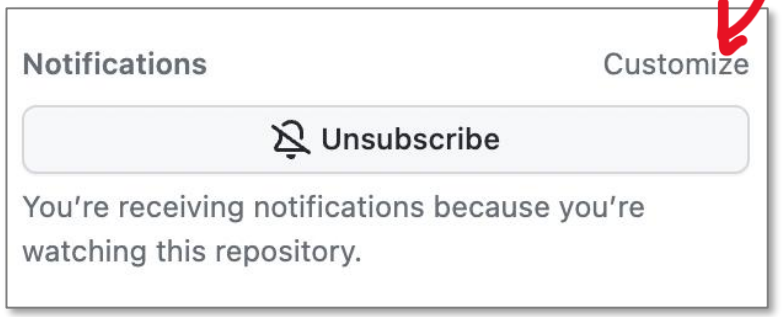
You can see a list of watching under notifications

https://github.com/watching



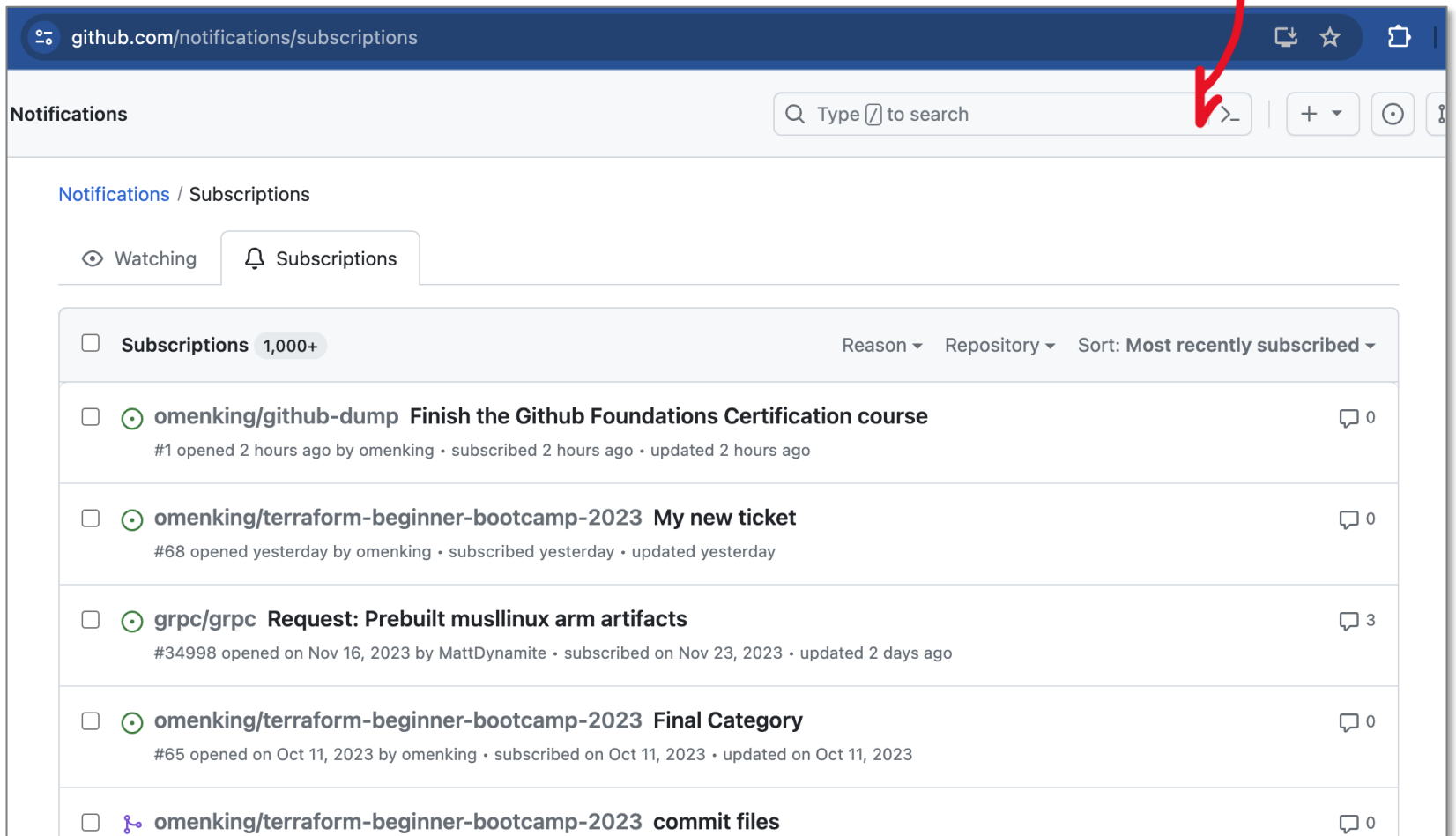
Notification Subscriptions

Subscriptions allow you to follow activities for a **specific Pull Request** or **Issue.**



You can see all your subscriptions under Notifications.

https://github.com/notifications/subscriptions

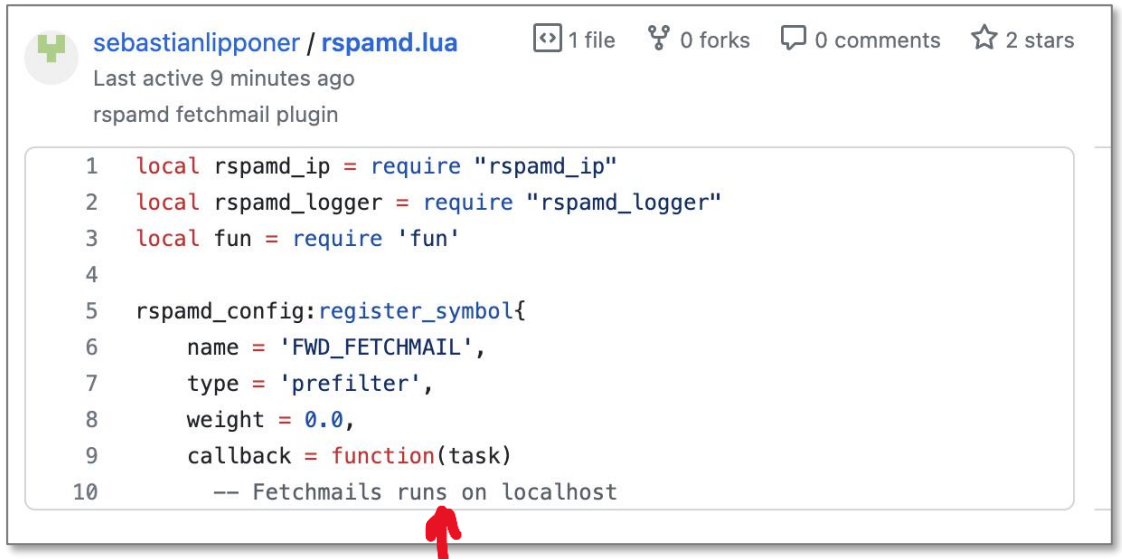


GitHub Gists

Gists provide a **simple way to share code snippets with others**

Every gist is a Git repository, which means that it can be **forked** and **cloned**.

https://gist.github.com



A **public gist** on gist.github.com

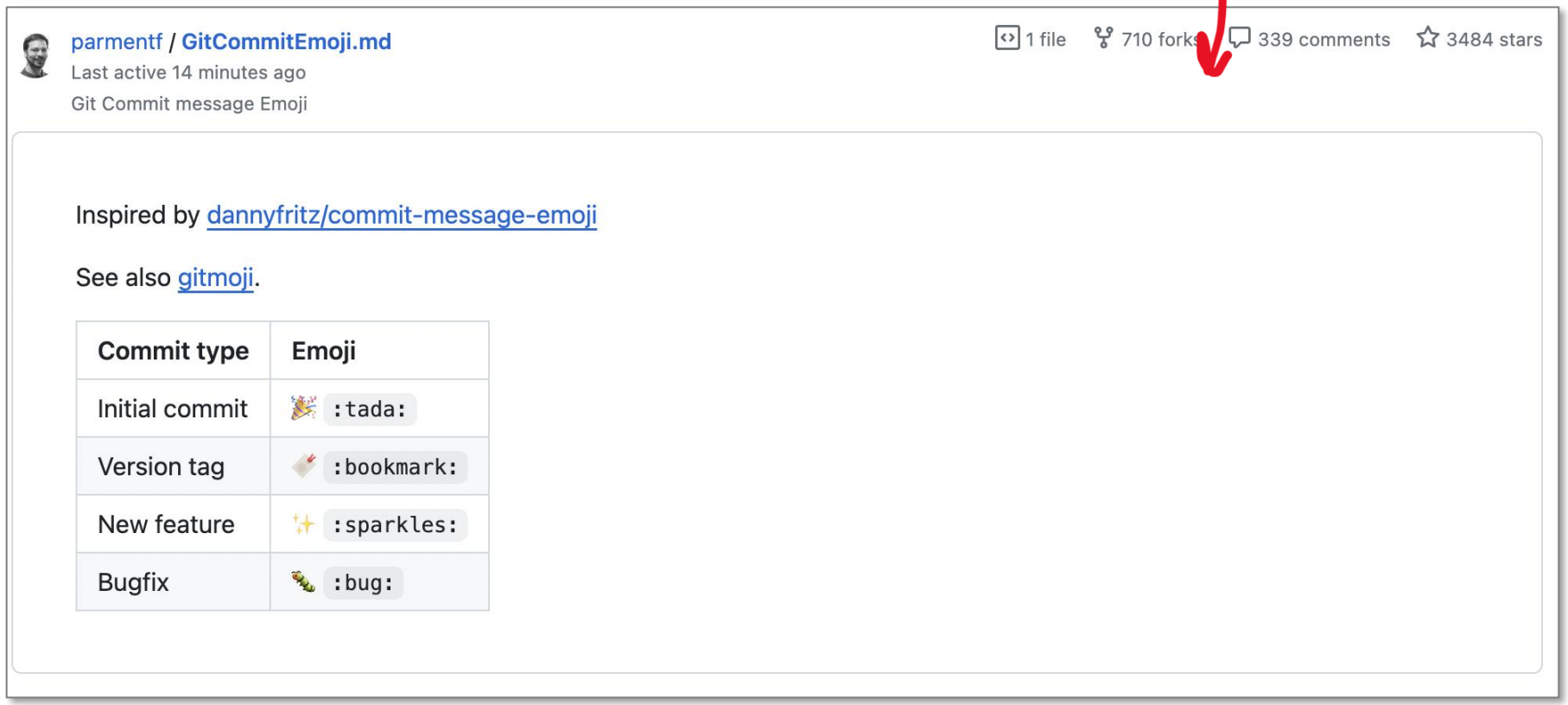
Gists can be **public** or **secret**.

- Public Gists are searchable
- Public Gists show up in Discover
  - https://gist.github.com/discover
- Secret gists are not searchable and do not show up in discovery
- Secrets gists are not private
  - You can share the URL with friends
- Secret Gists can be changed to be public
- Public Gists cannot be changed to be private
- You can pin Gists to your profile for other to easily find

- Gists allow other users to comment
- Gists allow you to easily navigation revisions

Creating GitHub Gists

If a Gist is a **markdown file** It will render the HTML.
This is a great way to share one-off documentation



GitHub Gists

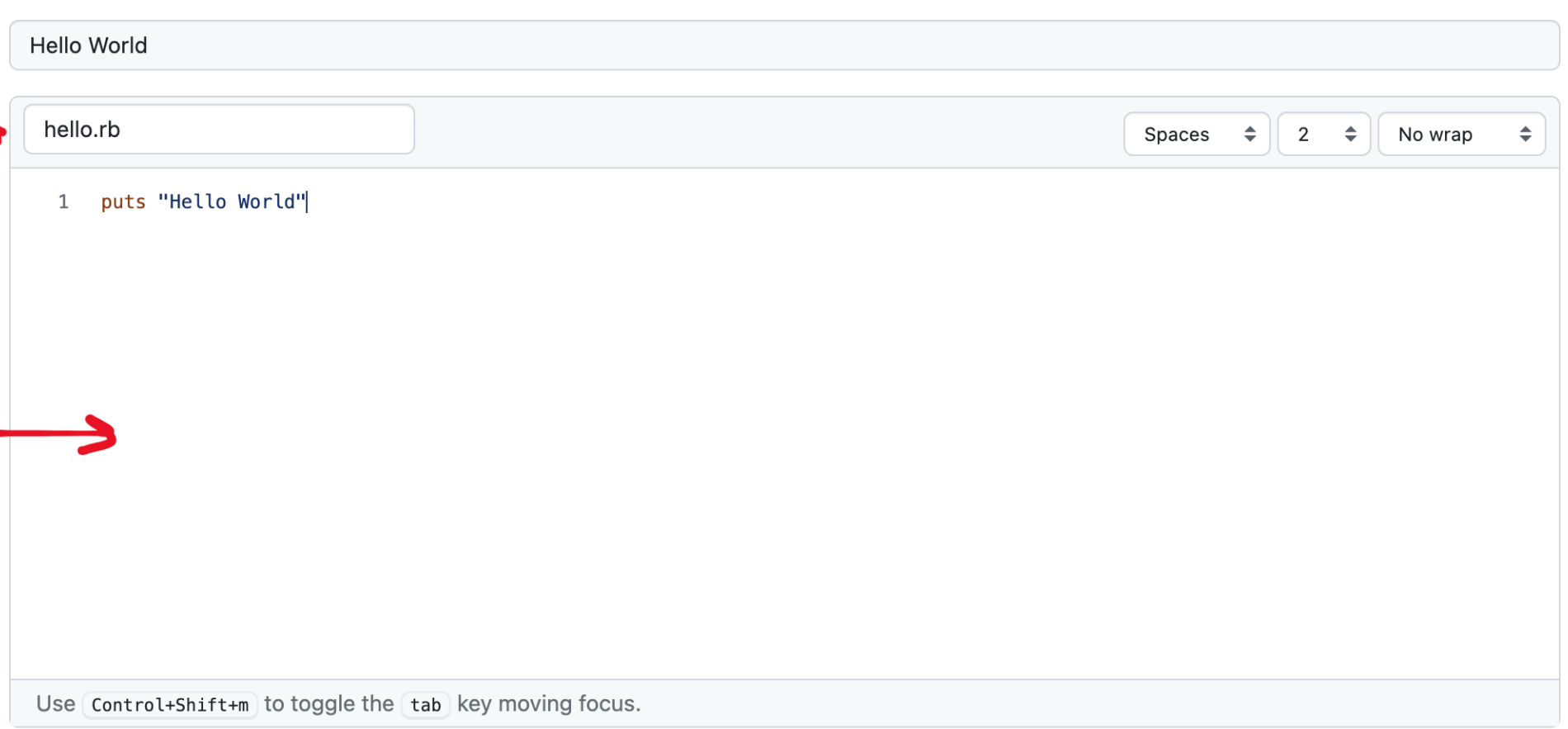
Create a Gist is a straightforward.

Name it:

Provide the **filename** with extension.

- The file extension will allow the file to apply syntax highlighting

Provide your **code**

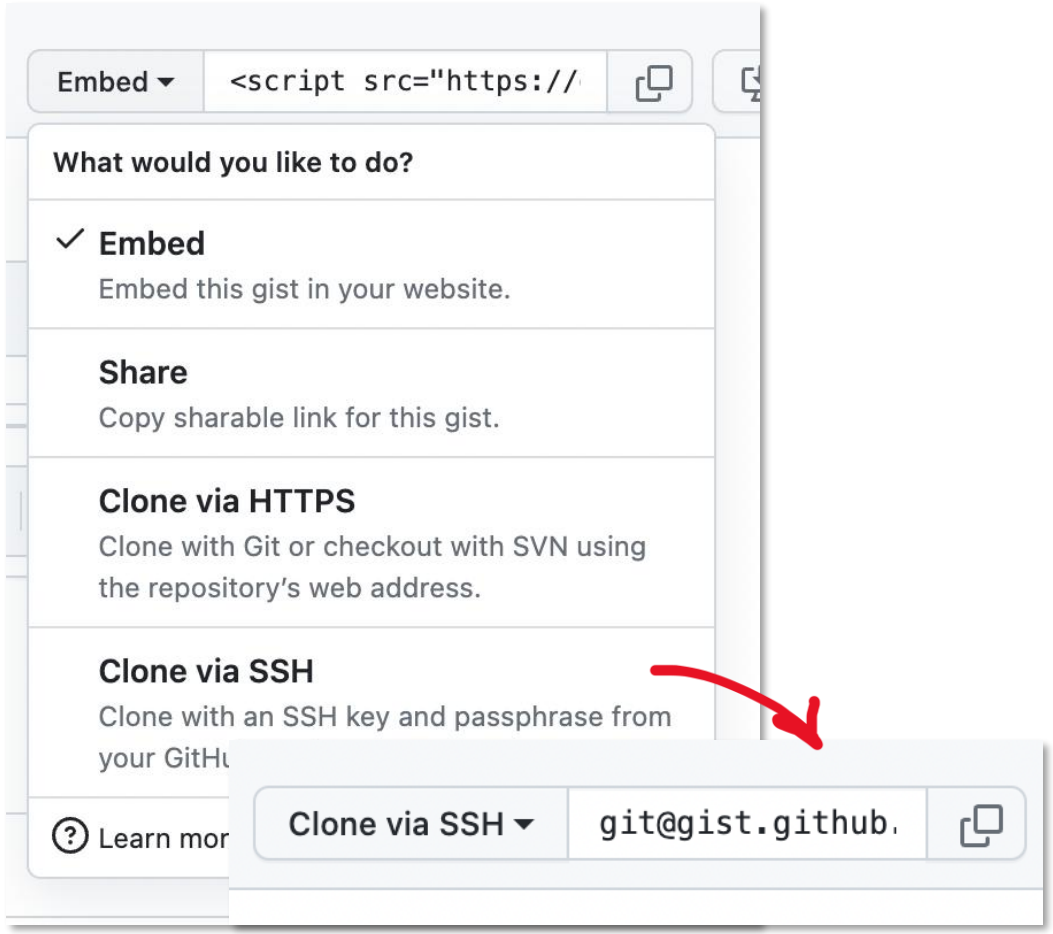


Attach associate files to the gist

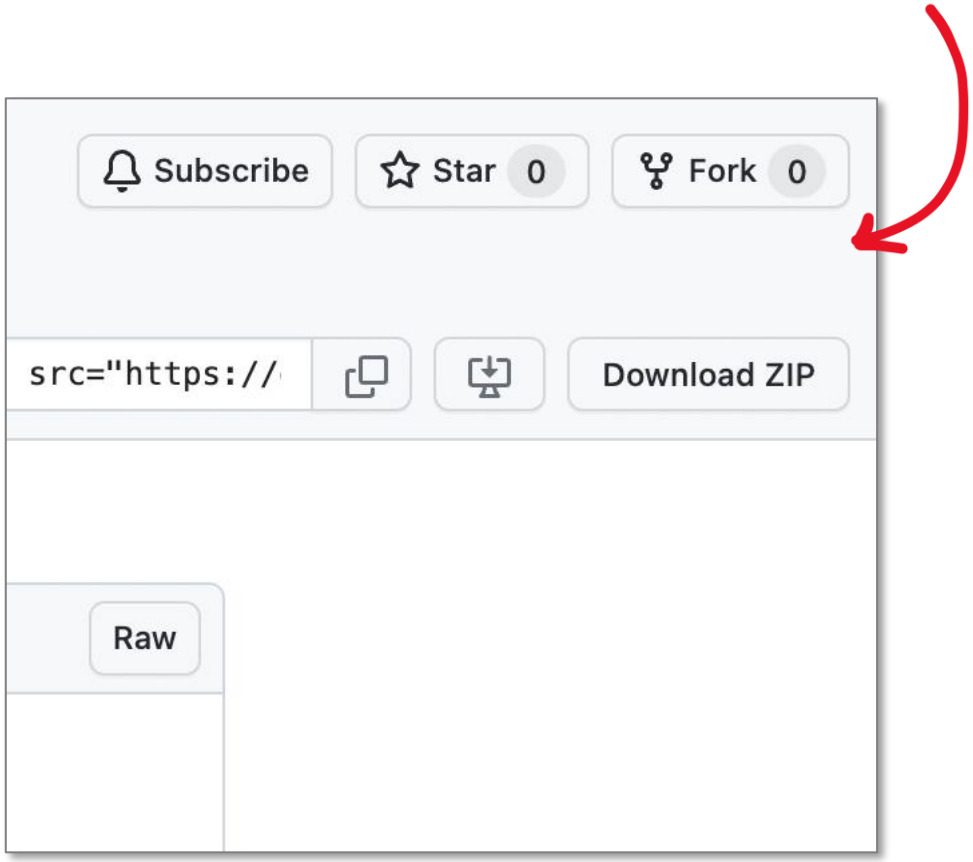
Set it as either **secret** or **private**

Fork and Clone Gists

You can **clone URL** by dropping down embed



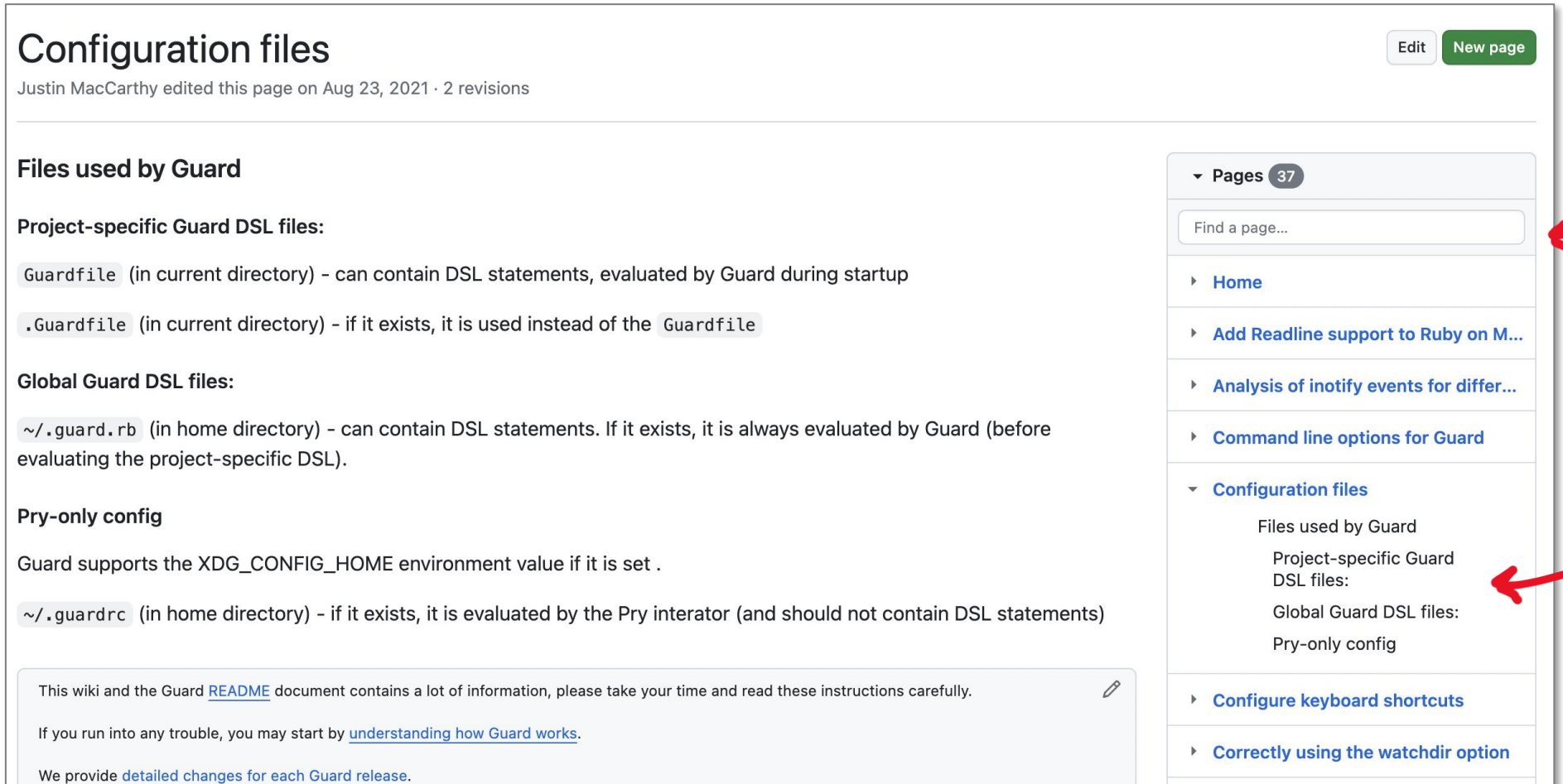
For Gist repos other than your own you **fork** them.



GitHub Wiki Pages

A **Wiki** is collaborative way to quickly **create documentation with multiple nested pages**

A wiki would serve as a **knowledge base** for your repo or organization.

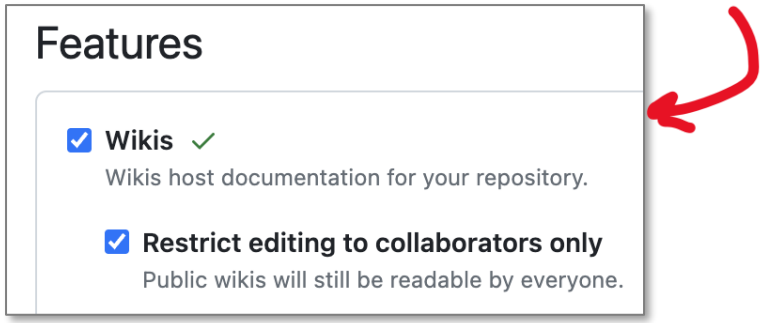


Search pages

Nested pages

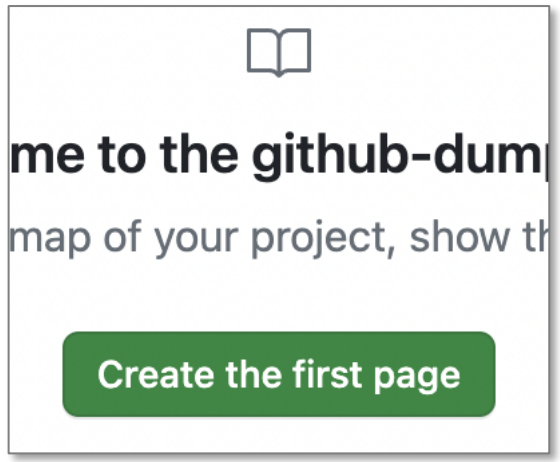
GitHub Wiki Pages

Turn **on the Wiki feature** for your GitHub Repo.



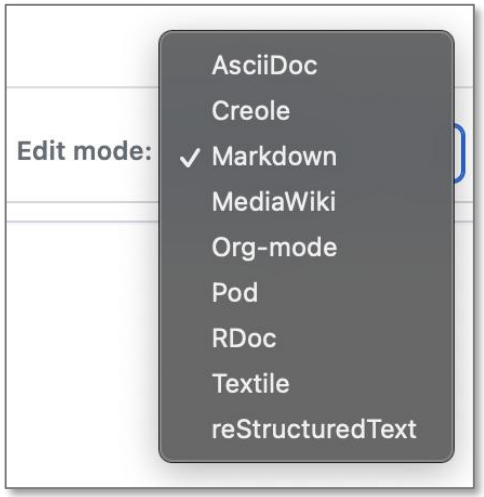
Go to your **Wiki page** of your Repo

Create your **first page!**



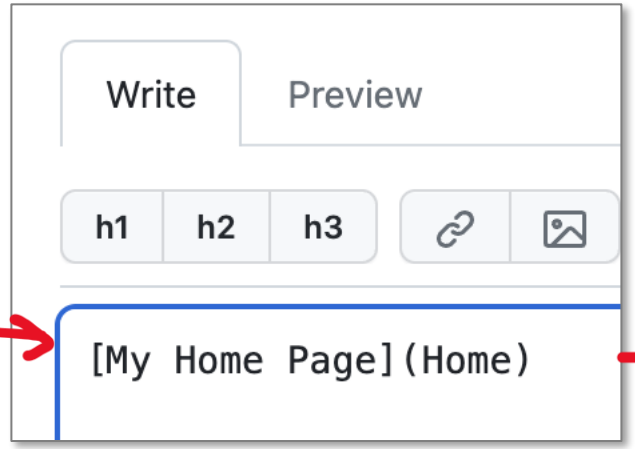
GitHub Wiki's can be used for free on Public repos.

Managing GitHub Wiki Pages



When you create a page, you can use **various markup languages**.

To **link to other pages,** you provide the GitHub Wiki pages title as the link



GitHub Wiki pages visibility

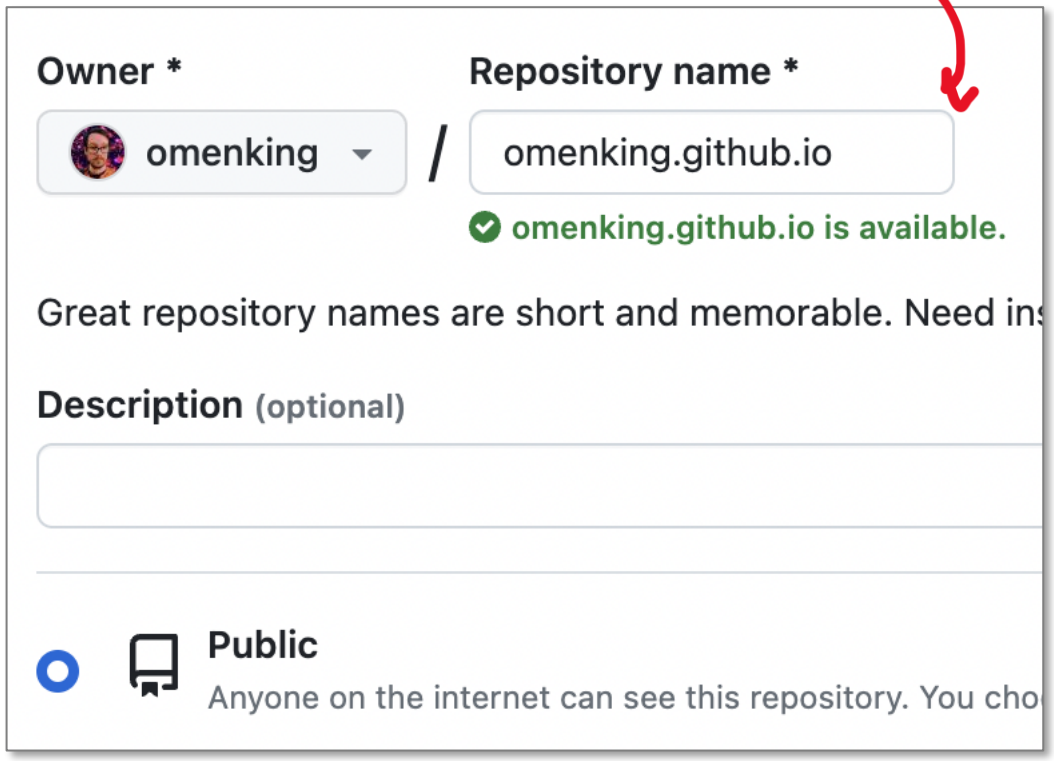
If you create a wiki in a **public repository**, the wiki is available to the public.

If you create a wiki in a **private repository**, only people with access to the repository can access the wiki.

GitHub Pages

GitHub Pages allows you to directly host a static website via a GitHub repository.

You need to create a **public repo** with the repo name being **<username>.github.io**



You simply **add HTML files and assets** to your repo.

Jekyll is static site generator which allows you to build complex static sites or blogs with ease.

- Jekyll uses yaml files with frontmatter format.
- Jekyll utilizes markdown files
- Jekyll directly integrates with GitHub Pages and is a GitHub project

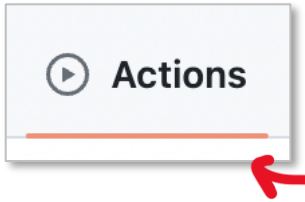
Your website will be available the domain name: **<username.github.io>**

Introduction to GitHub Actions

GitHub Actions is a **CI/CD pipeline directly integrated with your GitHub repository.**

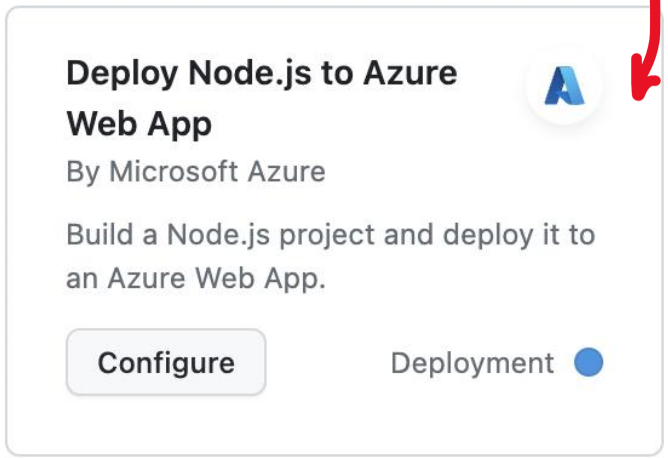
GitHub Actions allows you to **automate**:

- Running test suites
- Building images
- Compiling static sites
- Deploying code to servers
- and more…



Within a GitHub repo you'll have a **tab for Actions**.

GitHub Actions has **templates** you can use to get started.



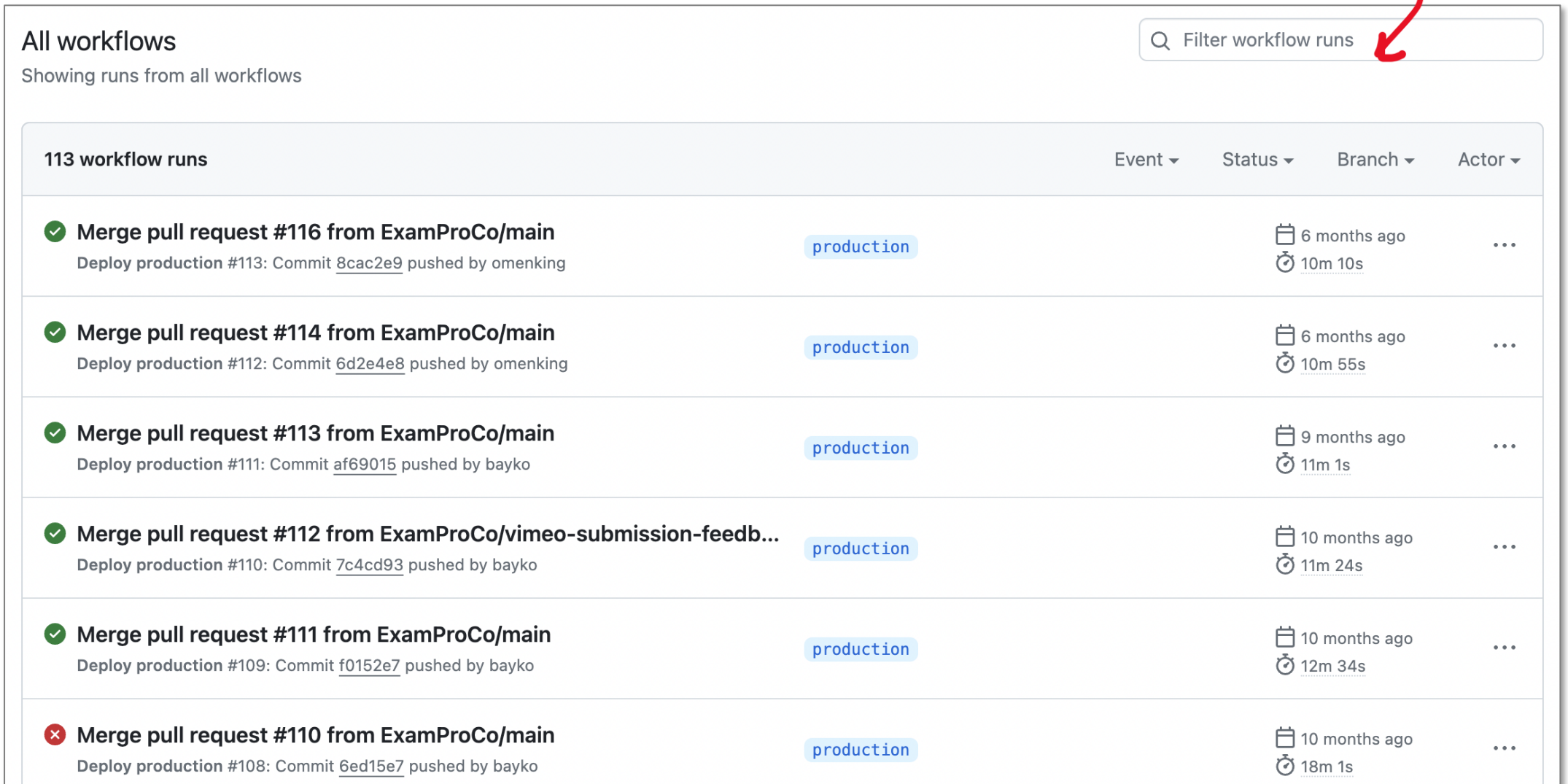
GitHub Actions files are defined as YAML files located in the **.github/workflow** folder in your repo.

You can multiple workflows in repo triggered by different events.

```yaml
on:
  push:
    branches: [ $default-branch ]
  workflow_dispatch:
env:
  AZURE_WEBAPP_NAME: your-app-name
  AZURE_WEBAPP_PACKAGE_PATH: '.'
  NODE_VERSION: '14.x'
permissions:
  contents: read
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v3
    ...
```

Introduction to GitHub Actions

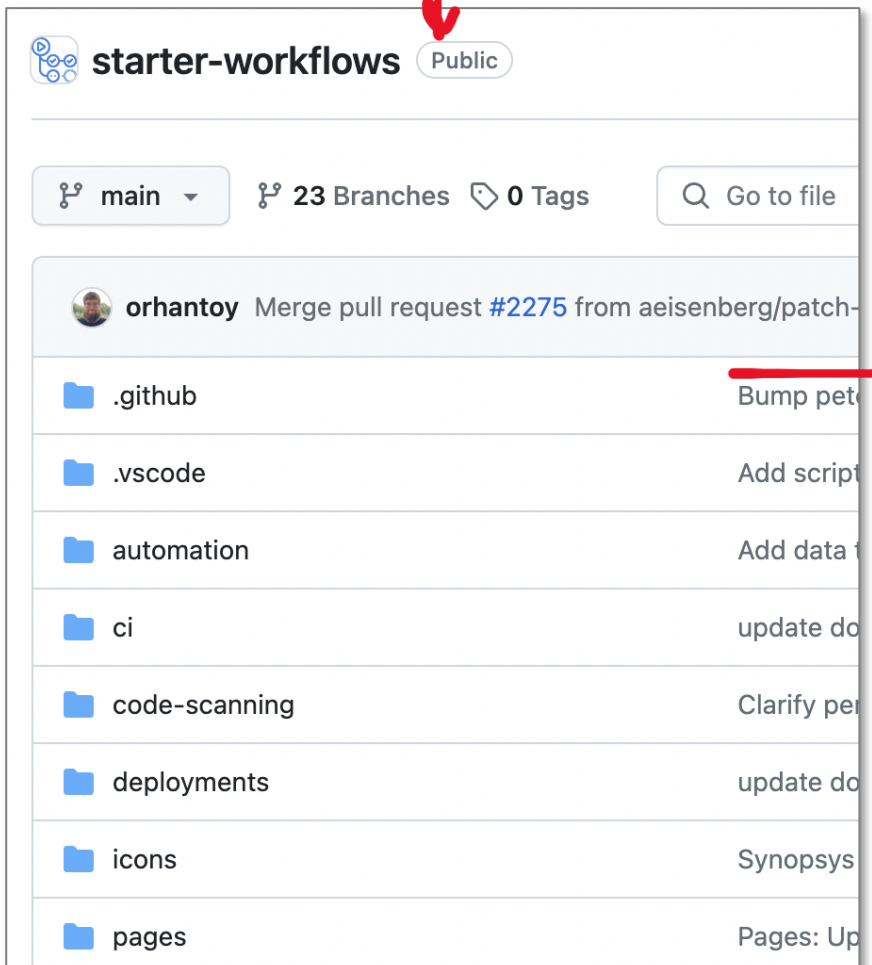
When you run GitHub Actions, you'll get **a history of workflow runs** where it will indicate if it was success a failure, and how long it took to run.

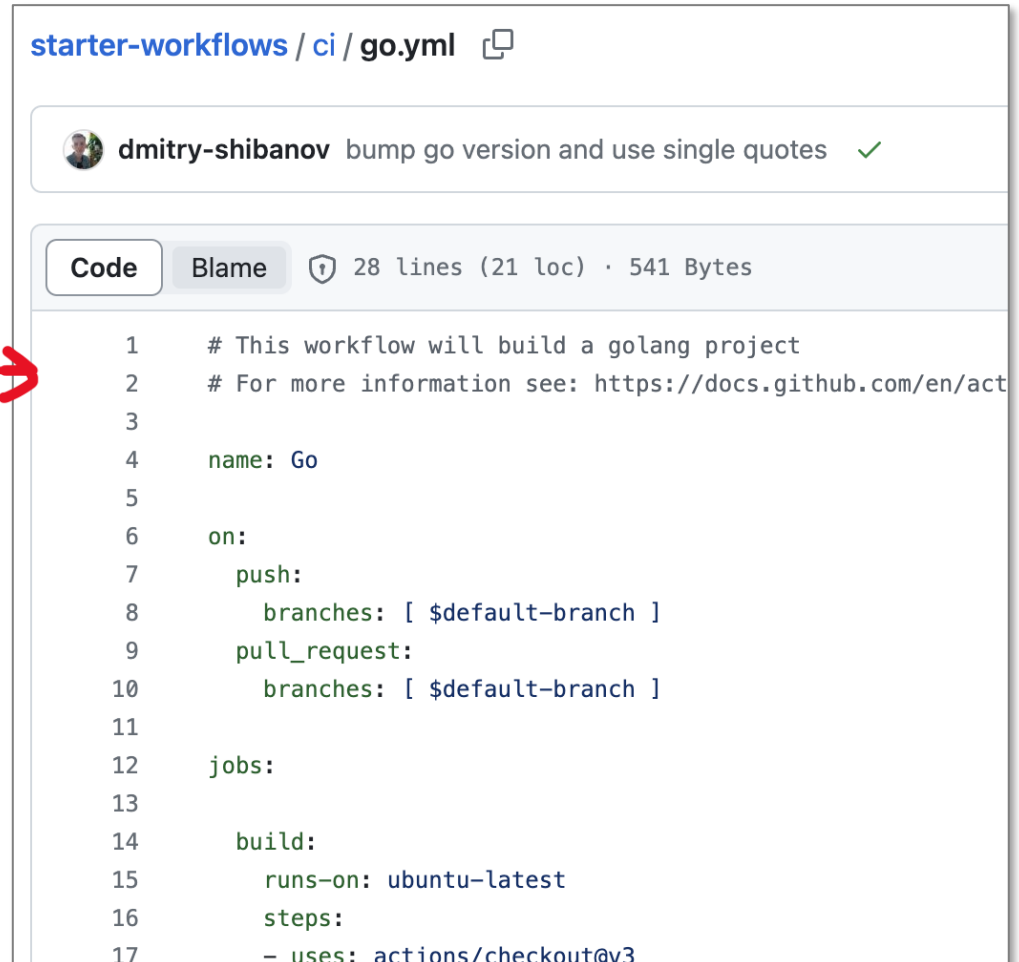


Finding GitHub Actions

GitHub has a repo of example workflows you can use to get you started

https://github.com/actions/starter-workflows





Different Types of GitHub Actions

Event Triggers causes a GitHub Action to run.

The **on** attribute specifies
the **event trigger** to be used:

```yaml
name: Example Workflow
on: [push]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v2
    - name: Run a script
      run: echo "This script runs on every push!"
```

GitHub Actions has 35+ event triggers

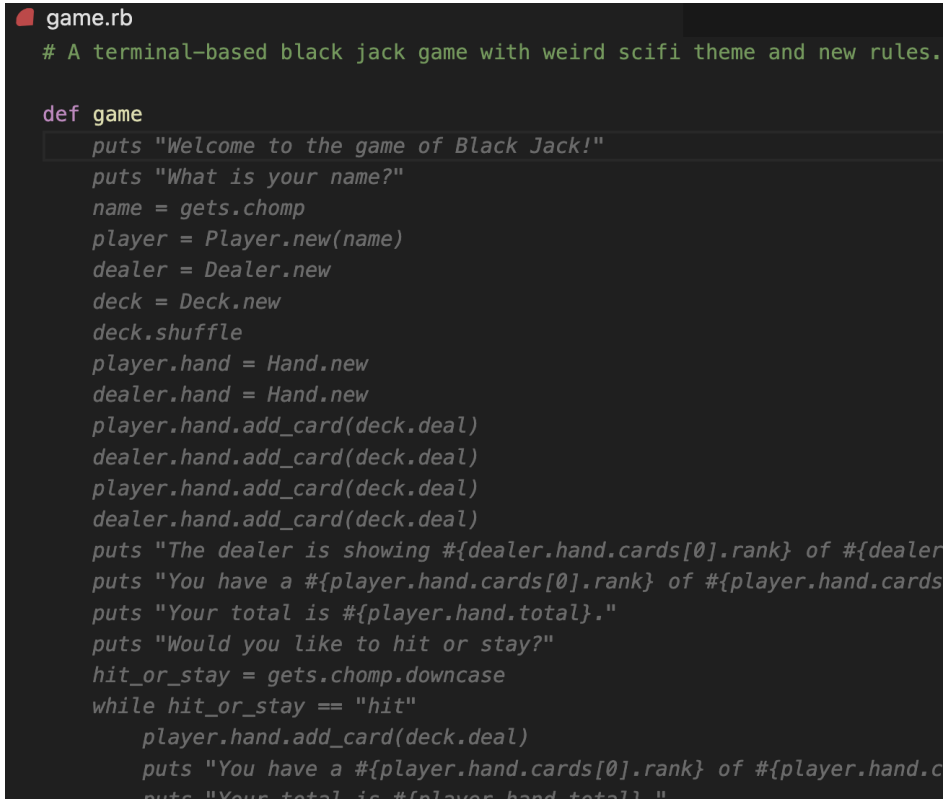
Examples of common GitHub Actions that could be triggered:

- **Pushes:** Trigger an action on any push to the repository.
- **Pull Requests:** Run actions when pull requests are opened, updated, or merged.
- **Issues:** Execute actions based on issue activities, like creation or labeling.
- **Releases:** Automate workflows when a new release is published.
- **Scheduled Events:** Schedule actions to run at specific times.
- **Manual Triggers:** Allow manual triggering of actions through the GitHub UI.

Introduction to GitHub Copilot

GitHub Copilot is **AI developer tool** that can be used with multiple IDEs via an extension

- **Code Completion**
- Chat in IDE and Mobile
- CLI assistance
- Code referencing
- Public code filter
- IP indemnity



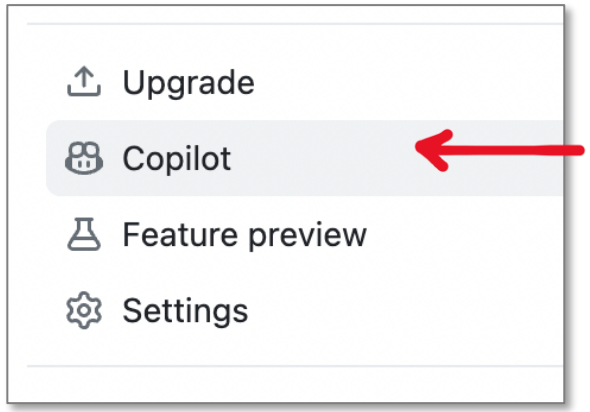
FREE TRIAL

You can try GitHub Copilot for free with a one-time 30-day trial.
GitHub Copilot is free to use for verified students, teachers, and maintainers of popular open source projects.

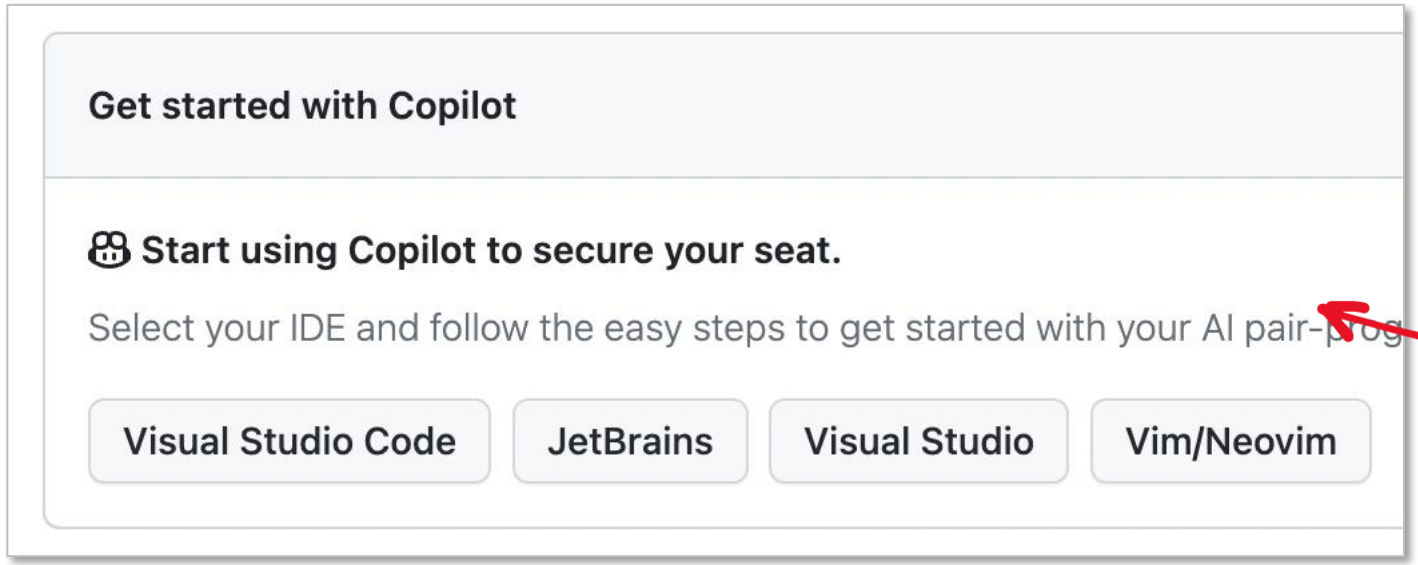
GitHub Copilot Individuals vs Business

	Copilot Individuals	Copilot Business
Pricing	$10 USD per month $100 USD per year	$19 USD per user per month
Types of GitHub accounts	Personal accounts	Organization or enterprise accounts
Telemetry	✓	✗
Blocks suggestions matching public code	✓	✓
Plugs right into your editor	✓	✓
Offers multi-line function suggestions	✓	✓
Organization-wide policy management	✗	✓
Exclude specified files	✗	✓
Audit logs	✗	✓
HTTP proxy support via custom certificates	✗	✓

Setting up Copilot

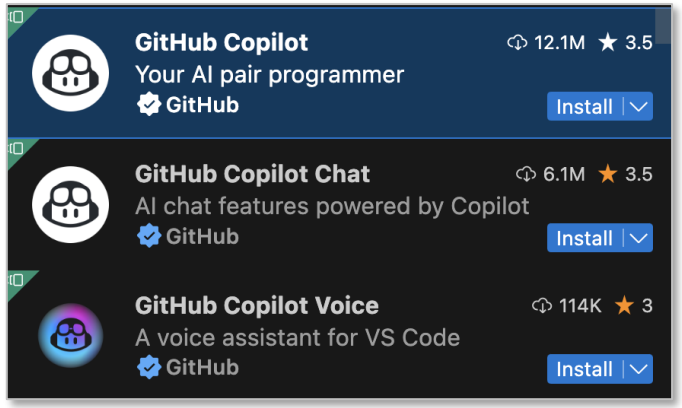


Under User Account dropdown there is a **Copilot** option.

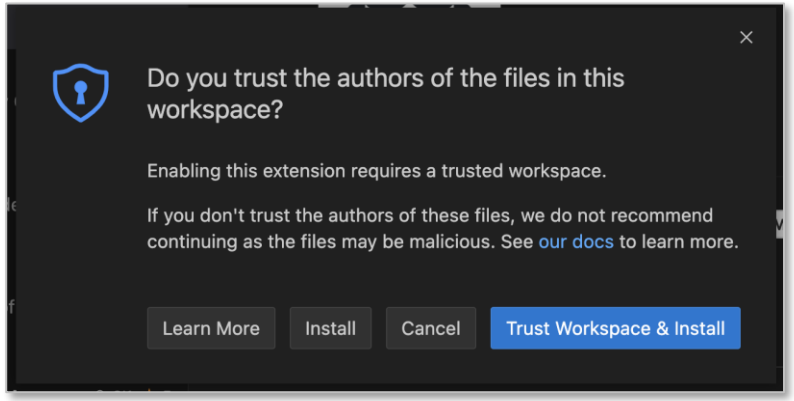


GitHub has various instructions to get started with GitHub Copilot based on your IDE.

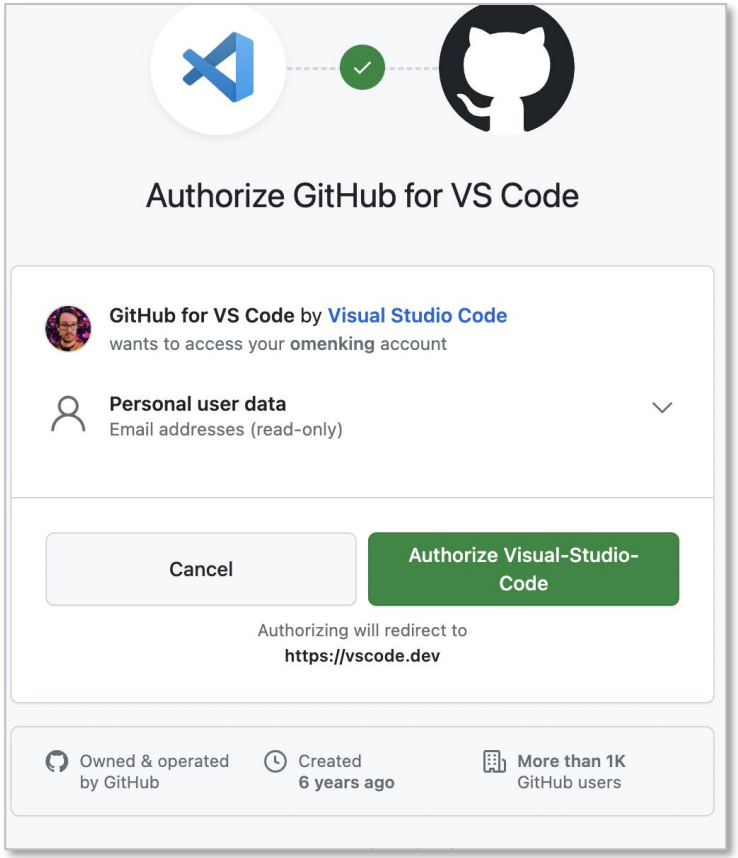
Setting up Copilot

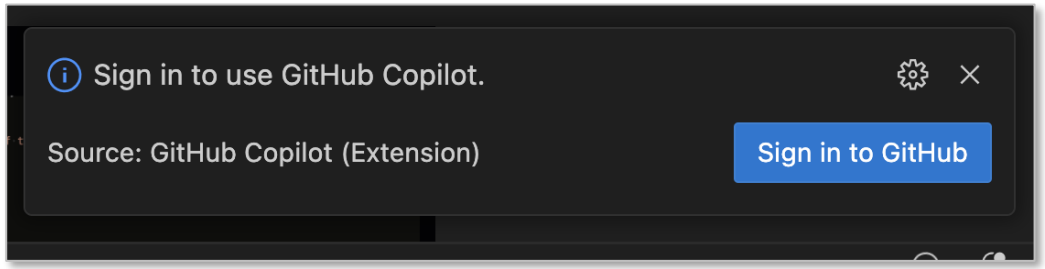


Install the **GitHub Copilot extension** in your VS Code



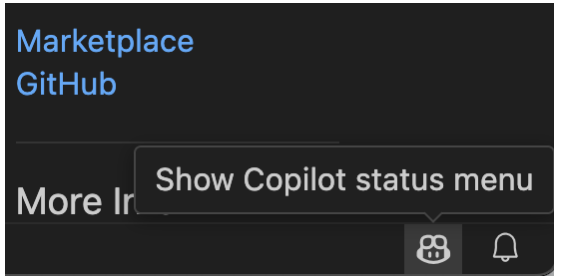
Choose **Trust Workspace & Install**





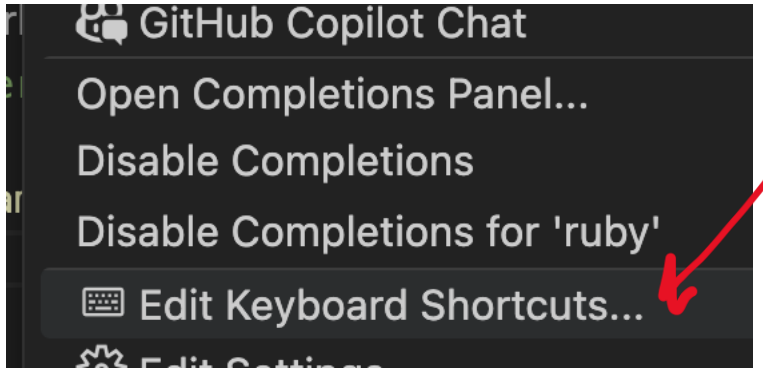
Sign in to GitHub

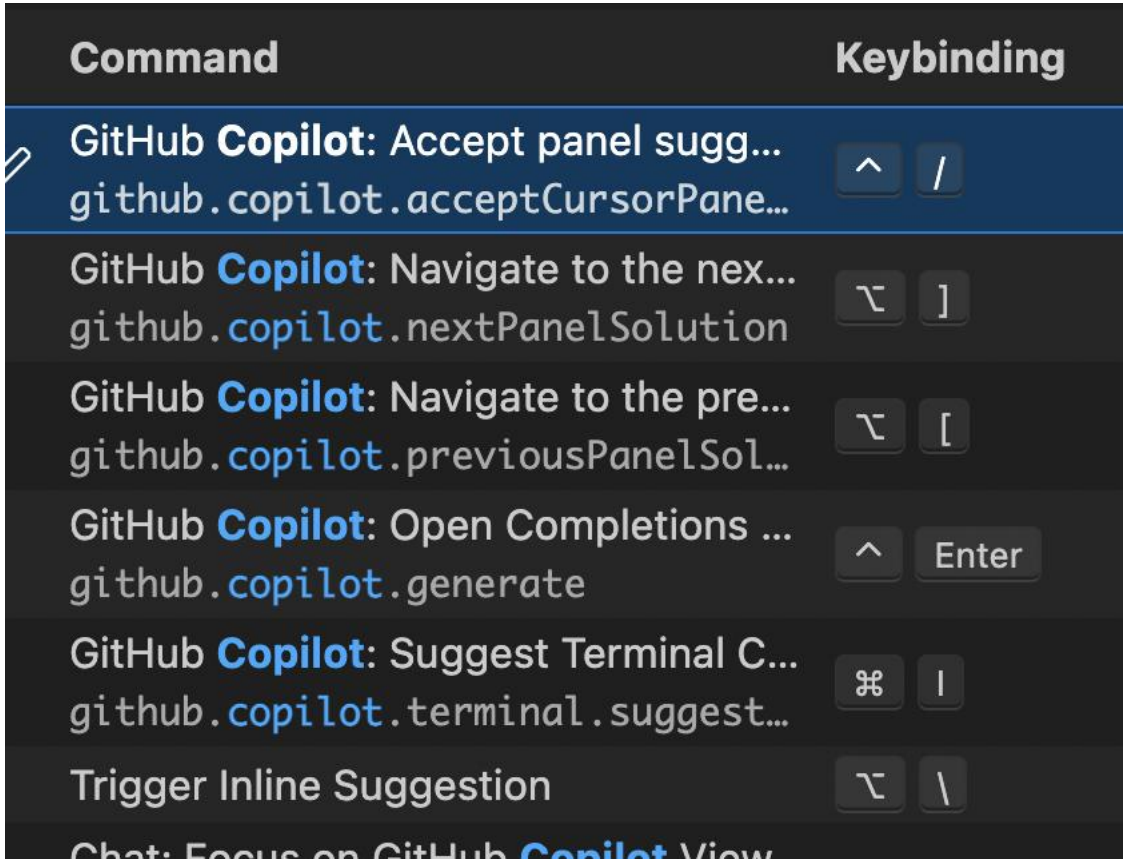
Copilot will show up in **bottom right corner**



Using Copilot

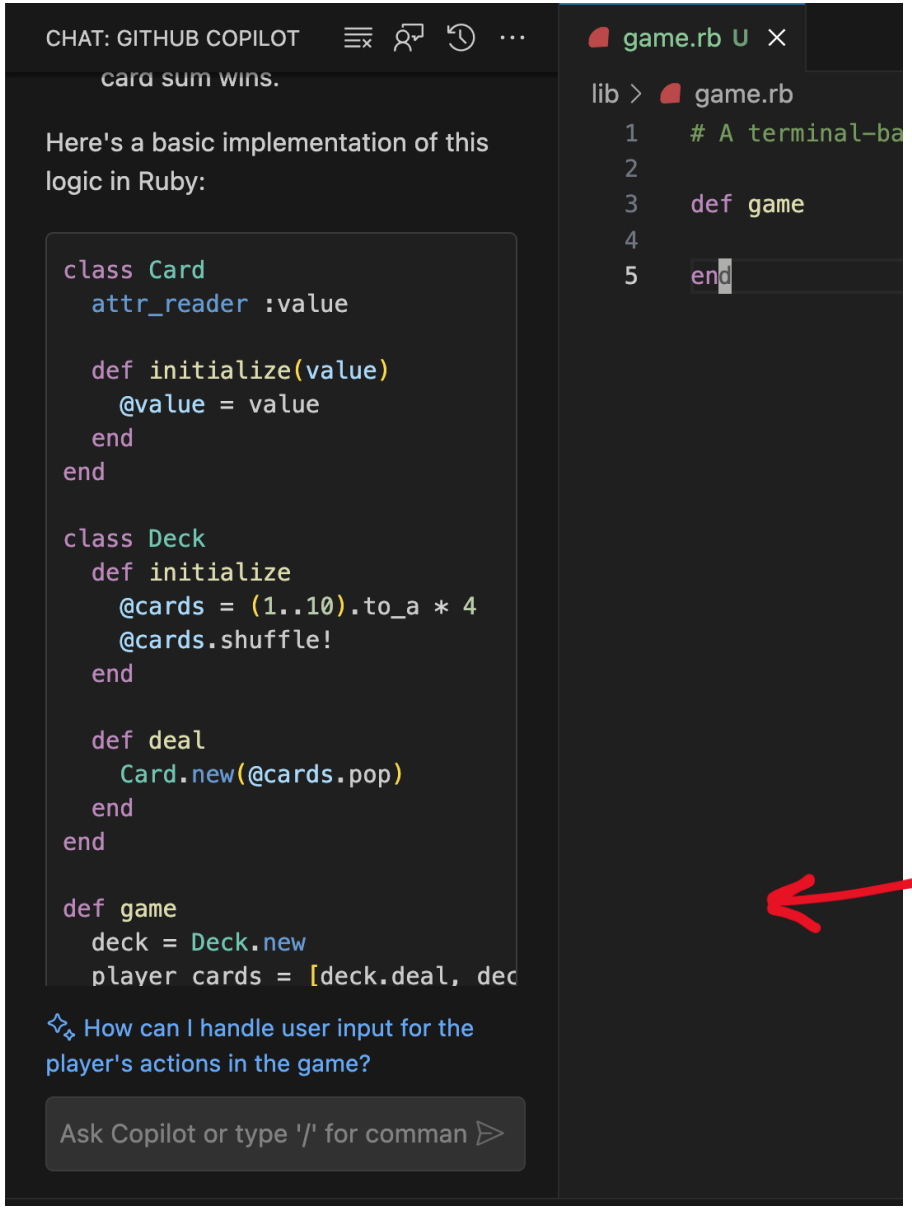
Copilot is best used when you **know or set hotkeys** to perform suggestions



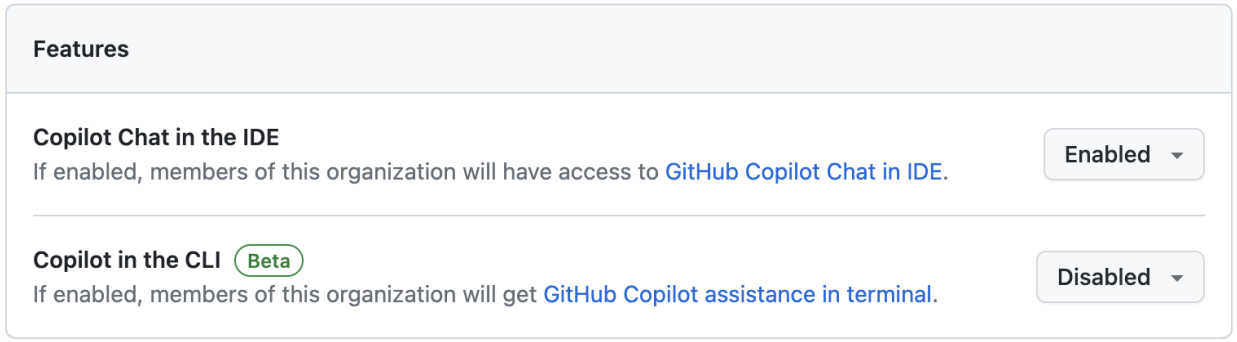


Its best to check what hotkeys you have since they may vary based on the documentation stated hotkeys

Using Copilot Chat



You to have **GitHub Copilot Chat feature** enabled in your personal account or for your organization.



Once enabled you can
chat with Google Copilot

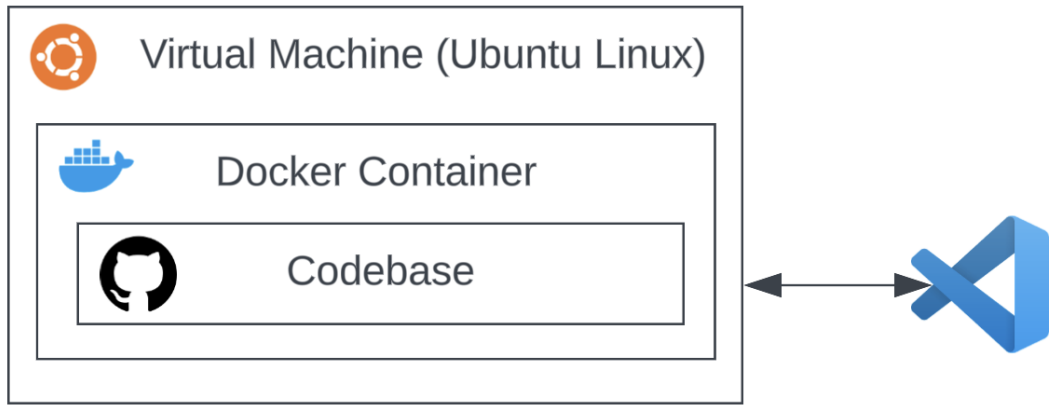
Introduction to GitHub Codespaces

GitHub Codespaces is **Cloud Developer Environment (CDE) integrated with you GitHub repo**.

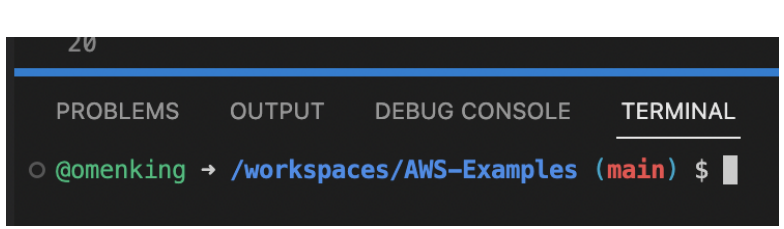
An environment is known as a **codespace.**

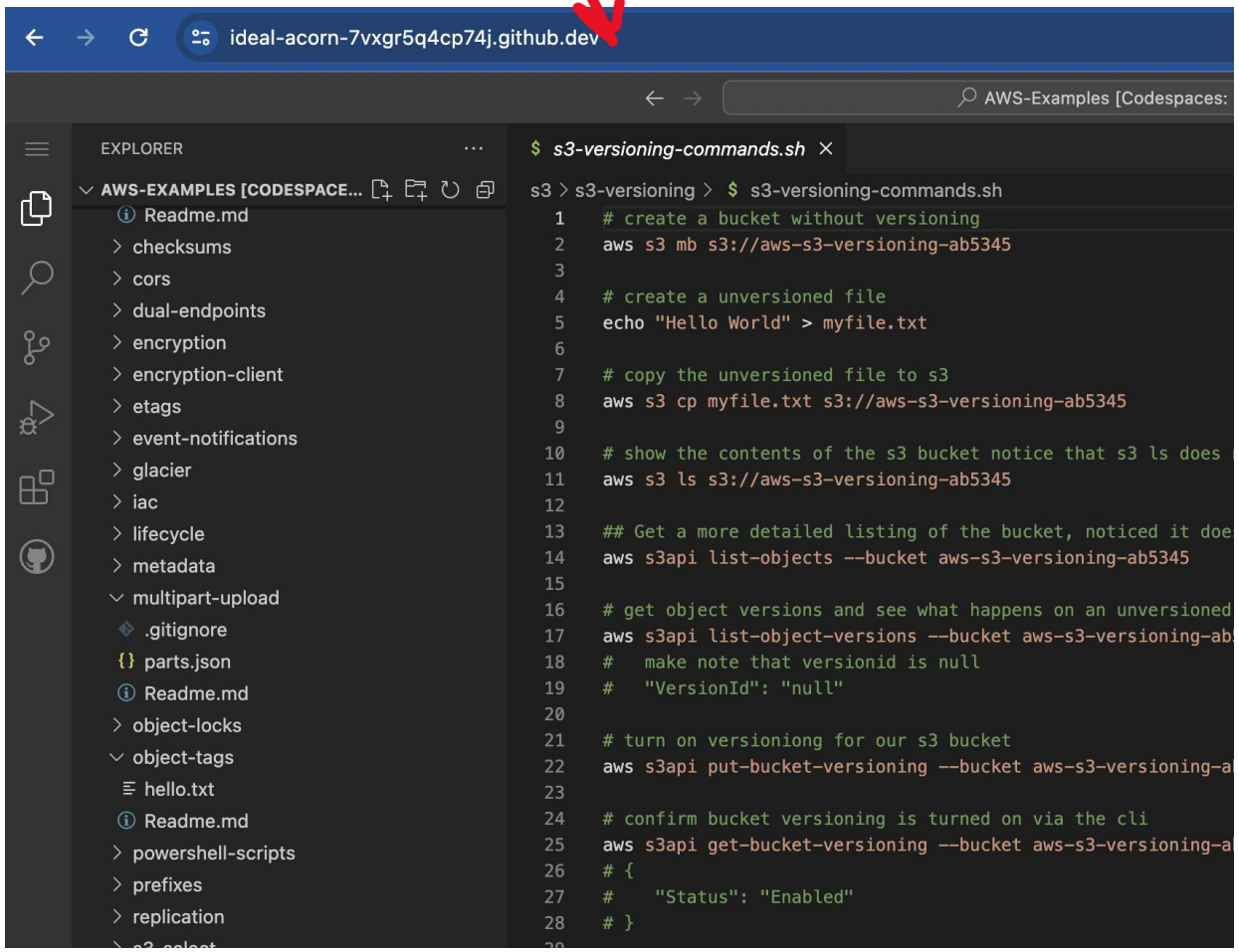
A codespace runs:

- in Ubuntu Linux docker container
  - on a virtual machine
    - hosted and managed by GitHub

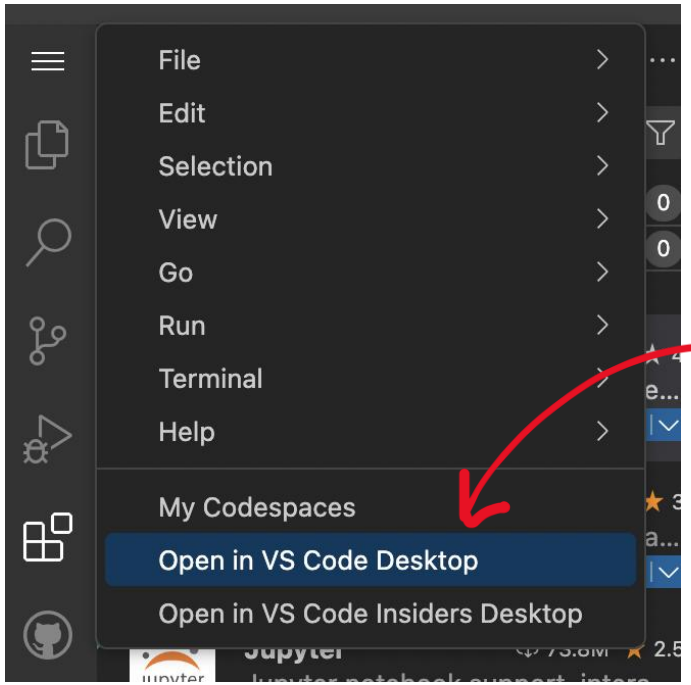


Your codebase will be cloned into **/workspaces**





Introduction to GitHub Codespaces



You can interact with your codespace via:

- Command Line
  - **SSH via GitHub CLI**
- Code Editors
  - VS Code Browser
  - **VS Code Desktop**
  - JetBrains IDE

In the VS Code Browser, you can tell it to connect this codespace To your locally installed VS Code

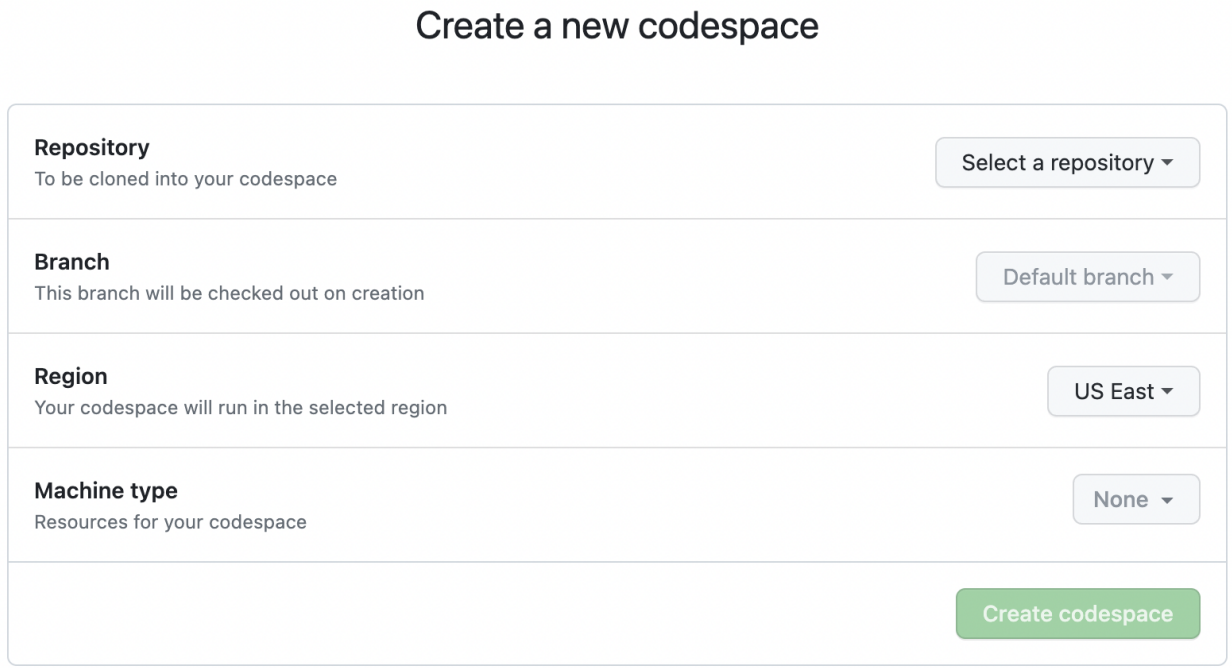
The GitHub CLI commands available for Codespaces:

- gh codespace code
- gh codespace cp
- gh codespace create
- gh codespace delete
- gh codespace edit
- gh codespace jupyter
- gh codespace list
- gh codespace logs
- gh codespace ports
- gh codespace rebuild
- gh codespace ssh
- gh codespace stop
- gh codespace view

Introduction to GitHub Codespaces

When you create a new codespace you **choose**:

- The repo
- The branch
- The Region
- Machine type



The **regions** you can choose:

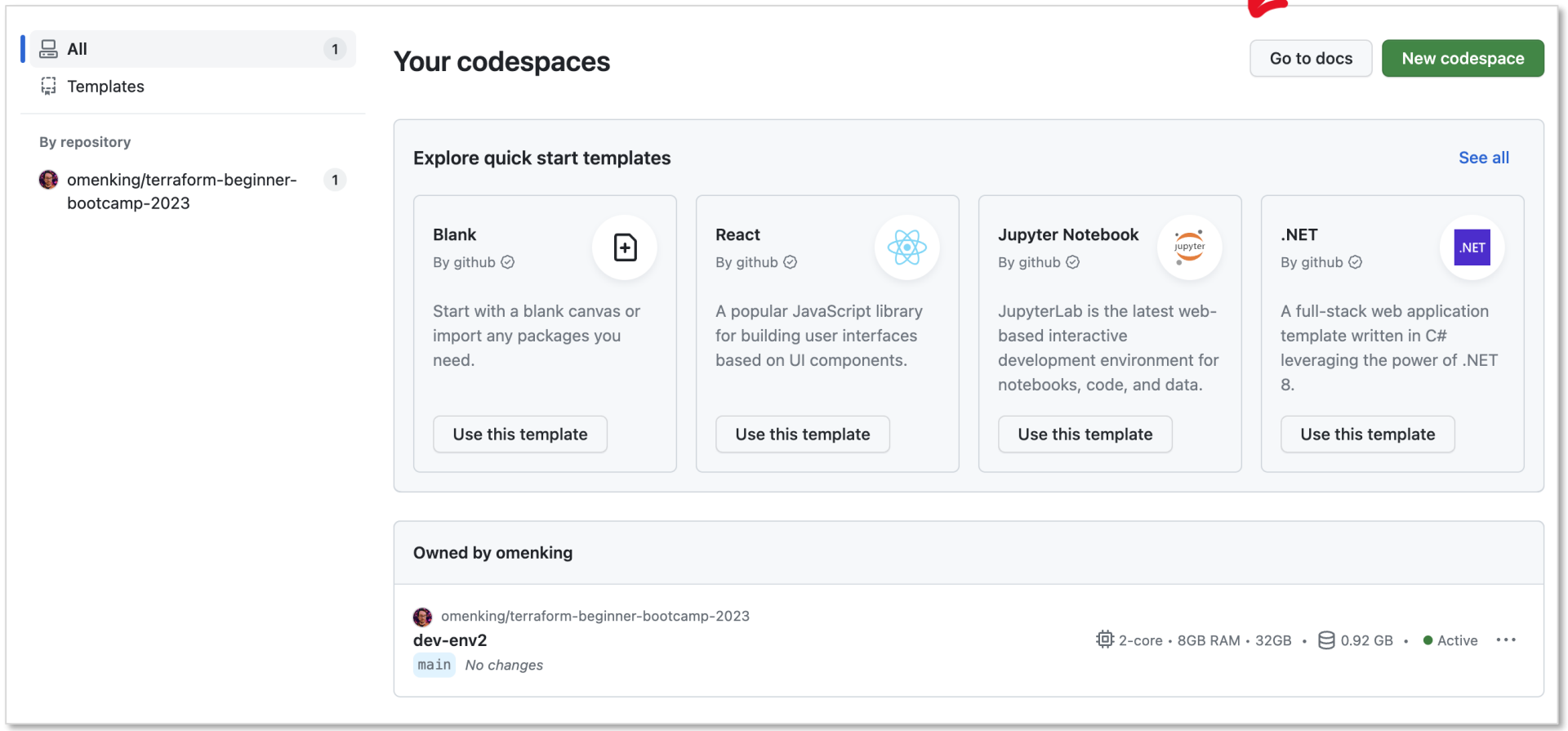
- US East
- US West
- Europe West
- Southeast Asia
- Australias

The capacity of your environment is based on the virtual **machine type** you choose:

- 2-Core (8GB RAM, 32 GB)
- 4-Core (16GB RAM, 32 GB)
- 8-Core (32GB RAM, 64 GB)
- 16-Core (32GB RAM, 128 GB)

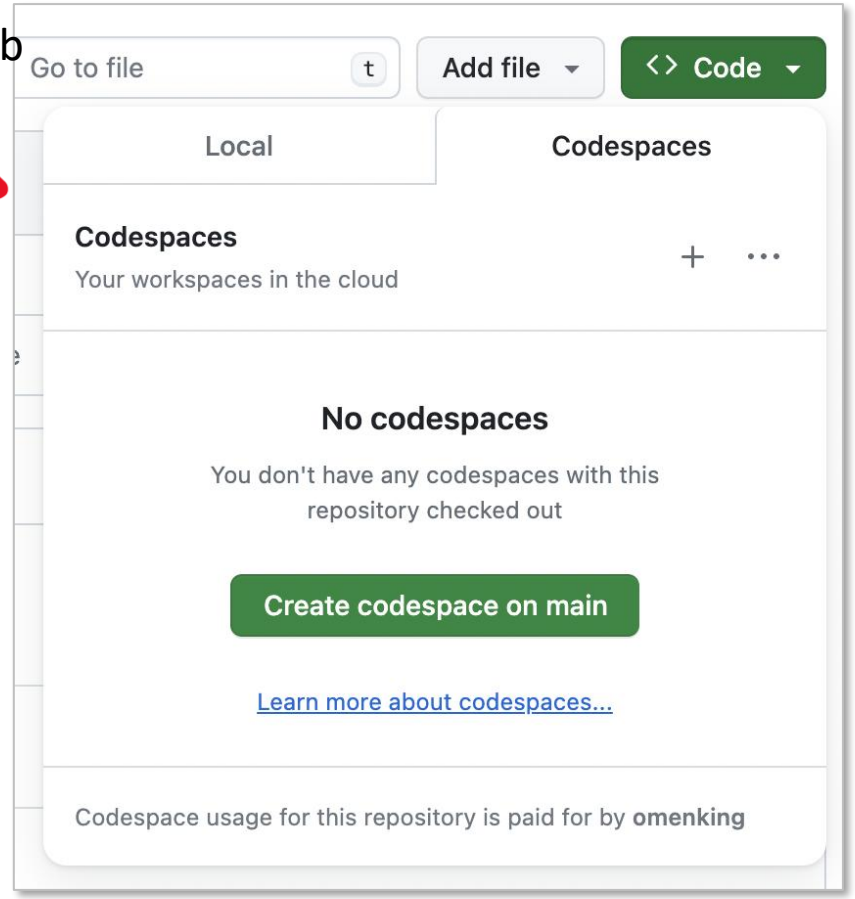
Introduction to GitHub Codespaces

You can see all of your codespaces across all your repos at **https://github.com/codespaces**



Starting GitHub Codespaces

You can start a Codespace via your GitHub repo via the **Code** button

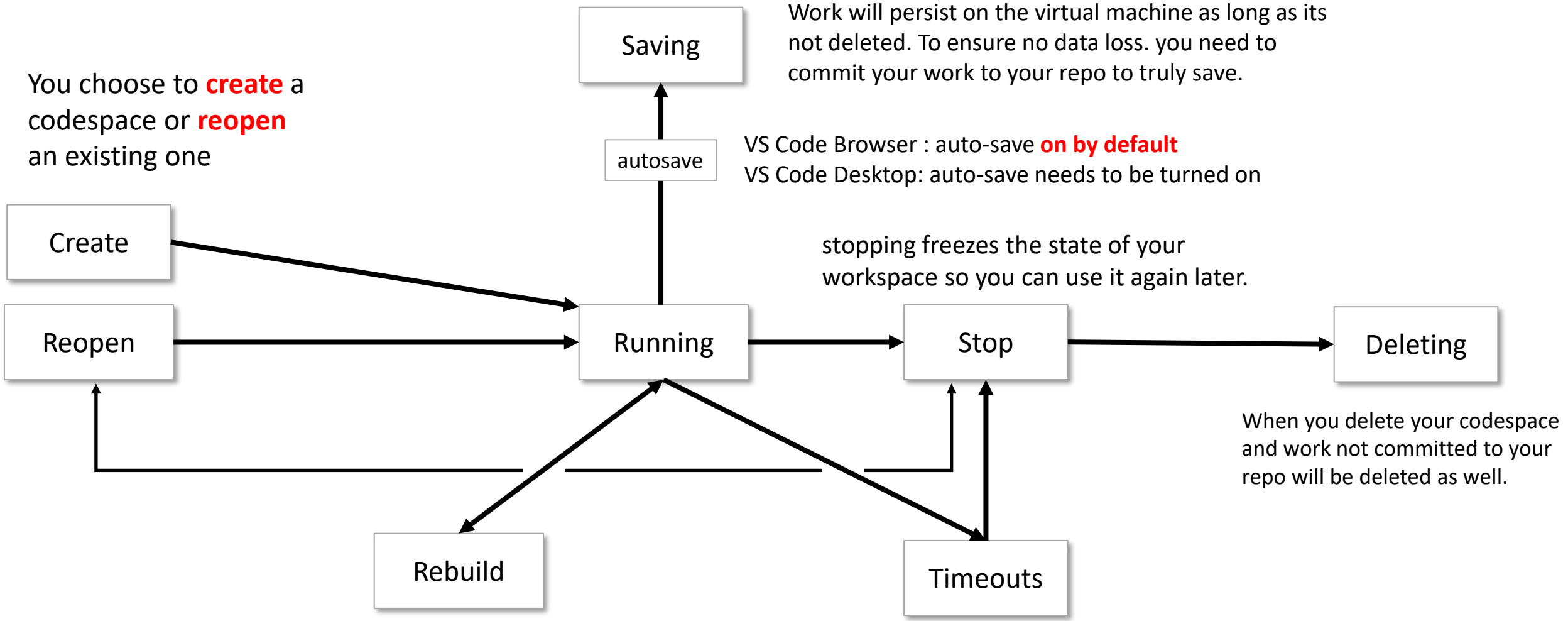


You can also use the **GitHub CLI** to create an open a codespace

```
gh codespace create
gh codespace open
```

There is a limit of concurrent codespaces allowed to run.
If you hit the limit you may need to stop other codespaces first.

GitHub Codespace Lifecycle



Renaming Codespaces

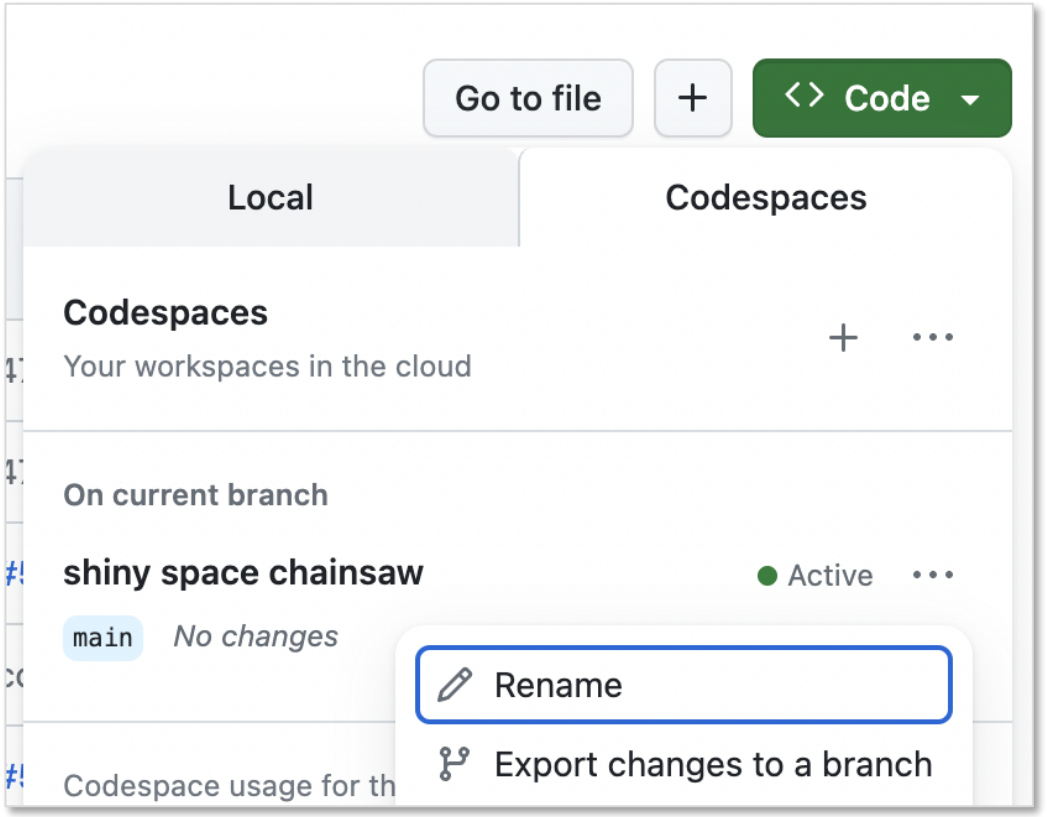
When you create a codespace it's assigned an **auto-generated display name**.

shiny-space-chainsaw-jjgx4w5r3q9xj

You can rename the auto-generated name in the case you have multiple codespaces to easily identify the purpose or state of your codespace.

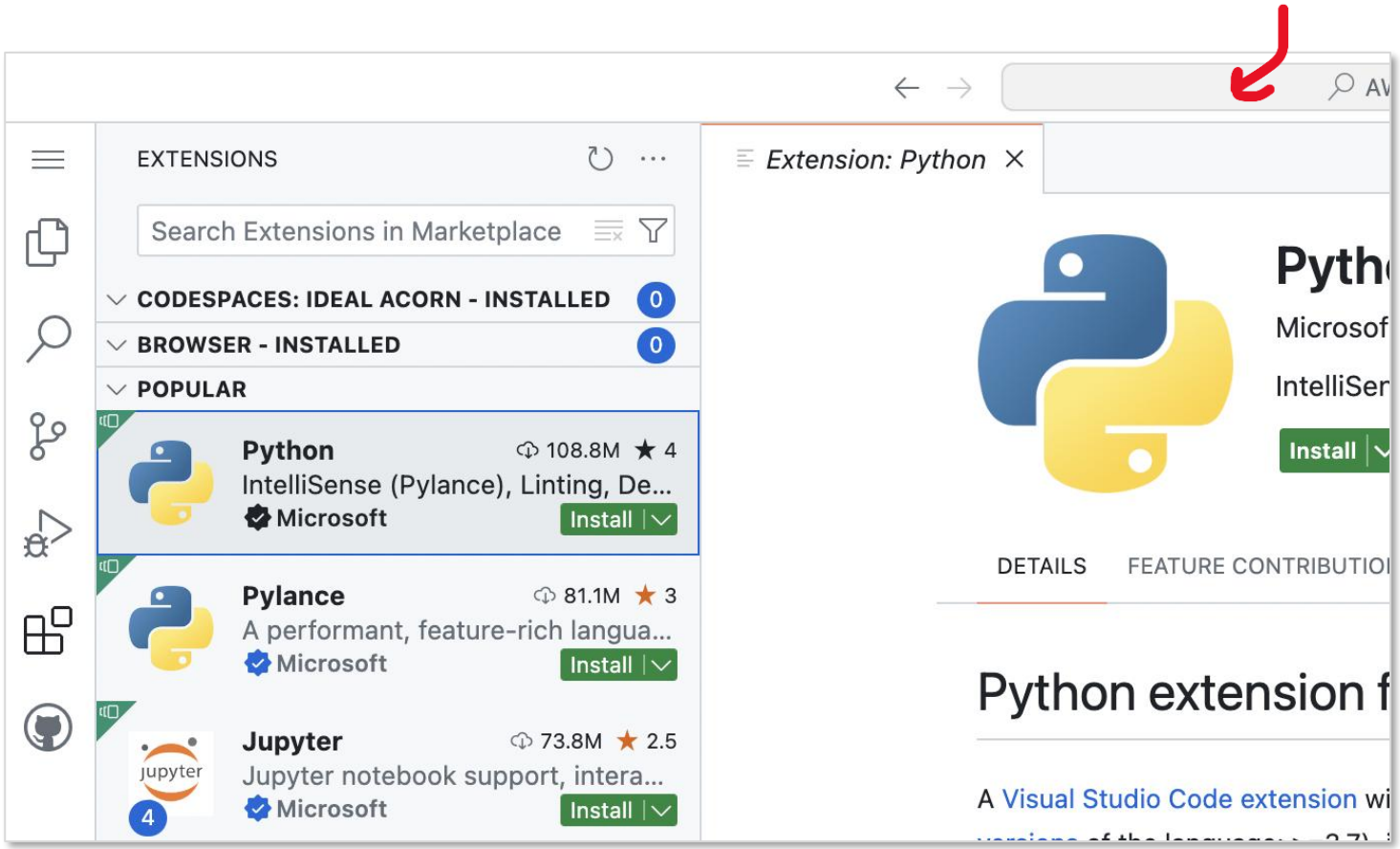
You can see all of your codespaces at

https://github.com/codespaces



VS Code Extensions

GitHub Codespaces **should persist your VS Code extensions** that you change in your codespace.

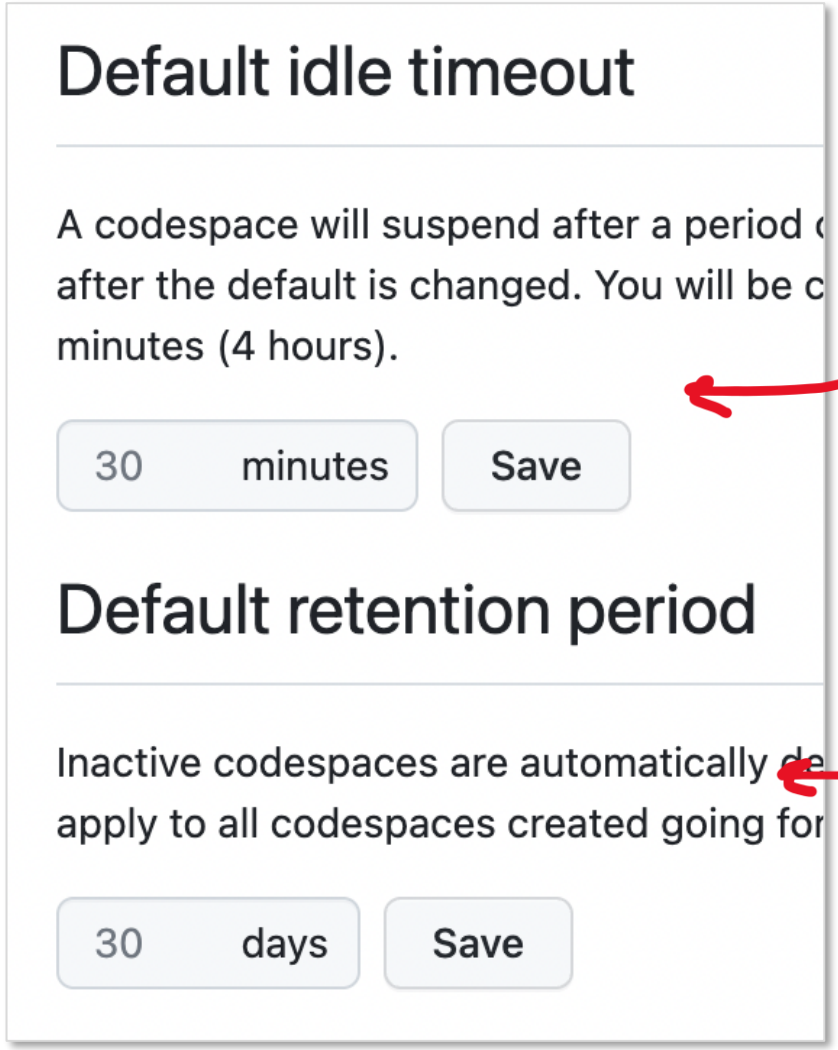


Most VSCode Marketplace extensions are available to install in VSCode Browser for GitHub codespaces

You can ensure your extensions reliably load every time by setting them in your devcontainer.json

Retention Period and Timeout Configuration

Under **GitHub Settings > Codespaces** you can change the **timeout** and **retention period.**



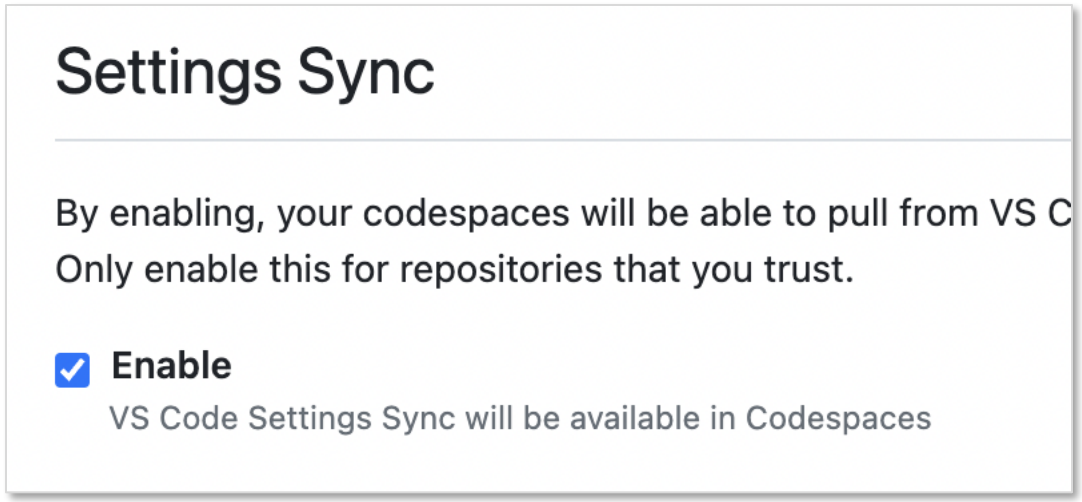
How long should codespaces wait to stop a codespace when their has been no activity.

- Minimum: 5 minutes
- Maximum 240 minutes (4 hours)
- Default: 30 minutes

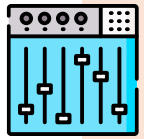
How many days should pass until inactive codespaces are automatically deleted?

- Minimum: 0 days
- Maximum 30 days
- Default: 30 days

Codespaces Sync Settings



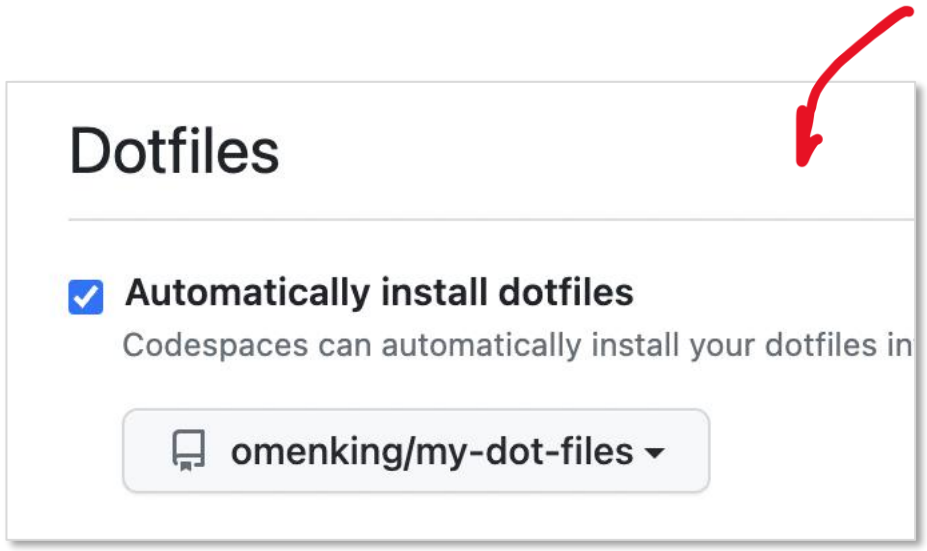
You can have codespaces able to pull from
VS Code Settings Sync service



VS Code Settings Sync lets you share your Visual Studio Code configurations such as settings, keybindings, and installed extensions across your machines so you are always working with your favorite setup.

Installing Dotfiles

You can configure GitHub Codespaces to **use dotfiles from a specifc repository** that will be used for all future codespaces you start.



When you create a new codespace with this feature, GitHub does two things:

1. GitHub clones your selected dotfiles repository to the codespace environment.

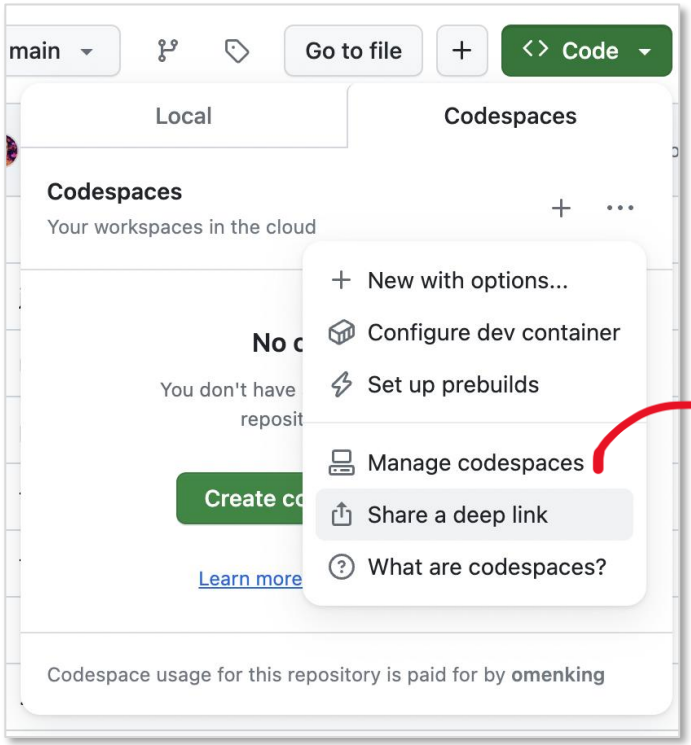
2. Looks for one of the following files to set up the environment.

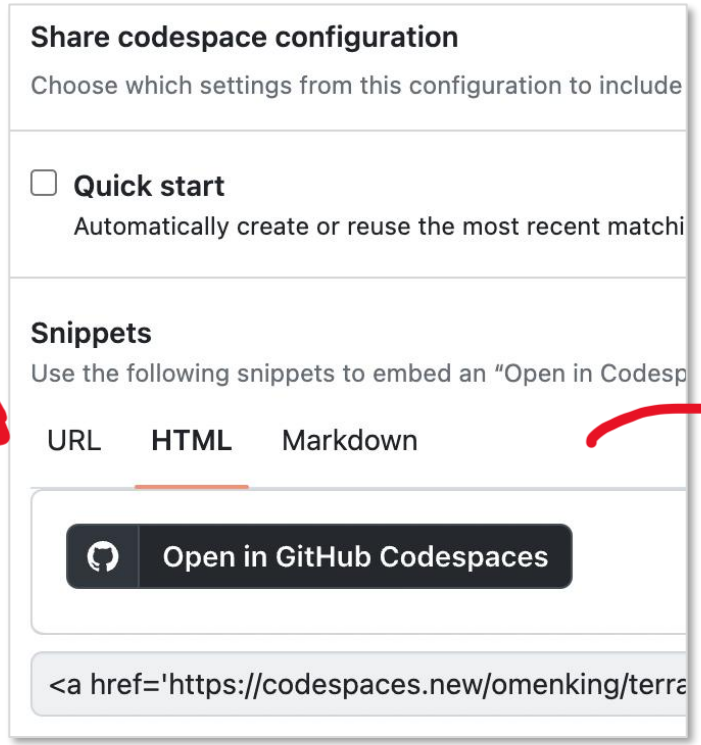
- *install.sh*
- *install*
- *bootstrap.sh*
- *bootstrap*
- *script/bootstrap*
- *setup.sh*
- *setup*
- *script/setup*

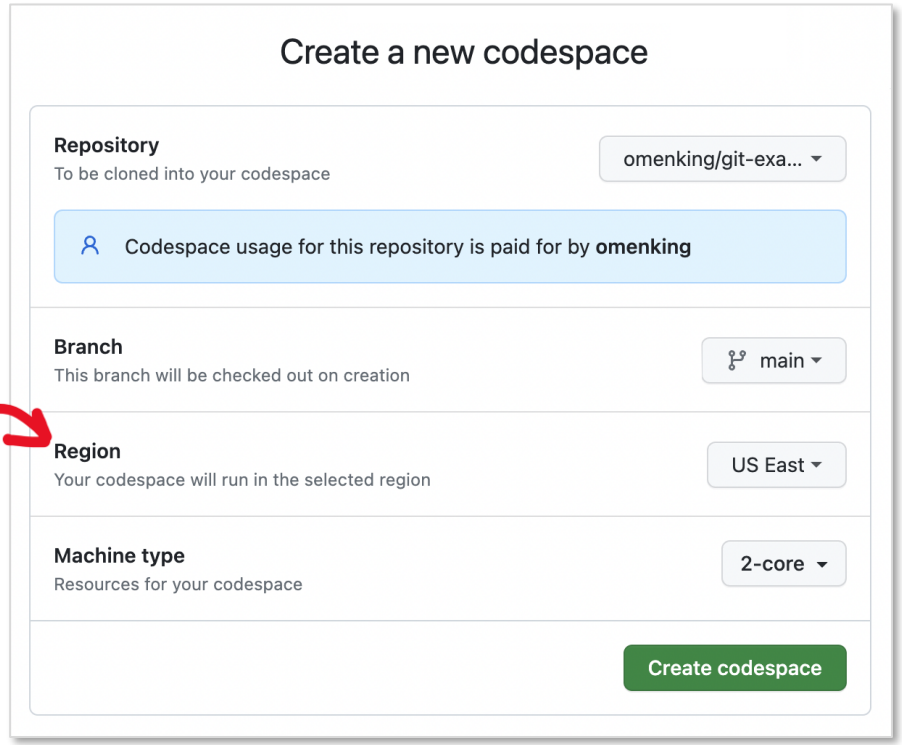
GitHub Codespaces Deep Links

Github Codespaces Deep Link is an easy to way to **generate a shareable link that will launch a Codespace for a specific Github Repo**.

https://codespaces.new/omenking/git-examples

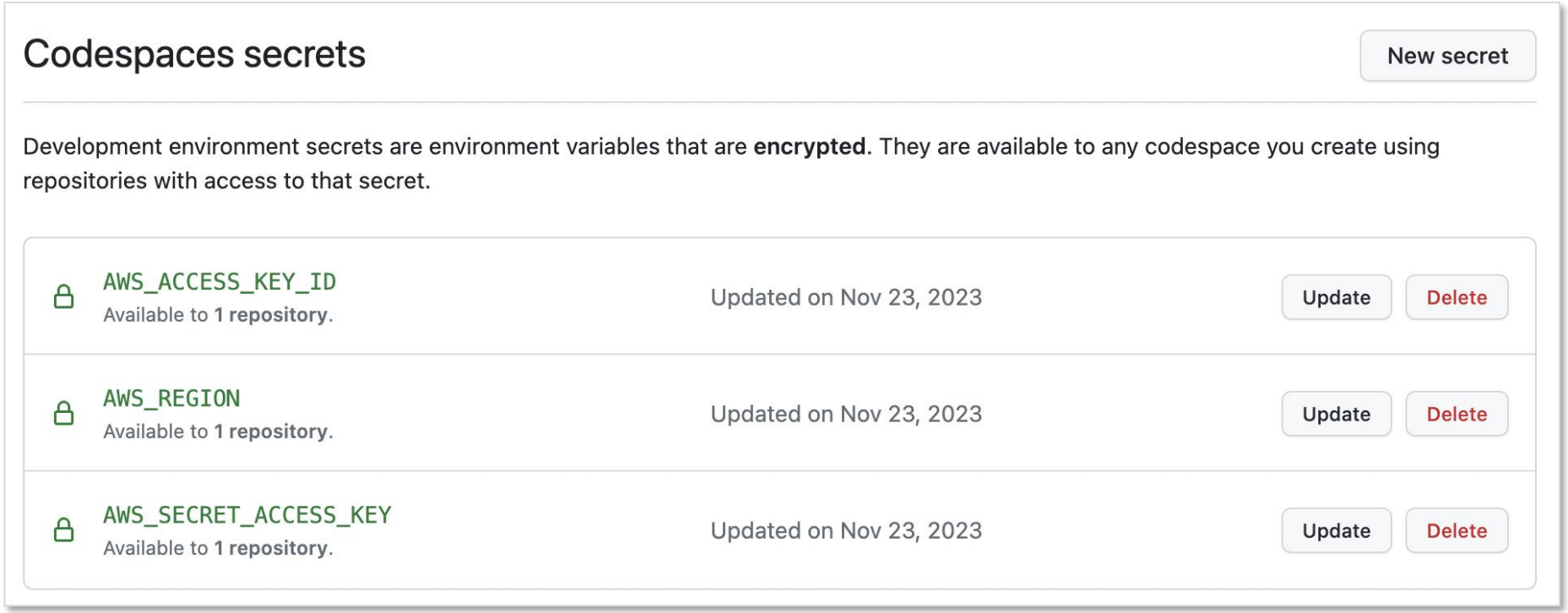






Codespaces Secrets

Under **Github Settings > Codespaces** you can can set GitHub Secrets



Secrets allows you to **set sensitive environment variables that will be loaded into your codespace**
This avoids hardcoding secrets, passwords or sensitive values into your codebase.

Dev Containers Configuration

Visual Studio Code Dev Containers allows you to **configure your docker container via json file**.

devcontainer.json file is expected in the root of your project.

```
{
  "name": "My Development Container",
  "image": "mcr.microsoft.com/vscode/devcontainers/base:0-focal",
  "settings": {
    "terminal.integrated.shell.linux": "/bin/bash"
  },
  "extensions": [
    "ms-python.python",
    "ms-vscode.cpptools"
  ],
  "forwardPorts": [3000],
  "postCreateCommand": "echo 'Container ready!'",
  "remoteUser": "vscode"
}
```

Dev Containers Configuration

You can **provide remote image** from a repo

```
"image": "mcr.microsoft.com/vscode/devcontainers/base:0-focal"
```

```
"build": {
    "dockerfile": "Dockerfile",
    "args": {
        "VARIANT": "3.8",
        "INSTALL_NODE": "true"
    }
},
```

You can **build your own image** by including the Dockerfile with your project

Dev Containers Configuration

```
"features": {
    "github-cli": "latest",
    "node": {
        "version": "lts"
    },
    "python": {
        "version": "3.8"
    }
},
```

Features allows you to quickly install programs on top of a common base image.

To **custom configure program** generally you will install these via your custom Dockerfile

```
FROM python:3.8-slim
WORKDIR /usr/src/app
ENV TF_VER=1.0.0
ENV TF_PREFIX=https://releases.hashicorp.com/terraform
RUN apt-get update && apt-get install -y --no-install-recommends \
curl \
unzip \
git \
&& rm -rf /var/lib/apt/lists/*

RUN curl -o terraform.zip ${TF_PREFIX}/${TF_VER}/terraform_${TF_VER}_linux_amd64.zip \
    && unzip terraform.zip \
    && mv terraform /usr/local/bin/ \
    && rm terraform.zip

COPY . .
EXPOSE 80
ENV NAME World
CMD ["python", "app.py"]
```

You could use the postCreateCommand to **install a bunch of programs** ontop of your base container.

```
"postCreateCommand": "bash ./.devcontainer/postCreateCommand.sh",
```

You can create your own features to quickly install on top of your own base image.

Dev Containers Configuration

Settings allows you to configure the **settings of VS Code**
Eg. Setting editor font and terminal size

```
"settings": {
  "terminal.integrated.shell.linux": "/bin/bash",
  "python.pythonPath": "/usr/local/bin/python"
},
"extensions": [
  "ms-python.python",
  "ms-vscode.cpptools",
  "github.vscode-pull-request-github"
],
```

Extensions allows you to install VS Code extensions
eg. VIM style keyboard controls

```
"forwardPorts": [3000, 5000],
```

You can list **ports you always want to forward**

Dev Containers Configuration

Command flags and their values we want to pass to when **docker run** command.

```
"runArgs": [
    "-e", "MY_ENV_VAR=myvalue",
    "-e", "ANOTHER_ENV_VAR=anothervalue",
    "-v", "/my/host/folder:/container/folder",
    "--network=host",
    "--user=$(id -u):$(id -g)",
    "--cpus=2.0",
    "--memory=2g",
    "--shm-size=1g",
    "--add-host=example.com:192.168.1.10",
    "--privileged",
    "--security-opt", "seccomp=unconfined",
    "--network-alias=myalias",
    "-e", "TZ=Europe/London",
    "-v", "/var/run/docker.sock:/var/run/docker.sock"
]
```

Dev Containers Configuration

Environment Variables
we want to pass to the running container

```
"containerEnv": {
    "NODE_ENV": "development",
    "PYTHONUNBUFFERED": "1",
    "TZ": "America/New_York",
    "LANG": "en_US.UTF-8",
    "HTTP_PROXY": "http://proxy.example.com:8080",
    "HTTPS_PROXY": "http://proxy.example.com:8080",
    "NO_PROXY": "localhost,127.0.0.1,.example.com",
    "JAVA_HOME": "/usr/lib/jvm/java-11-openjdk-amd64",
    "MAVEN_HOME": "/usr/share/maven",
    "DOCKER_TLS_VERIFY": "1",
    "DOCKER_HOST": "tcp://docker.example.com:2376",
    "DOCKER_CERT_PATH": "/certs/client",
    "DATABASE_URL": "postgresql://user:password@localhost:5432/mydatabase",
    "REDIS_URL": "redis://localhost:6379",
    "API_KEY": "your-api-key",
    "LOG_LEVEL": "debug"
}
```

Dev Containers Configuration

Mounts allows you to **map files and folders to your containers**

```
"mounts": [
  "source=${localWorkspaceFolder}/data,target=/container/data,type=bind,consistency=cached",
  "source=myvolume,target=/container/volume,type=volume",
  "source=/var/run/docker.sock,target=/var/run/docker.sock,type=bind",
  "source=${localWorkspaceFolder}/config/settings.json,target=/container/settings.json,type=bind,consistency=cached",
  "source=${localWorkspaceFolder}/docs,target=/container/docs,type=bind,consistency=cached,readonly"
]
```

- **Bind Mount a Local Directory:** Mount a local directory for direct file access.
- **Mount a Docker Volume for Persistent Storage:** Use a Docker volume for data that persists across container rebuilds.
- **Mount the Docker Socket:** Allow the container to interact with the host's Docker daemon.
- **Mount a Specific File:** Mount a specific configuration file into the container.
- **Mount a Host Directory as ReadOnly:** Mount a directory from the host as read-only.

Dev Containers Configuration

command that **runs after creating the container.**

```
"postCreateCommand": "echo 'Container ready!'",
"remoteUser": "vscode"
```

the **default username** that VS Code should use when connecting to the container.

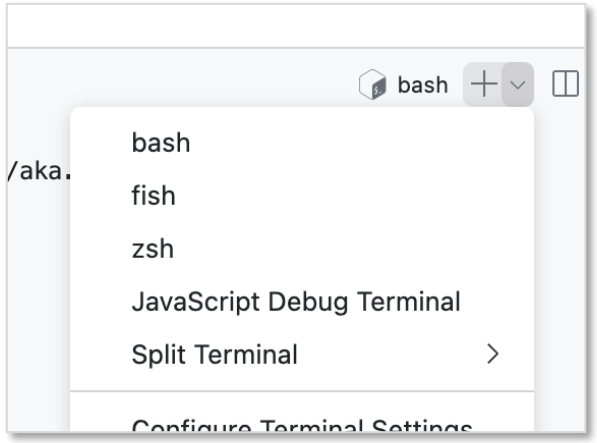
You could use the postCreateCommand to **install a bunch of programs** ontop of your base container.

```
"postCreateCommand": "bash ./.devcontainer/postCreateCommand.sh",
```

Changing Shell

GitHub Codespaces VS Code Browser default devcontainer images comes with:

- Bash (default)
- Zsh
- Fish



If you build or provide an image to devcontainer.json with other shells installed, you can define and provide the path to that shell for use:

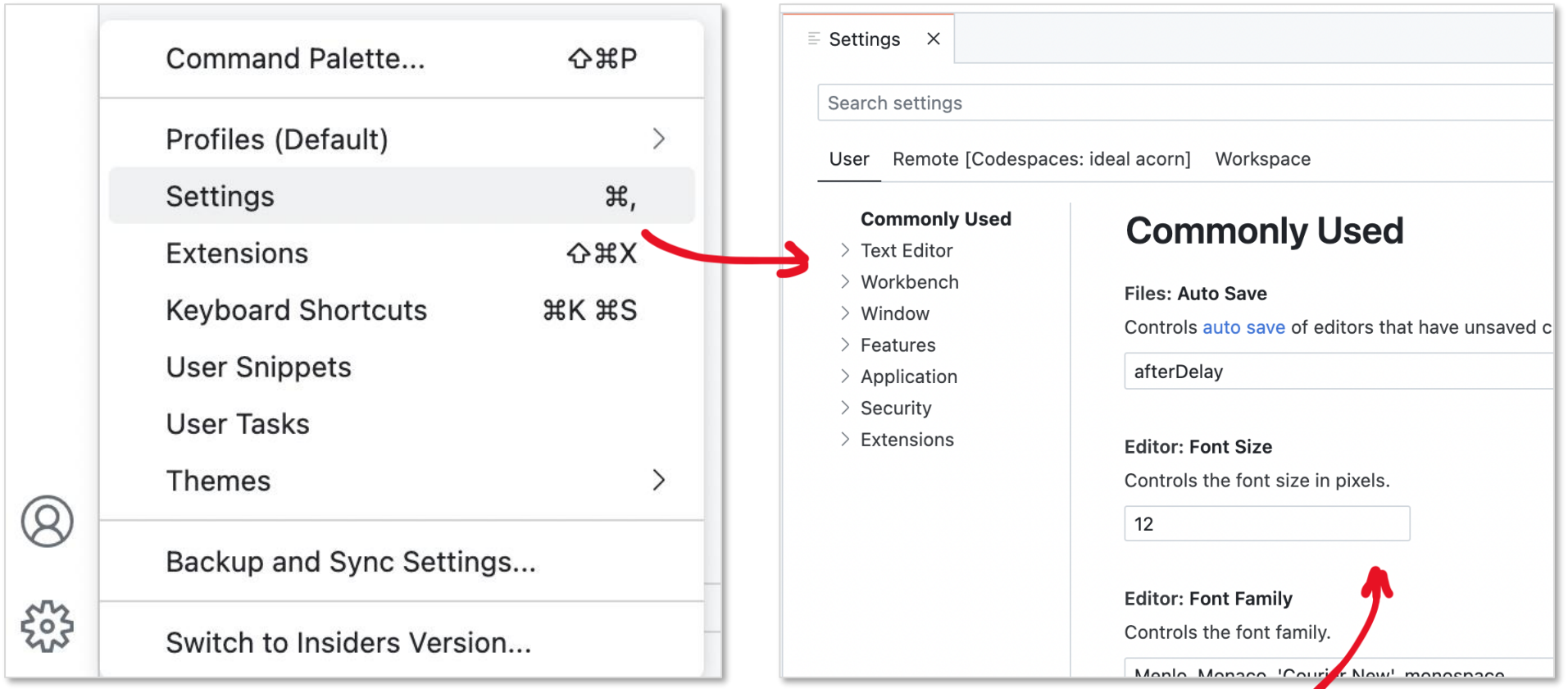
```
{
  "terminal.integrated.profiles.linux": {
   "csh": {
     "path": "csh"
   }
  },
  "terminal.integrated.defaultProfile.osx": "zsh",
  "terminal.integrated.defaultProfile.linux": "bash",
  "terminal.integrated.defaultProfile.windows": "PowerShell"
}
```

You can configure the settings.json in the VSCode to change the defaults based on Operating System.

settings.json is directly edited in the UI of VS Code.

VS Code Settings

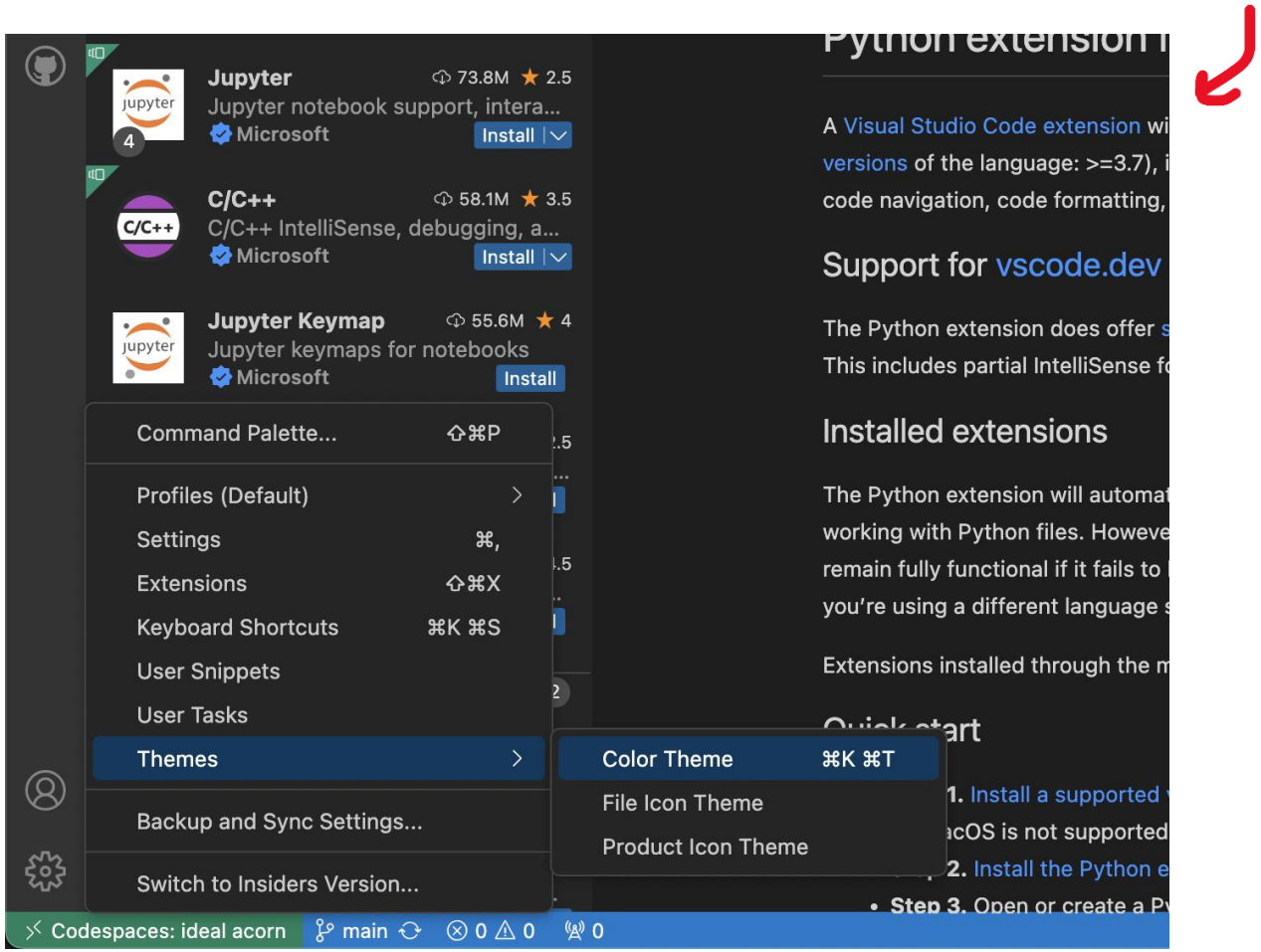
Github Codespaces **should persist your VS Code settings** that you change in your codespace.



When you edit Settings in the UI you are **actually modifying a settings.json file**.
It get synced and backed up hosted by GitHub
You can override setting.json options in your devcontainer.json file

VS Code Theme

Github Codespaces **should persist your VS Code theme** that you change in your codespace.



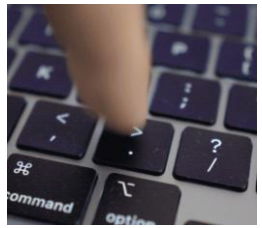
user interface decorators or themes are personal choices that should not be put in the devcontainer. json file

GitHub.dev Editor

Github.dev Editor is a **VS Code browser that instantly loads** that **has no attached compute.**

There are two ways you can open your GitHub repo in Github.dev:

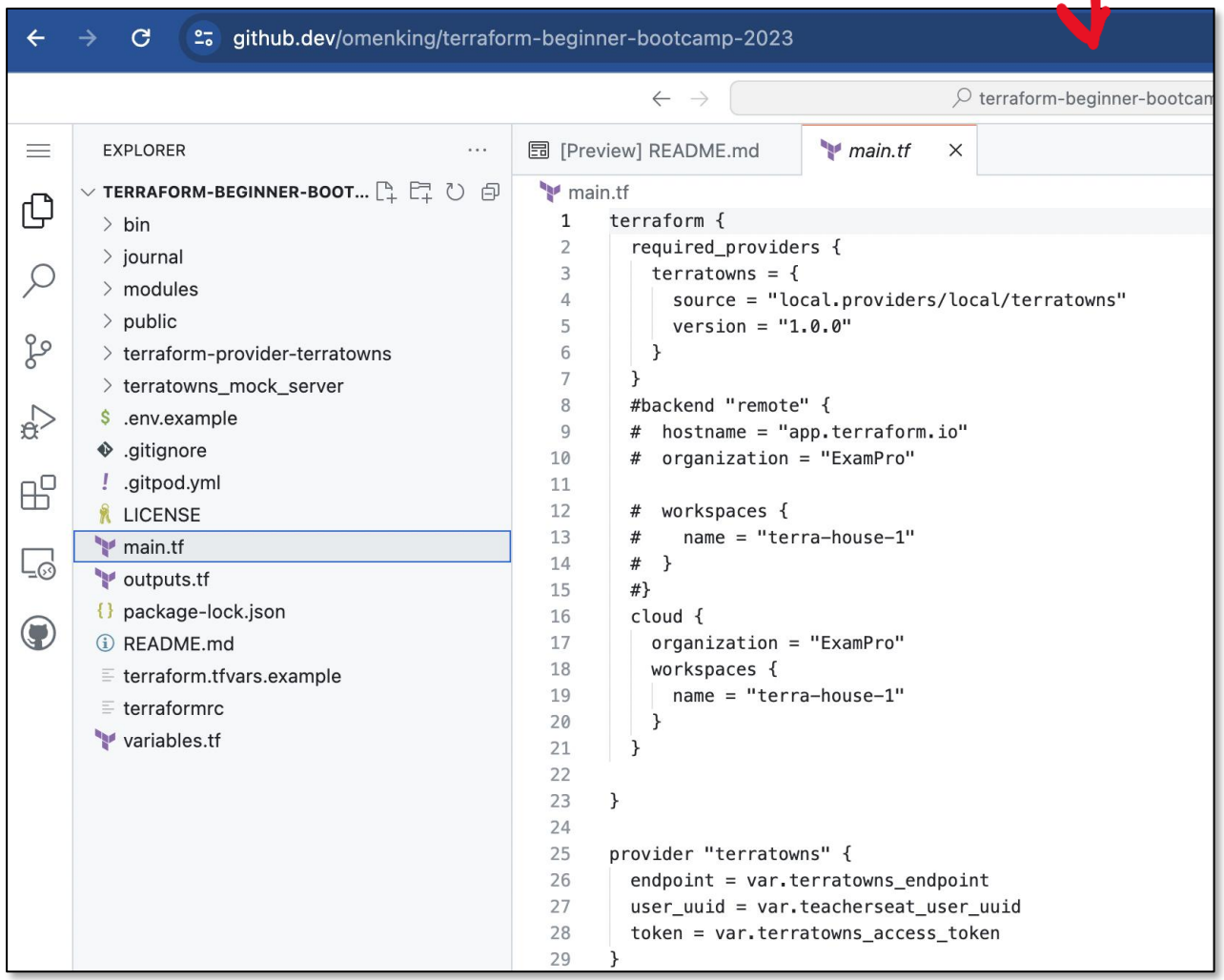
1. Press the period (.) hotkey on your keyboard when you have your GitHub Repo page opened.



2. Place your GitHub Path after the URL:

github.dev/ <user/org> / <repo name>

Github.dev is intended for **quick changes to file and committing code** in an IDE experience.



GitHub.dev vs GitHub Codespaces

	GitHub.dev	Codespaces
Cost	FREE	• Free monthly quota • Storage and Core hours per month • Machine type changes cost
Availability	Available to everyone on GitHub.com.	
Start up	• Instantly launches. • No devcontainer.json configuration	• Takes a few minutes to startup • Does load devcontainer.json
Compute	• There is no attached compute • You can't run code or apps	• Dedicated attached VM • Run and debug your code and apps
Terminal access	No terminal	Terminal access
Extensions	Limited subset of extensions	Most extensions for them VS Code Marketplace

Open Source

Open-source is **source code made freely available** for possible modification and redistribution.

Open source benefits include:

- Encourages global collaboration.
- Speeds up innovation.
- Offers adaptability and customization.
- Reduces software costs.
- Enhances learning for developers.
- Typically high-quality and reliable.
- Provides transparency for trust and security.

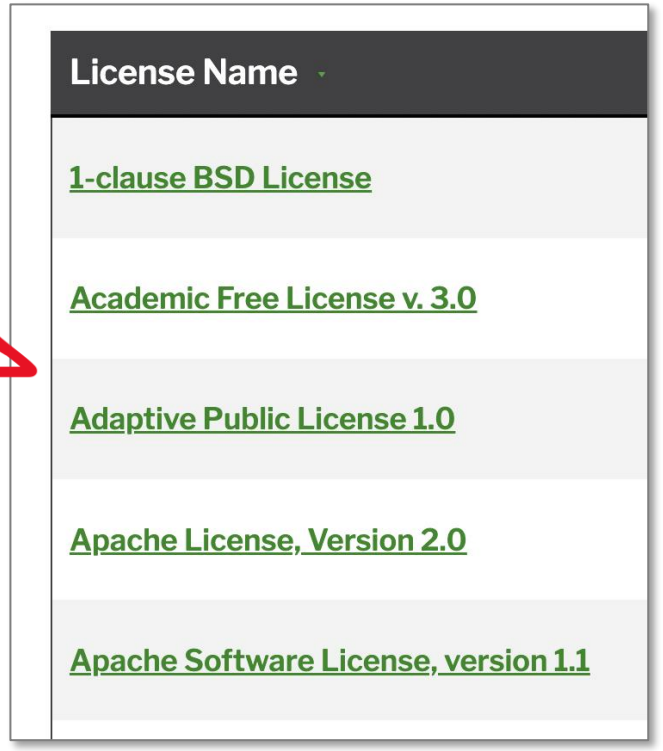
Open-source software has often have **free-community versions** which makes it easy for personal developers or small organizations to quickly adopt technology.

Ranking	Project	Leading company	Market value
1	Linux	Red Hat	$16 billion
2	Git	GitHub	$2 billion
3	MySQL	Oracle	$1.87 billion
4	Node.js	NodeSource	?
5	Docker	Docker	$1 billion
6	Hadoop	Cloudera	$3 billion
7	Elasticsearch	Elastic	$700 million
8	Spark	Databricks	$513 million
9	MongoDB	MongoDB	$1.57 billion
10	Selenium	Sauce Labs	$470 million

Open Source

The Open Source Initiative (OSI) is a non-profit organization is the steward of the Open Source Definition, the set of rules that define open source software

THE OSI maintains **a list of open-source licensing documents** for all different use-cases which are readability available for project owners to adopt



The Top five most popular open-source licenses:

- **MIT License:**
  - Known for its simplicity and permissiveness.
- **GNU General Public License (GPL):**
  - Ensures that modified versions remain open.
- **Apache License 2.0:**
  - Provides explicit patent rights grants.
- **GNU Lesser General Public License (LGPL):**
  - Similar to GPL but with some allowances for linking to proprietary software.
- **BSD License:**
  - Characterized by minimal restrictions on redistribution.

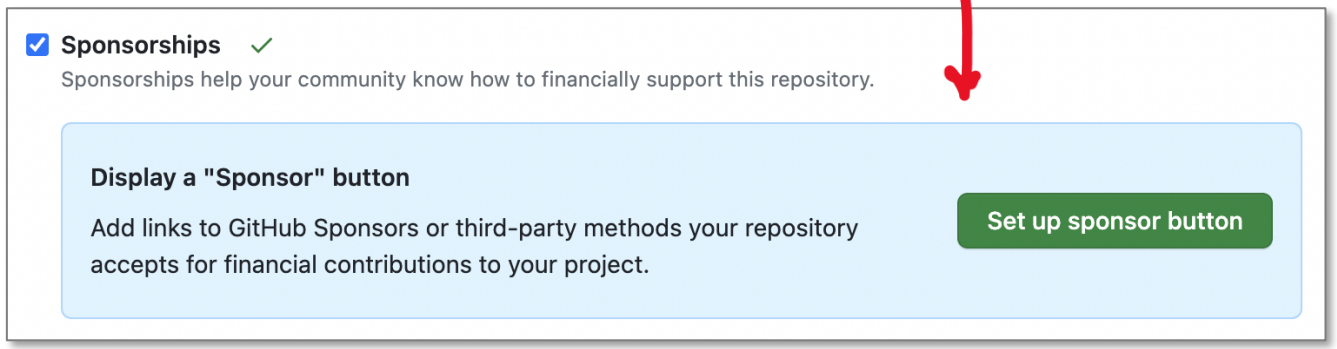
GitHub Sponsors

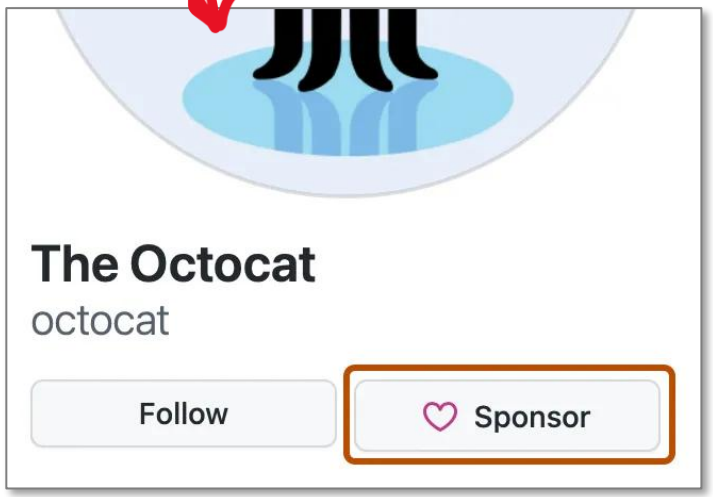
GitHub Sponsors allows GitHub users **to collect donations for their GitHub hosted open-source projects.**

Sponsorship payments are facilitated two ways:

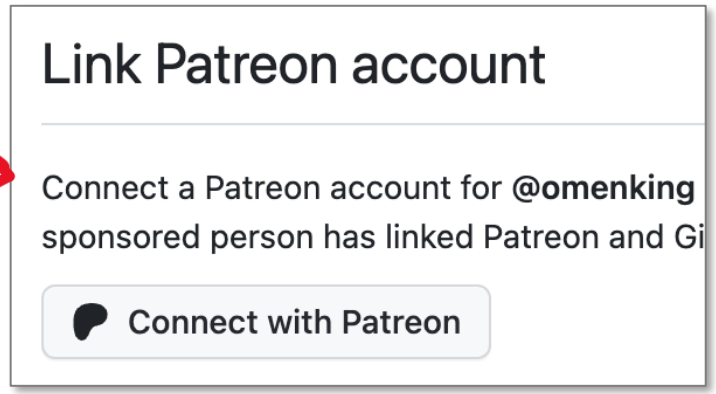
- Payments through GitHub on Github.com
- Payments through Patreon on Github.com

You can turn sponsor on for a **specific repo**





You can **connect Patreon** via your Account settings



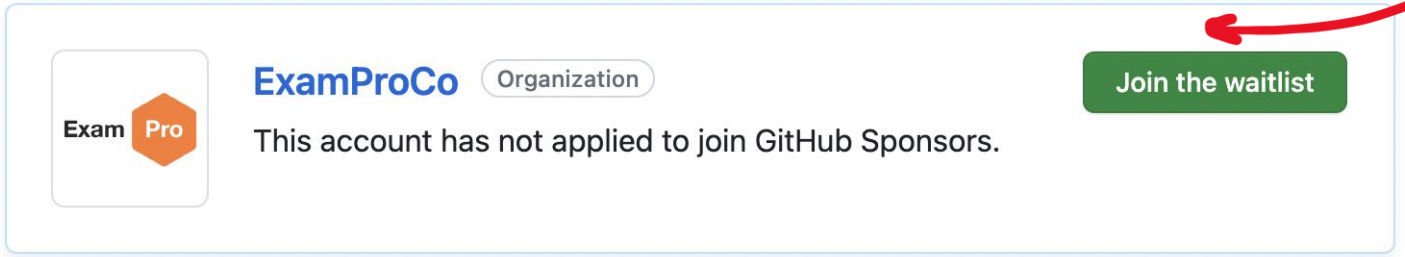
GitHub Sponsors

To receive sponsorship through GitHub on Github.com you'll need to get accepted into **GitHub Sponsors**.

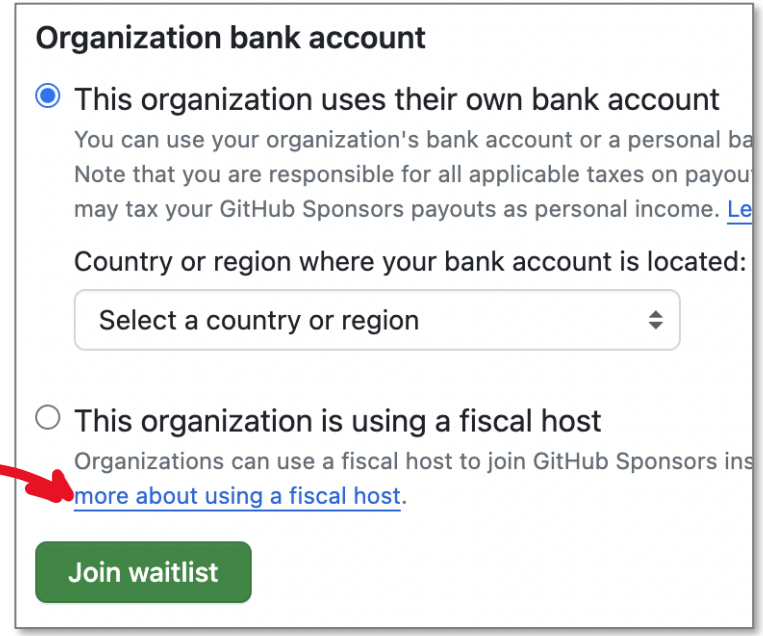
https://github.com/sponsors

Personal accounts can apply

Organizational accounts can apply



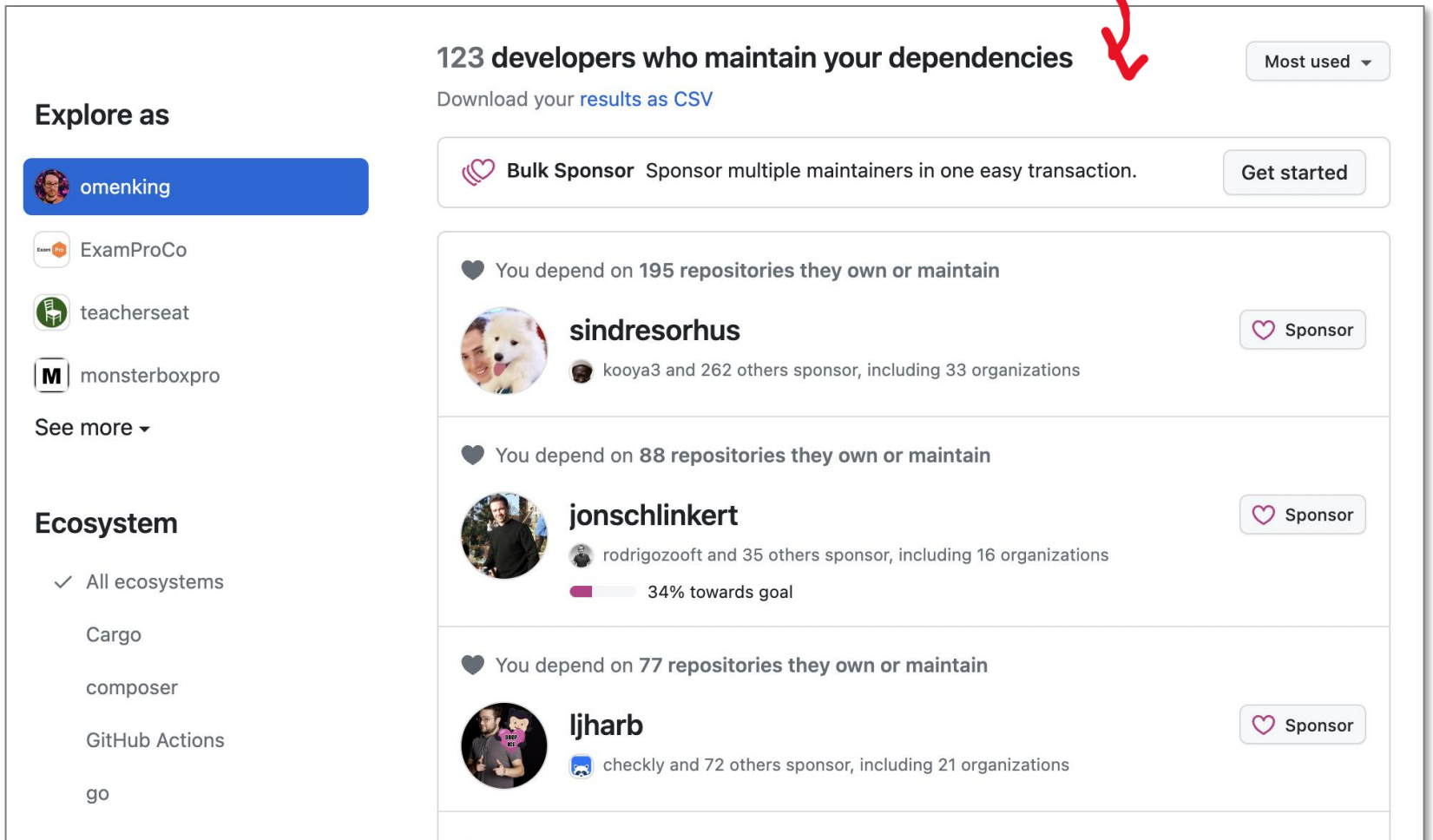
Setting up GitHub sponsors via GitHub requires you to connect your bank account and may have **other fiscal requirements**.



GitHub Sponsors

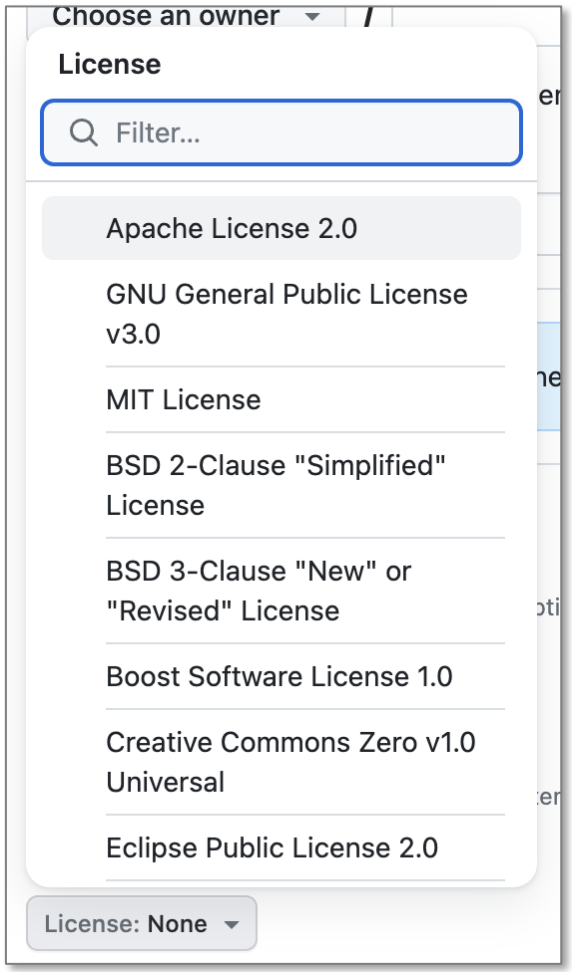
GitHub makes it easy to locate people who maintains your dependencies so you can **support open-source contributors**

https://github.com/sponsors/explore



GitHub and Open source projects

When creating a new repo GitHub makes it easy to **quickly add** a selected Open source license to your repo



They will **display** this on the GitHub repo page

README | Code of conduct | MIT license | Security

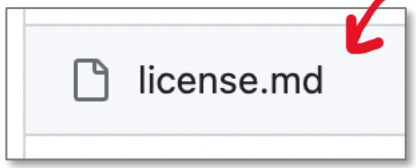
The MIT License (MIT)

Copyright (c) 2024 Vercel, Inc.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

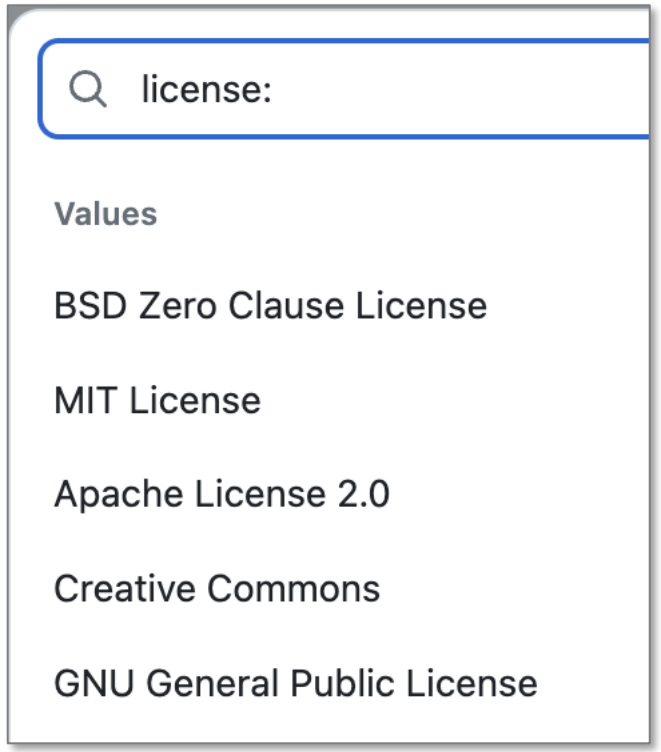


The file in the repo will be listed as either:

- license.md, LICENSE or LICENSE.md

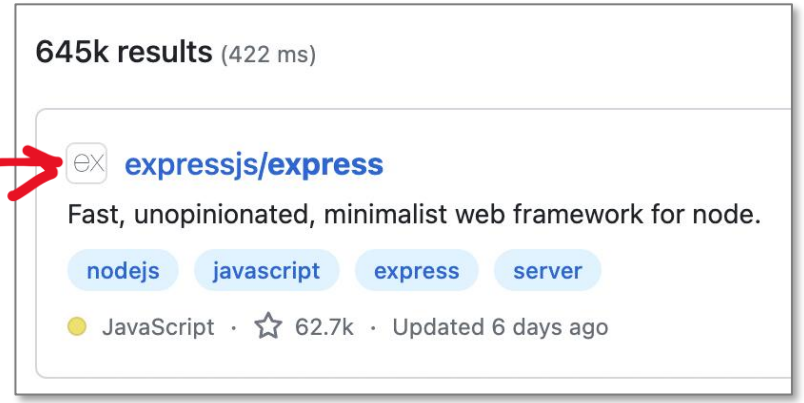
Open source discovery

GitHub **search** makes it easy to find public repo's based on Open source license



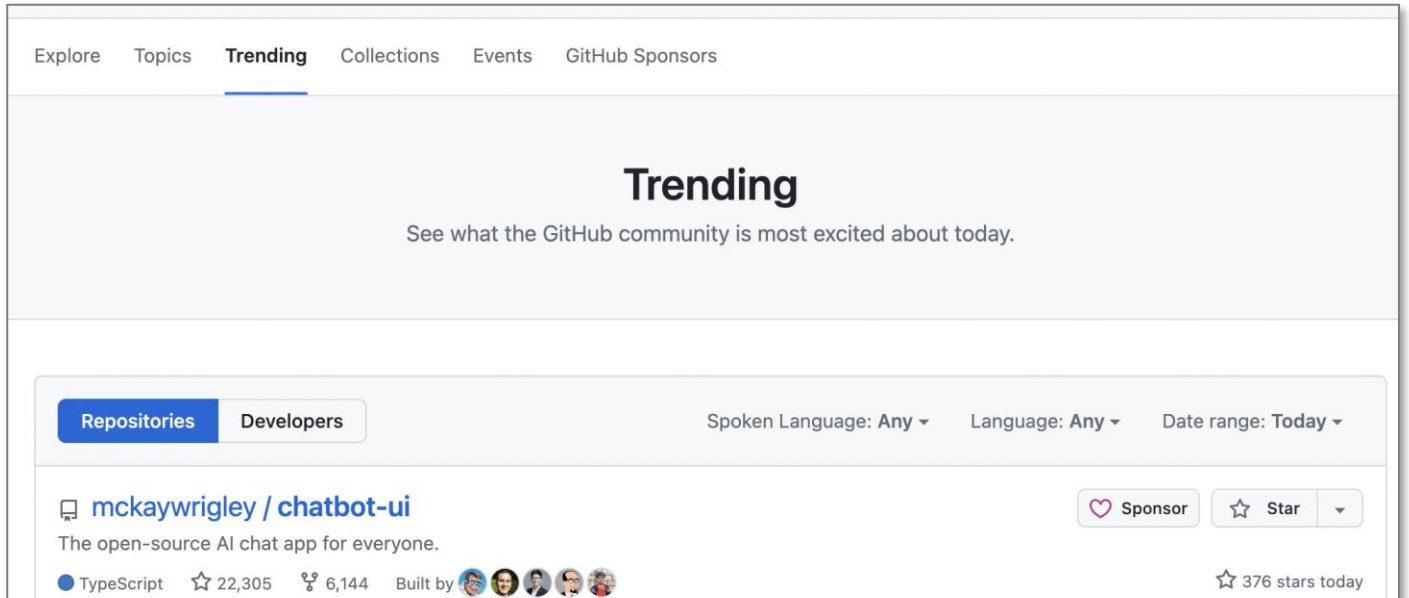
Many open-source GitHub repos are hosted or mirrored on GitHub.

Just type the open-source project name into search



In Explore:

- GitHub Topics allows you to find repos for specific categories eg. Azure
- GitHub Trending: shows you most popular repos

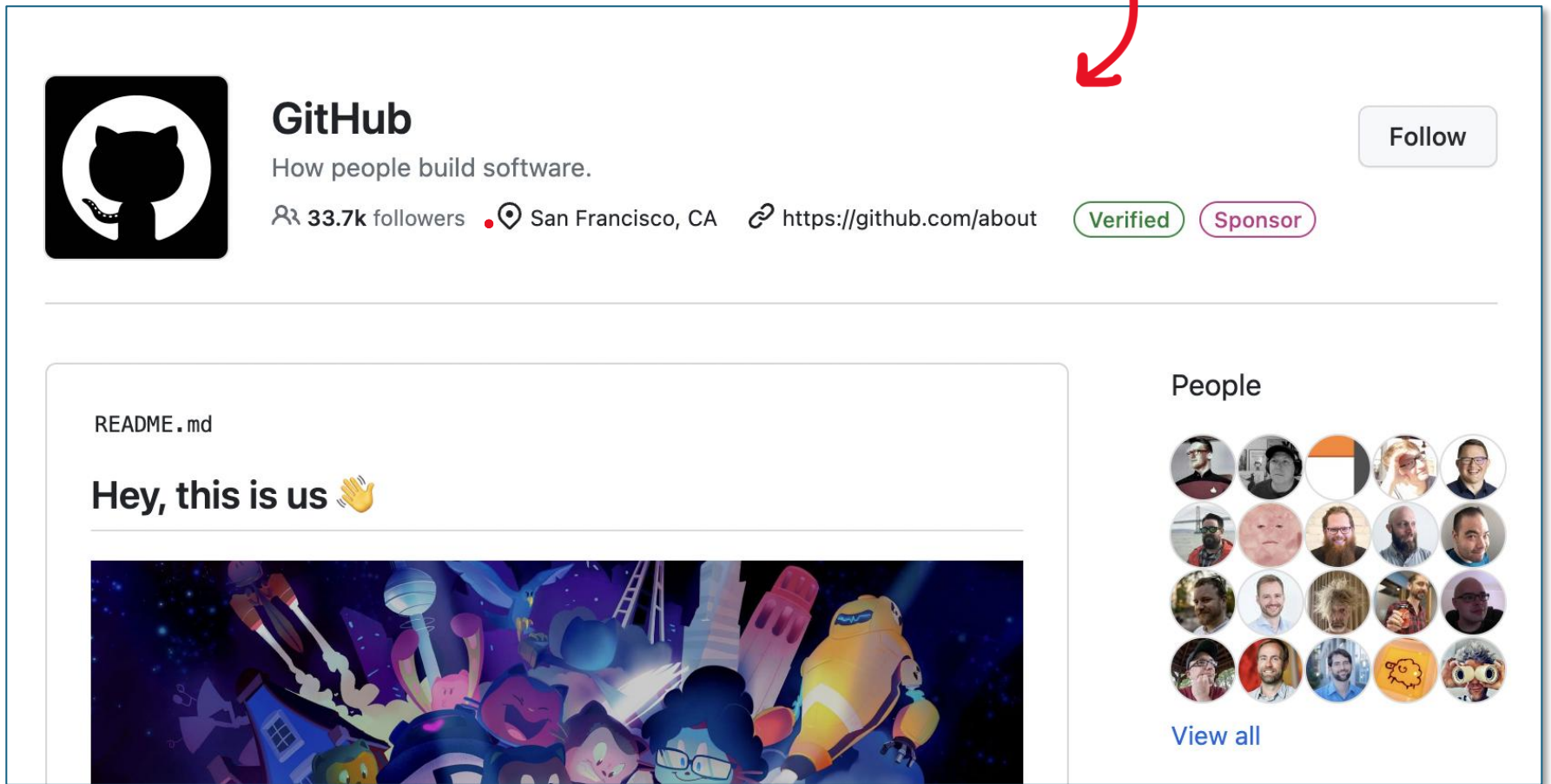


Following Organizations

On a GitHub Organizations public profile, you can **choose to follow** them to notifications of their public activity on your personal dashboard

This activity includes:

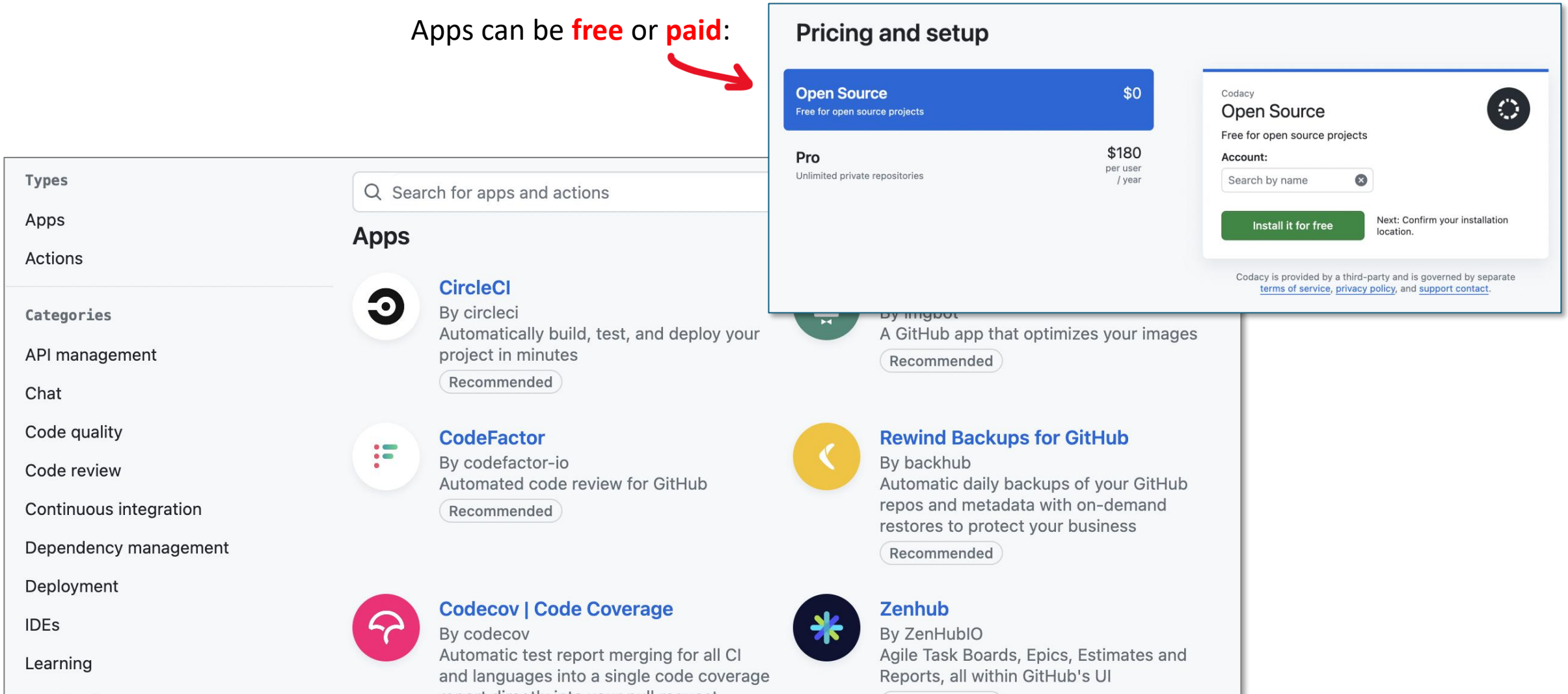
- new discussions
- sponsorships
- repositories



GitHub Marketplace

GIthub Marketplace are apps that **integrate with your Github repos** to provide additional functionality.

Apps can be **free** or **paid**:



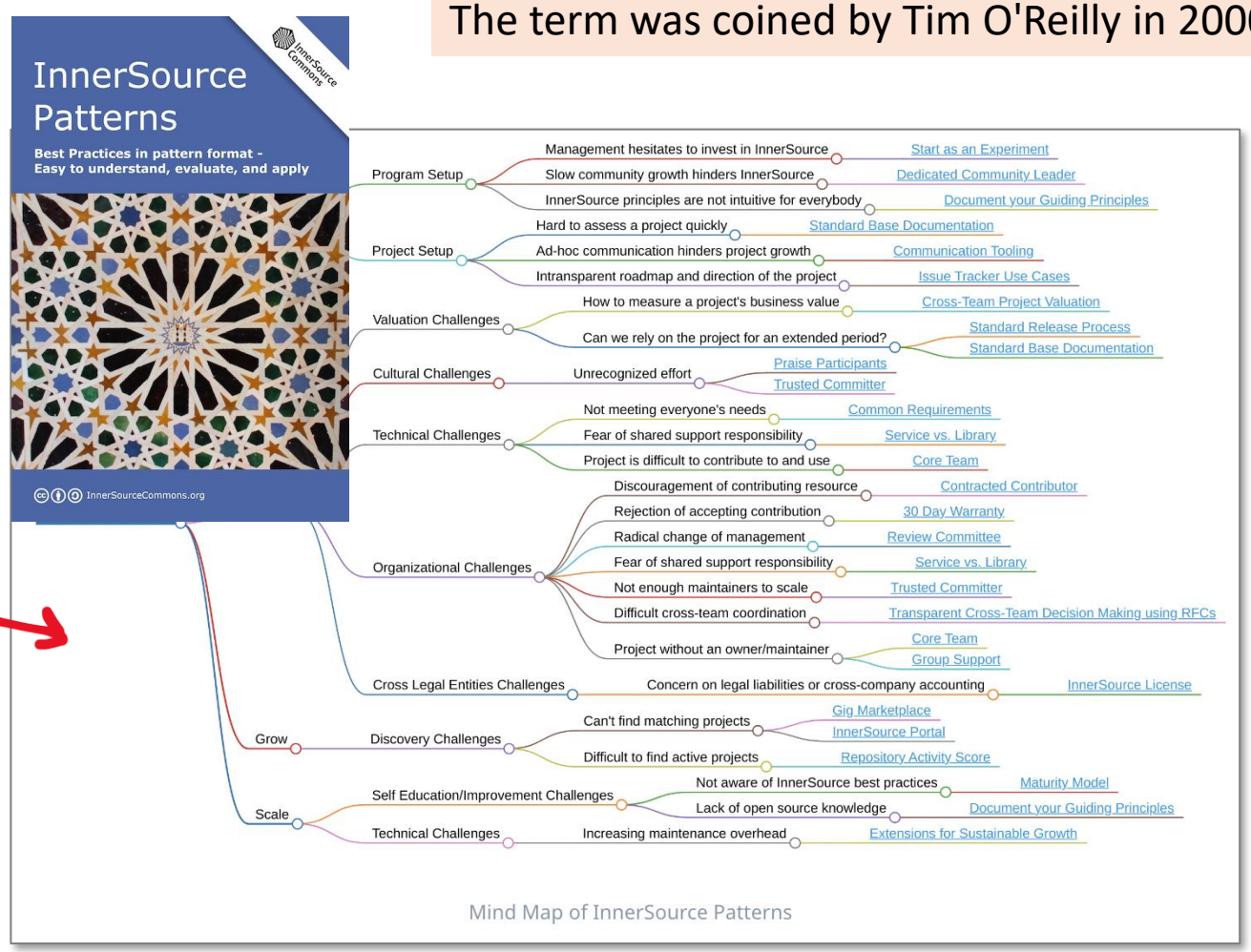
InnerSource

InnerSource is organization and development best practices of non-open-source and/or proprietary software.

The term was coined by Tim O'Reilly in 2000

InnerSource is **not a strict guideline** but **a loose strategy** to establish an open source-like culture within organizations

Innersourcecommons.org provide several **free structured resources** to help understand how to adopt InnerSource



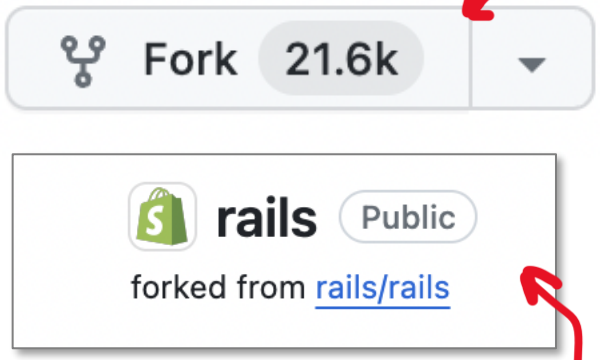
Mind Map of InnerSource Patterns

https://patterns.innersourcecommons.org/explore-patterns

InnerSource vs Open source

	InnerSource	OpenSource
Scope	Within an organization	Public, global community
Accessibility	Limited to company employees	Open to anyone
Purpose	Improve collaboration and efficiency internally	Share and collaborate on projects globally
Contribution	Employees of the organization	Anyone (developers, users, etc.)
Visibility	Code and discussions often private to the organization	Publicly available and visible
Governance	Dictated by internal policies and culture	Usually governed by open source licenses and community rules

Forking



A forked repo is indicated **underneath its repo name.**

Forking repo's allows you to create a copy of a repo.

Forking allows you to:

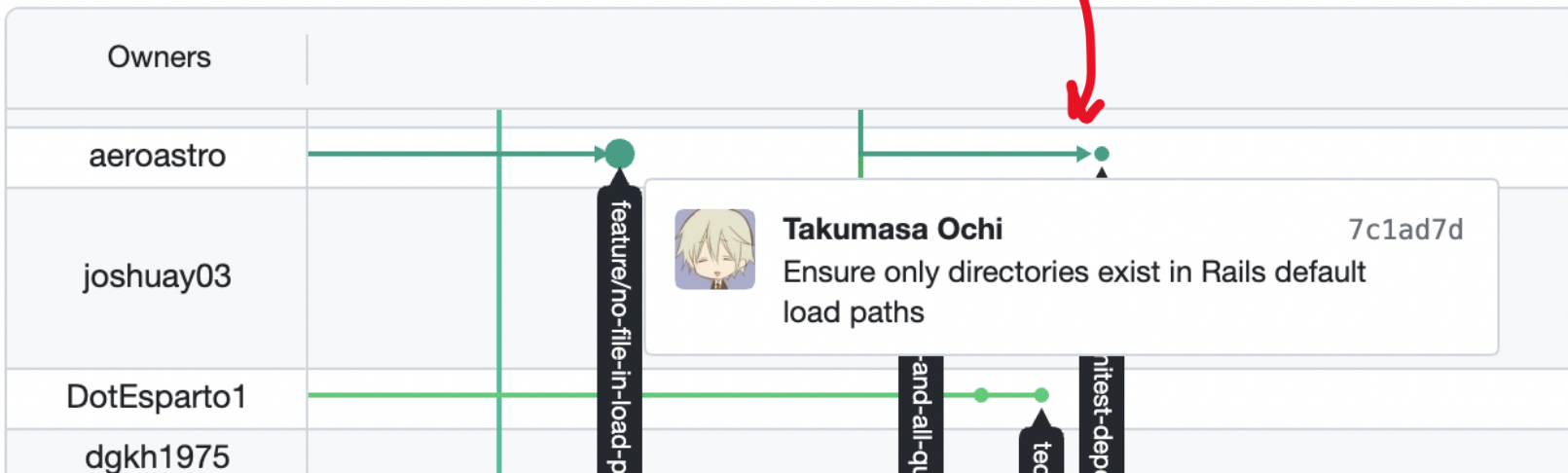
- Take an open-source or source available repo and go your own way
- Quickly apply bugs and patch fixes
- A separate repo to work on community contributions
  - You can create cross-repo pull-requests to get accepted in the original repo

You may prefer using a forked repo over the original.

You may find original projects abandon, and a forks becoming the go-to repo for the project.

Fork's help keep public projects alive, on the edge and collaborative

Using **Network Graph** under Repo Insights you can often find community solutions to bugs or missing features.



Discoverable repos

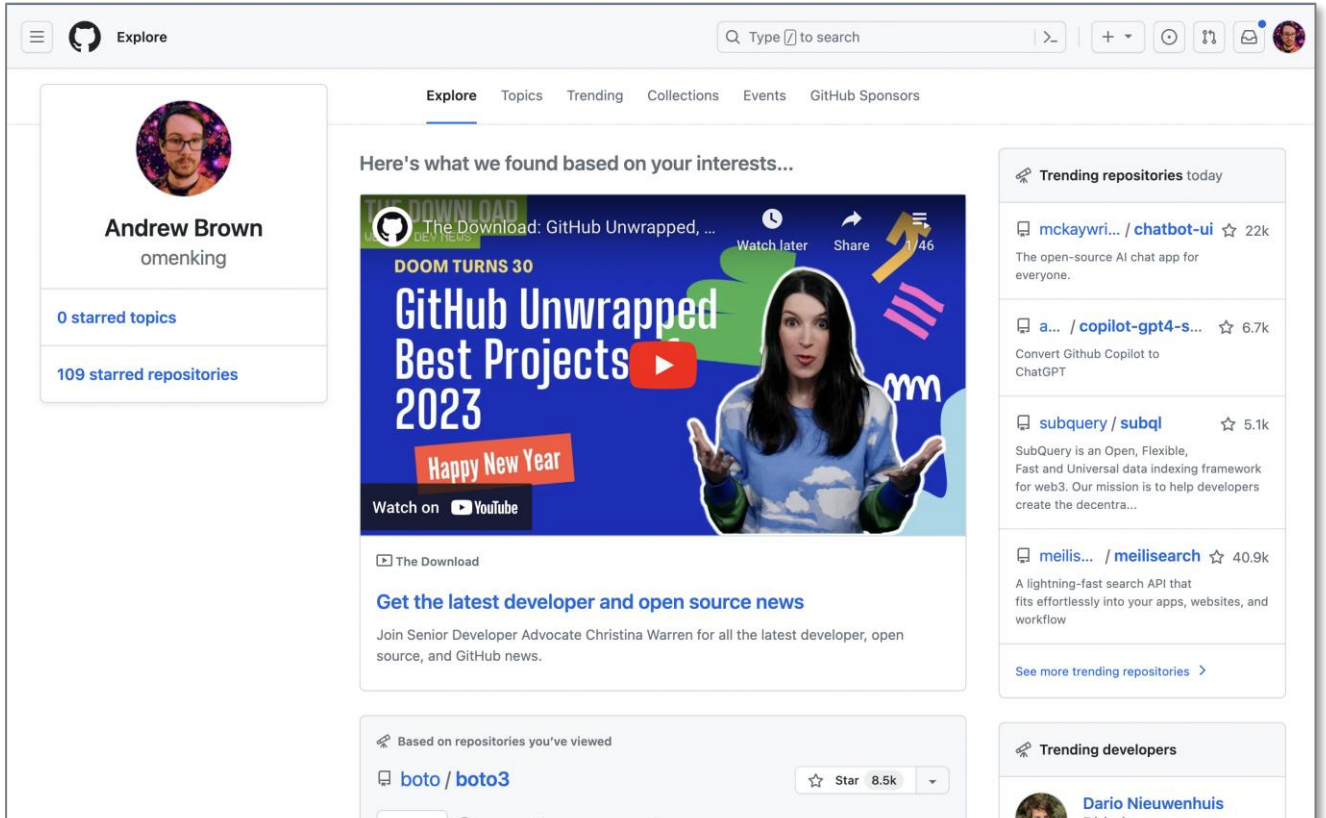
GitHub Repos can be set as public, making repos easily searchable on GitHub and via search engines.

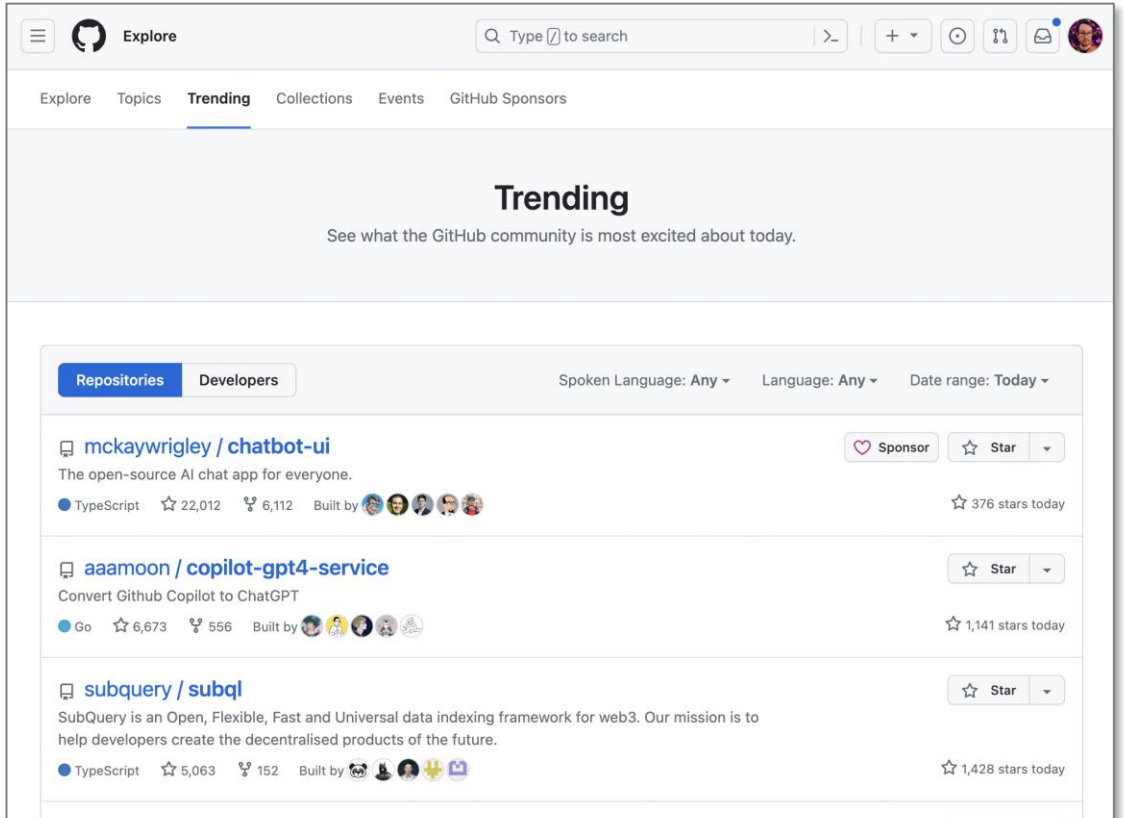
Public repos serve a community purpose for knowledge sharing, educational resources and open-source projects.

Besides just search GitHub curates content via their community pages:

https://github.com/explore

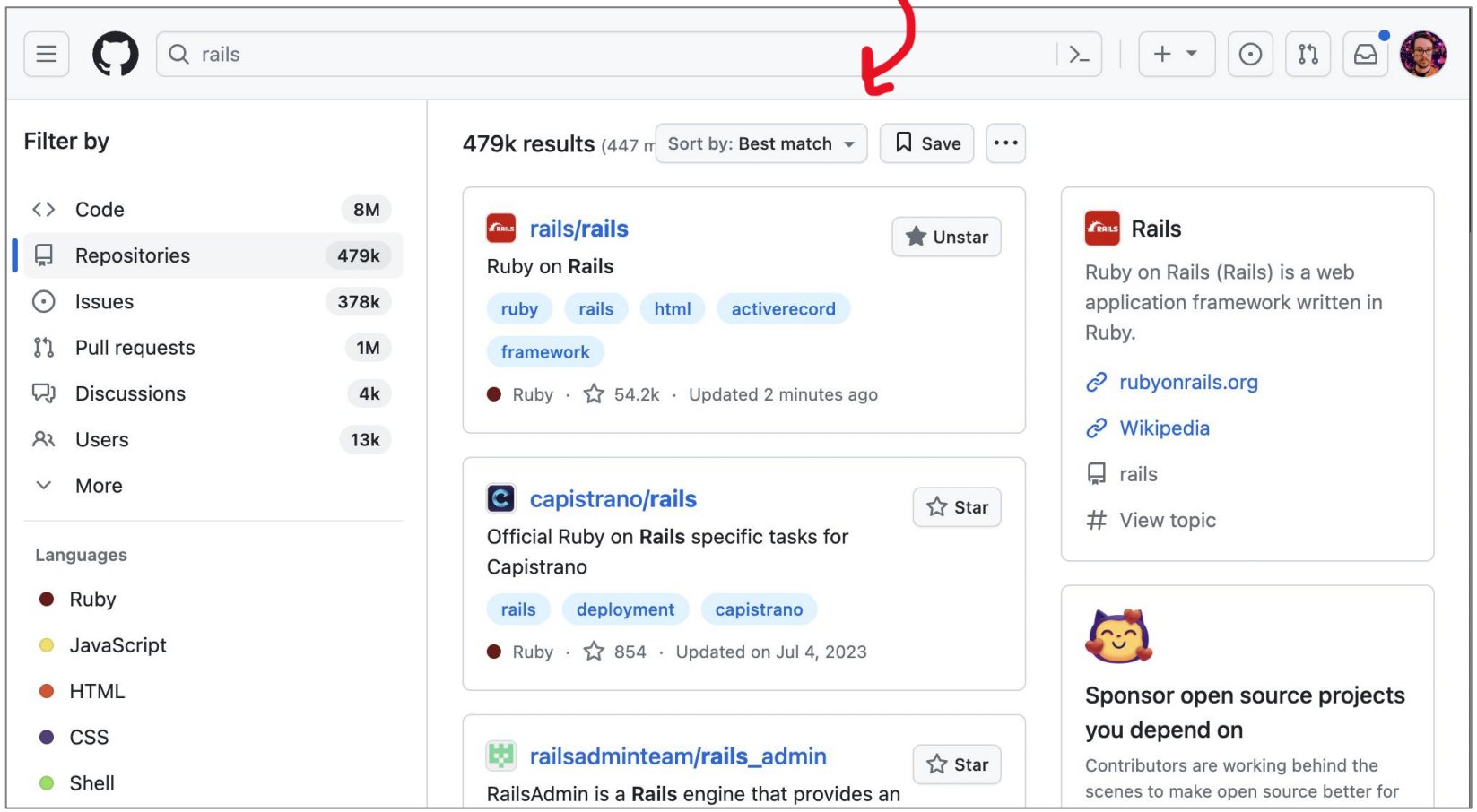
https://github.com/trending



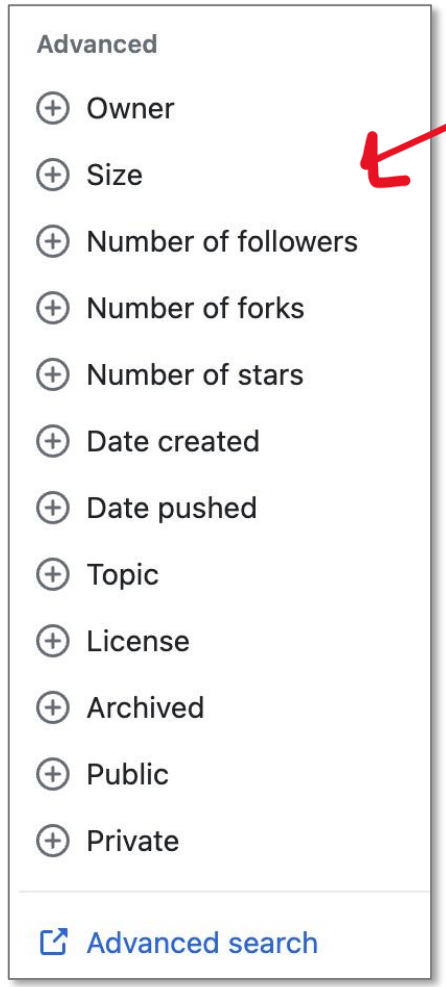


Discoverable repos

GitHub provides a **robust search** that lets us not only repo's names but the contents of their repos.

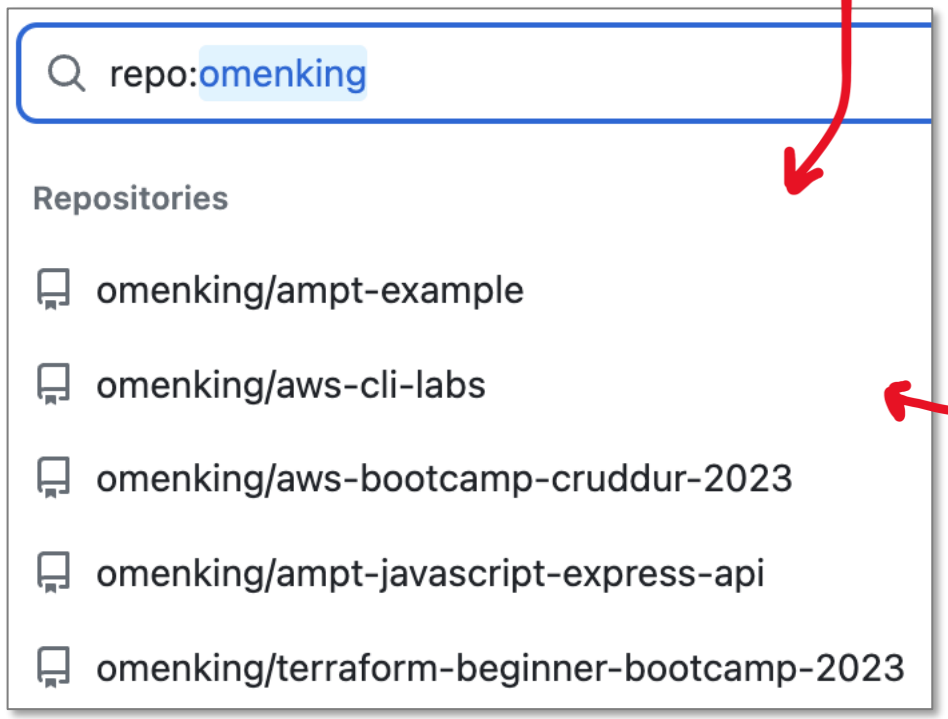


GitHub also has several **advanced search** options



Discoverable repos – Search Syntax

GitHub global search box has a **search syntax** to help quickly apply advanced search options

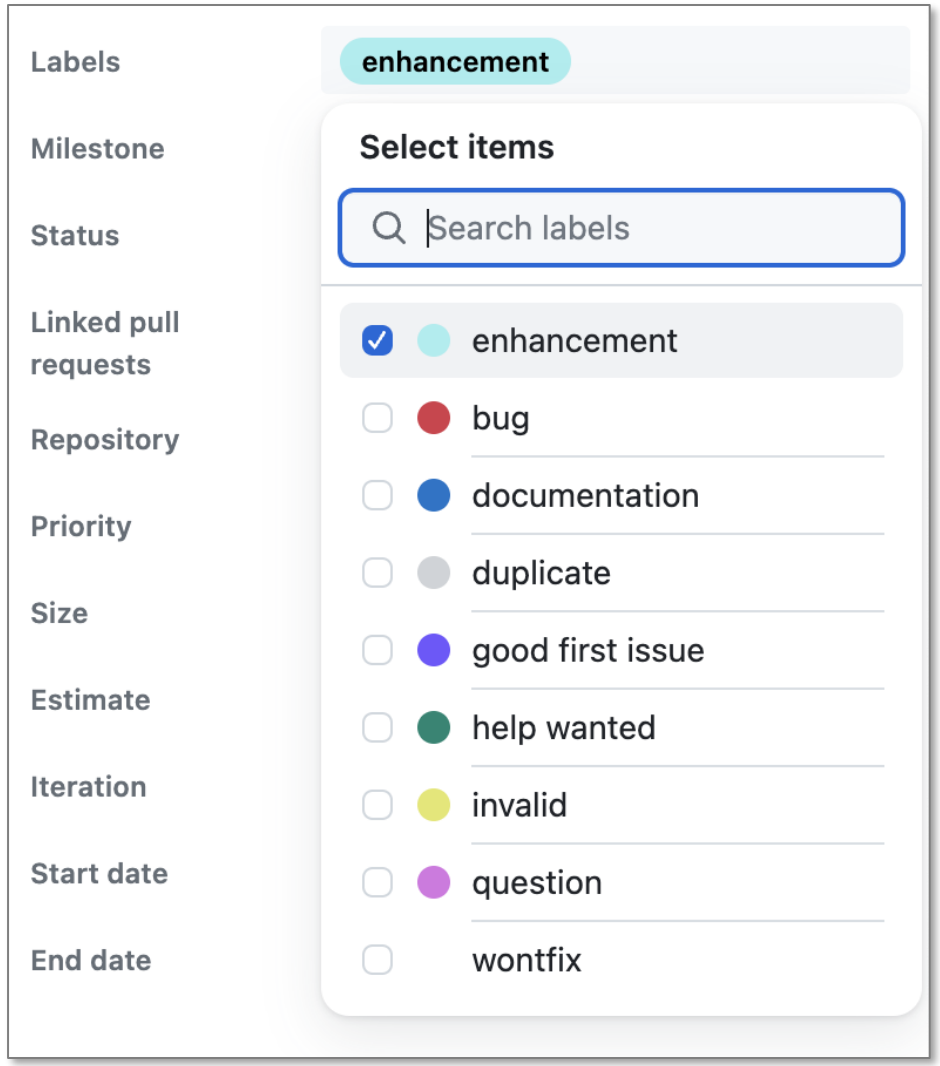


Search Syntax	Description
language:python	Finds repositories written in Python.
stars:>100	Repositories with more than 100 stars.
forks:>=50	Repositories forked at least 50 times.
size:<1000	Repositories smaller than 1000 KB.
user:username	Repositories owned by a specific user.
repo:username/repository	Specific repository from a user.
filename:file.ext	Search for a specific file.
path:/docs/	Files in a specific directory.
extension:py	Files with a .py extension.
is:issue is:open	Open issues.
is:pr is:closed	Closed pull requests.

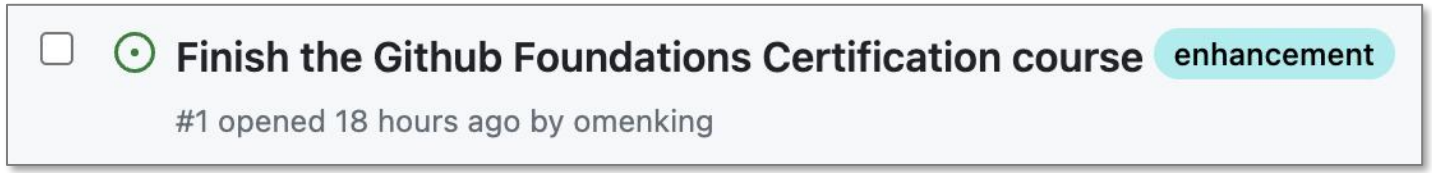
Search syntax also supports : **regular expressions, wildcards, NOT, OR AND, multiple terms and more**

Labels

GitHub Labels are used to **categorize** issues, pull requests, and discussions



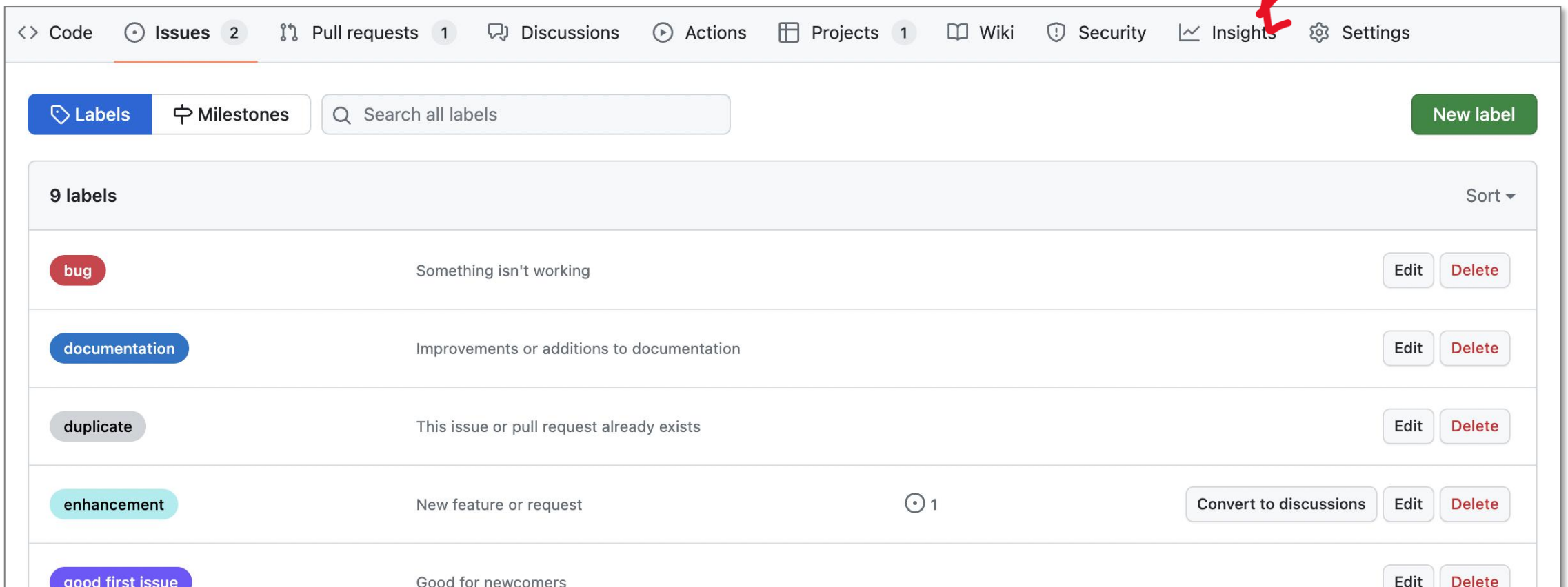
You can **apply multiple Labels** to a labeled item



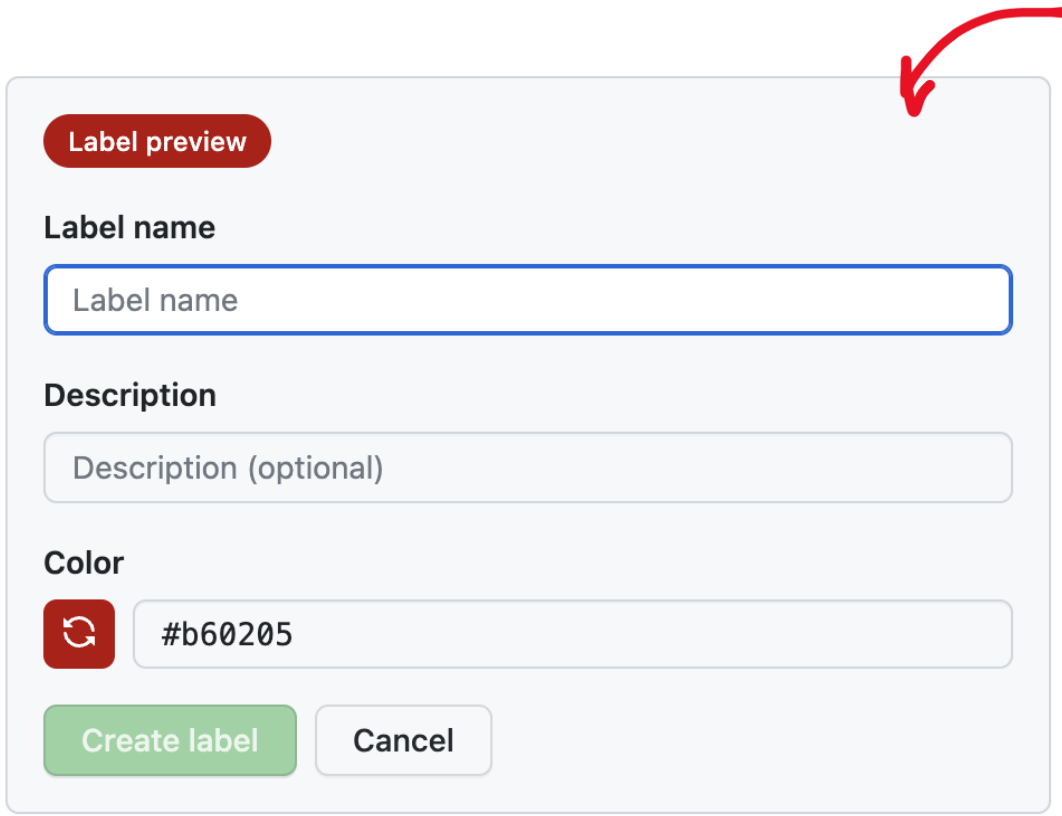
Labels

You can **manage your labels** or create new labels on the Labels page for your repo

https://github.com/<username/org>/<repo-name>/labels



Labels



Creating a label as simple as providing a:

- Name
- Description
- Colour

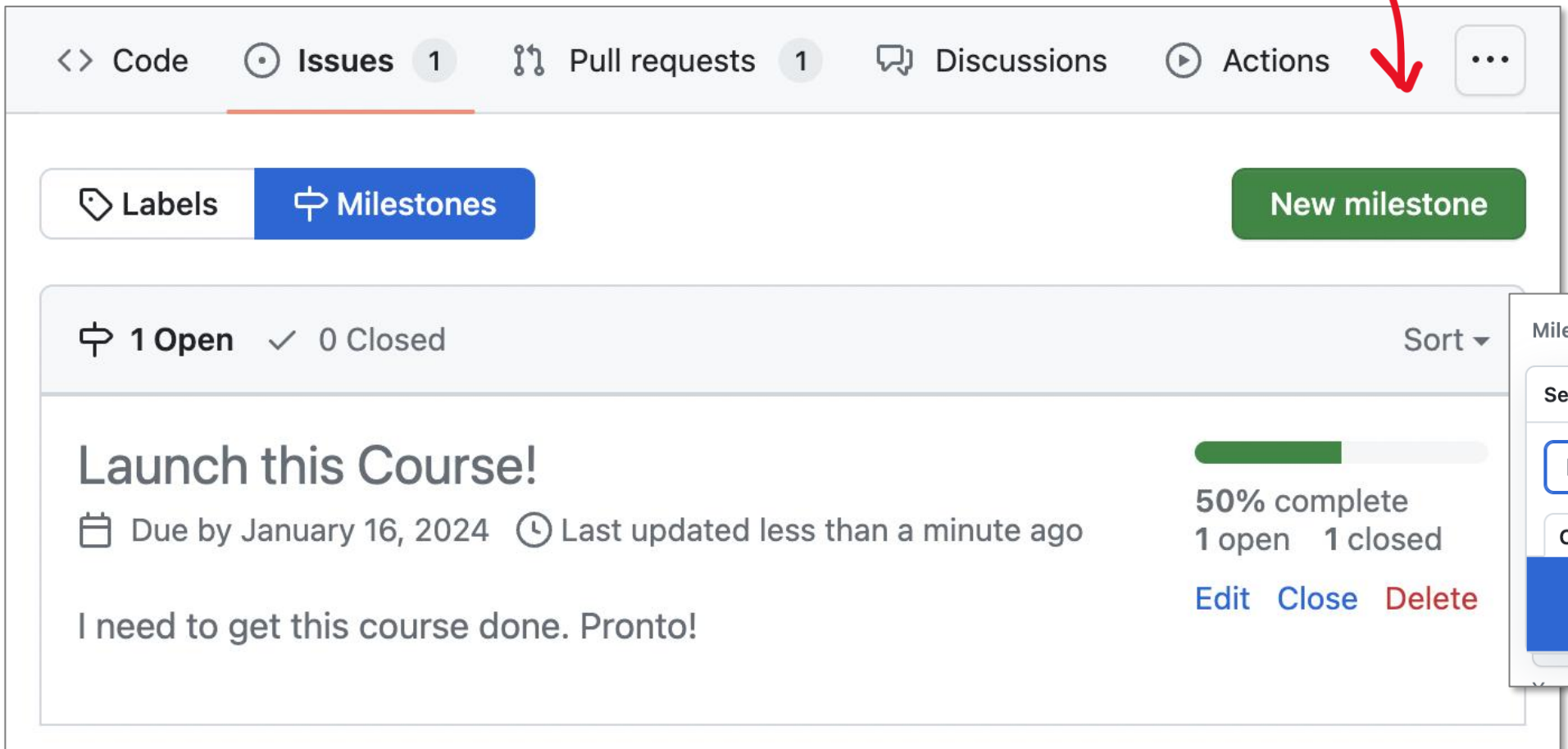
Github CLI provides commands for labels

- gh label clone
- gh label create
- gh label delete
- gh label edit
- gh label list

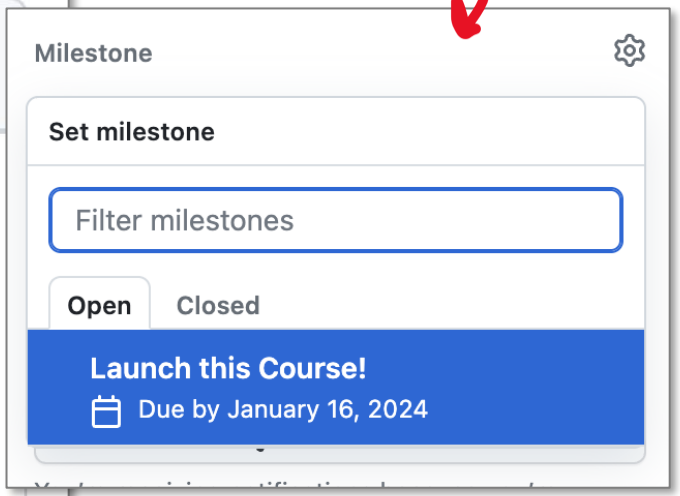
```
gh label create bug \
--description "Something isn't working" \
--color E99695
```

Milestones

Milestones allow you to **group multiple issues into an end goal** which show completion towards the goal for each closed Issued



You **associate** Issues to Milestones



Introduction to GitHub Projects

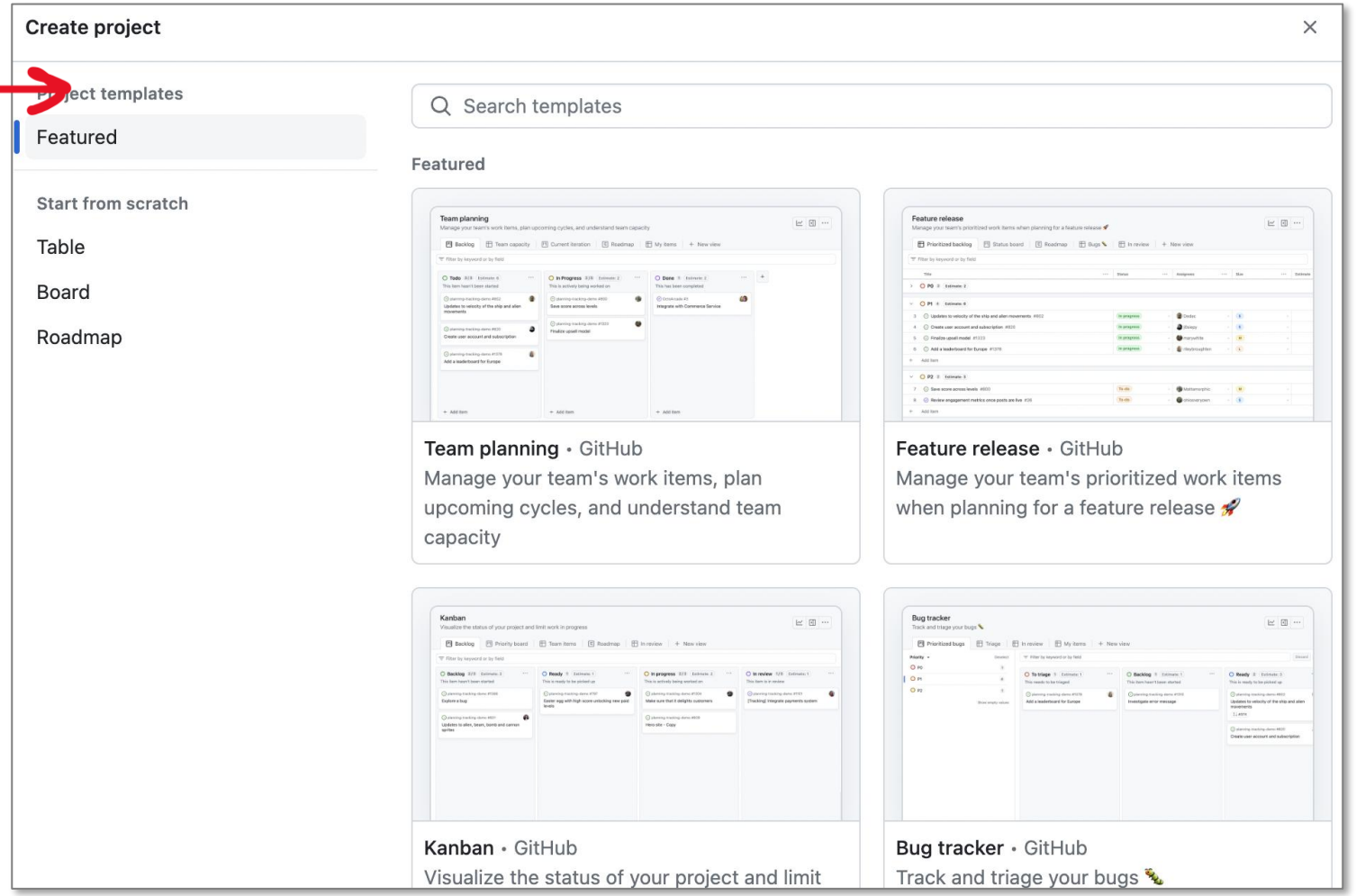
GitHub Projects is a **planning and tracking tool** when on working on GitHub Repo.

A project has an adaptable view that can be changed at anytime between:

- Spreadsheet (Table)
- Taskboard (Board)
- Roadmap (Roadmap)

Projects **directly integrate** with Issues and Pull Requests.

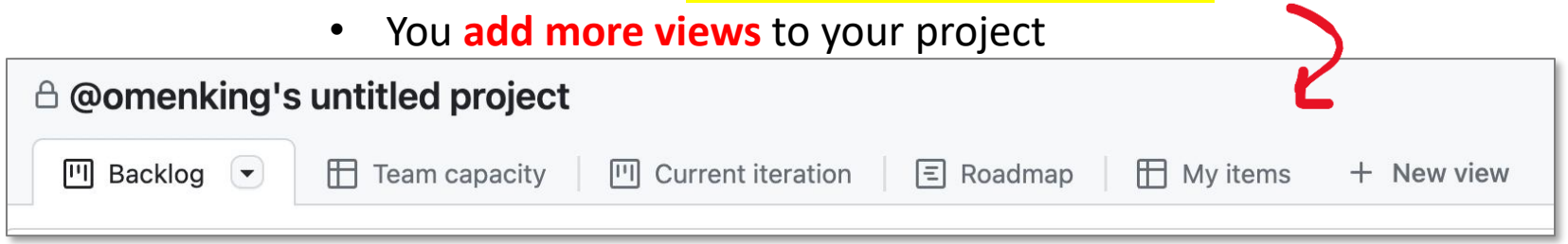
Projects have built-in workflows to **automate common actions**.



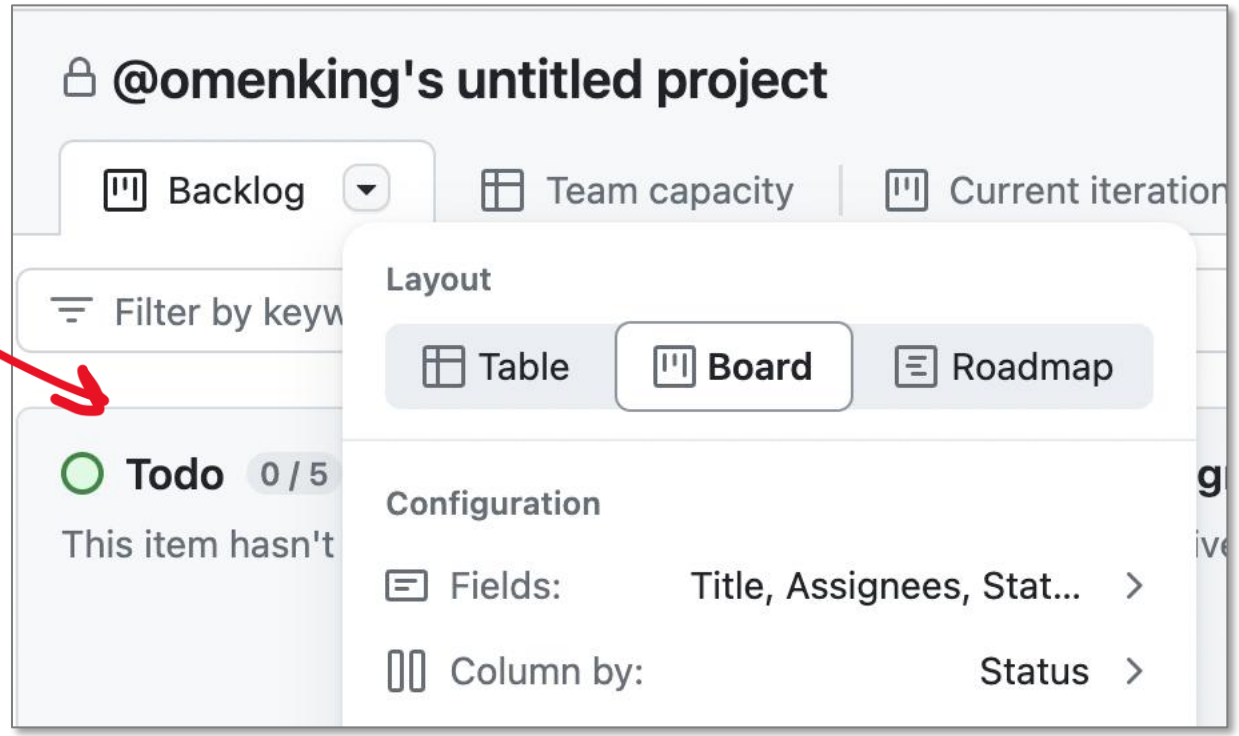
GitHub Projects Layout Options

A GitHub Project is **composed of multiple views**

- You **add more views** to your project

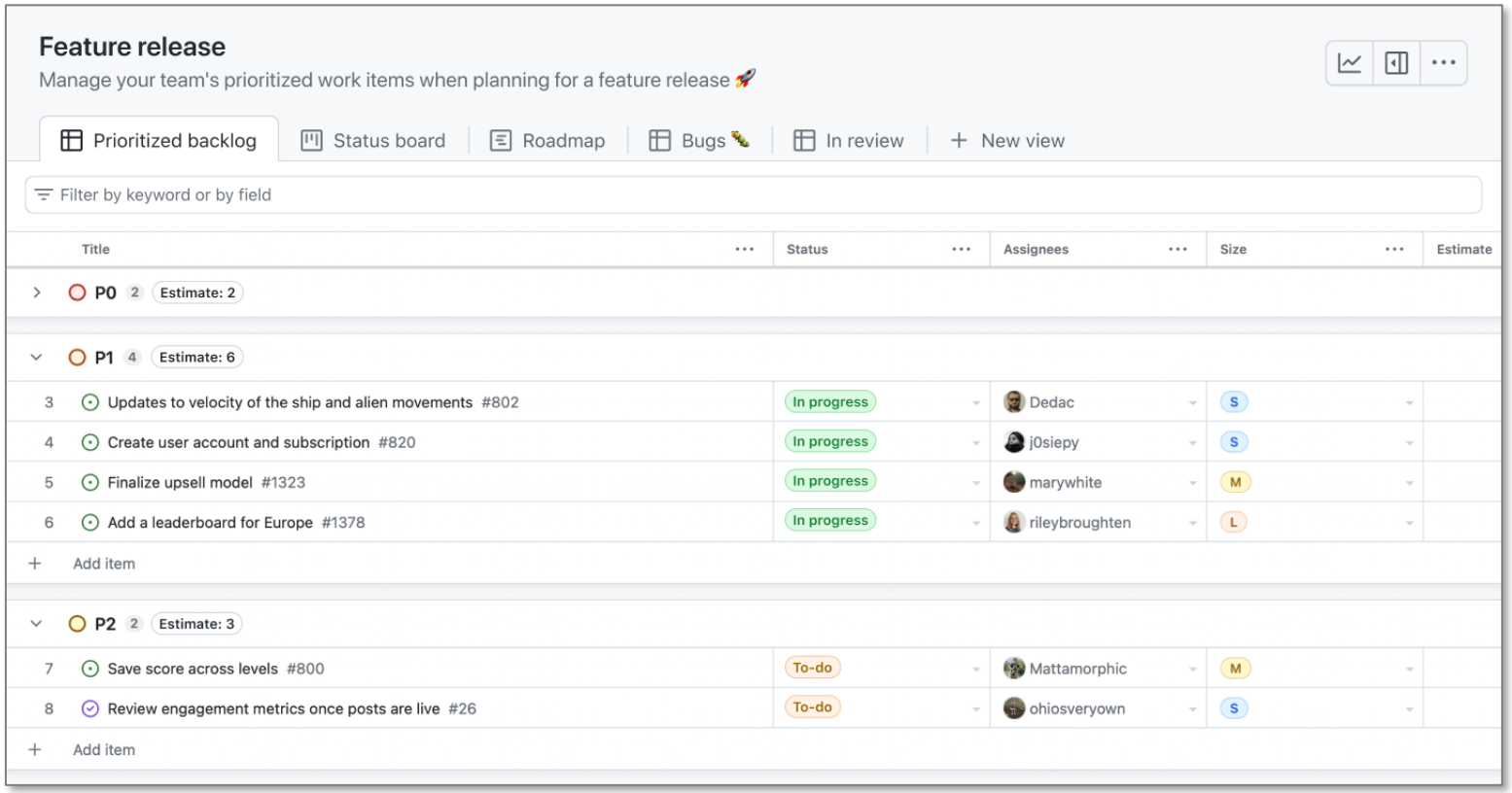


Each **view's layout can be changed at anytime** to accommodate the project use-case:



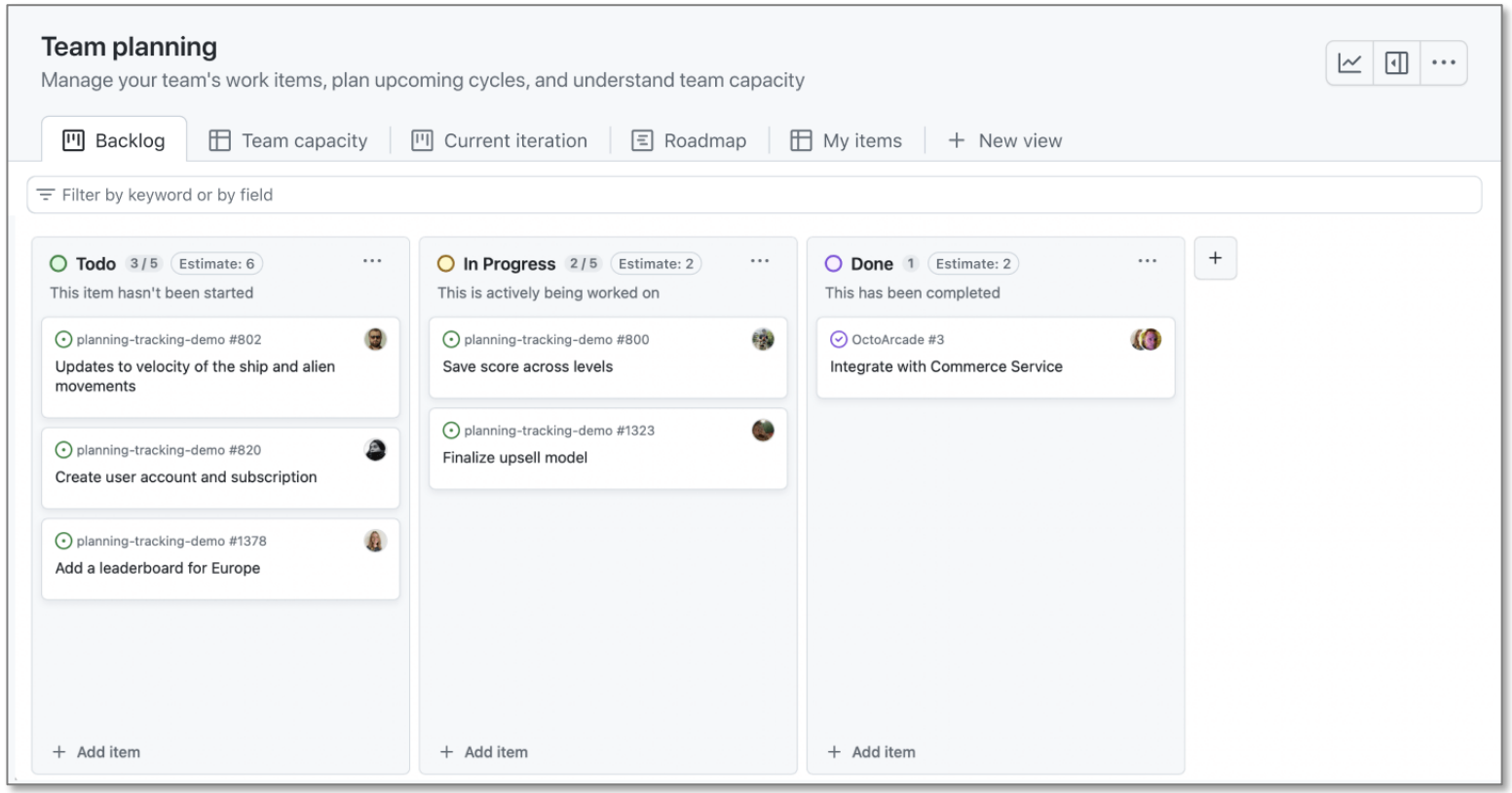
GitHub Projects Layout Options

Table layout is great for your traditional ticket tracker and large amount of tasks



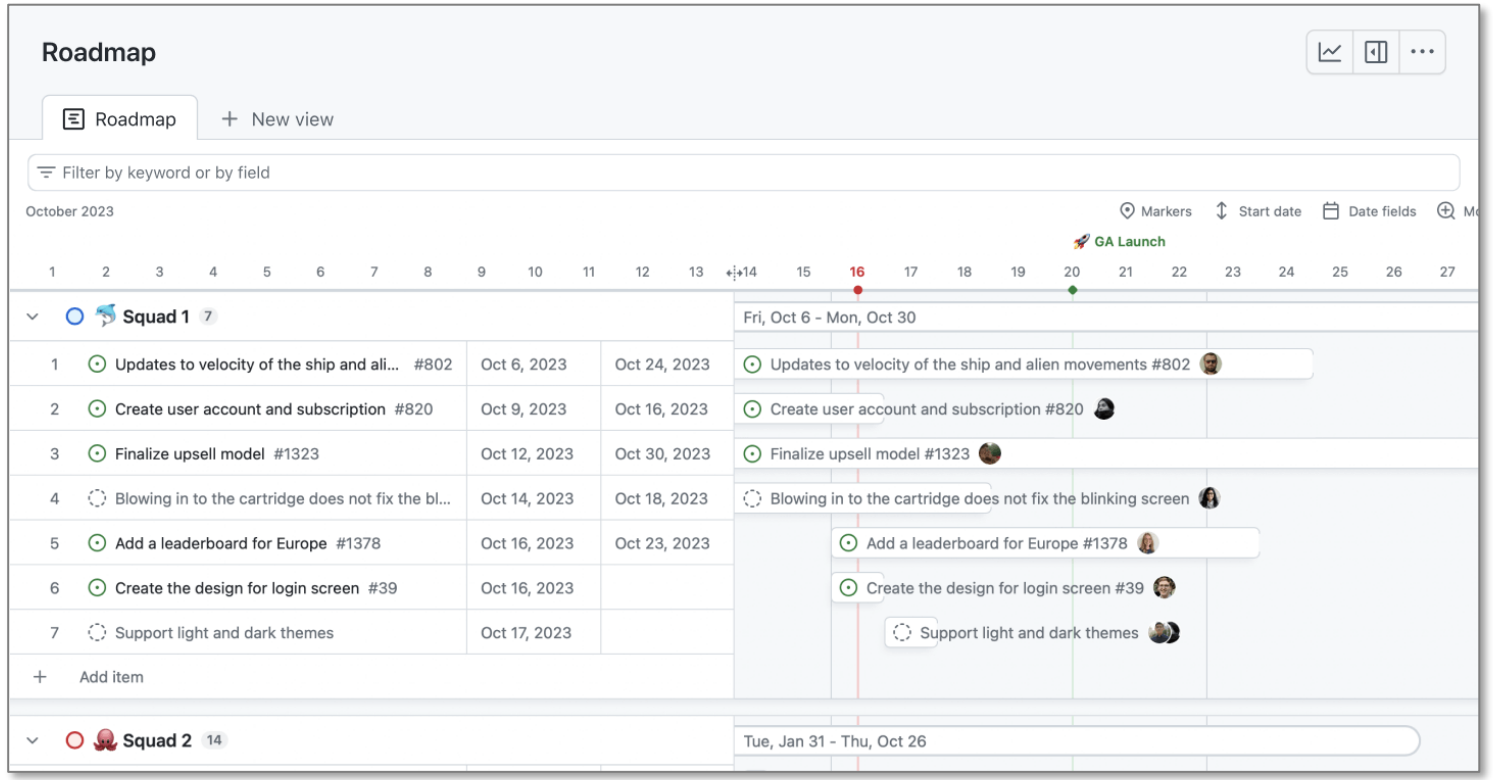
GitHub Projects Layout Options

Board layout is great for your agile tasks and understand clearly the state of Issues



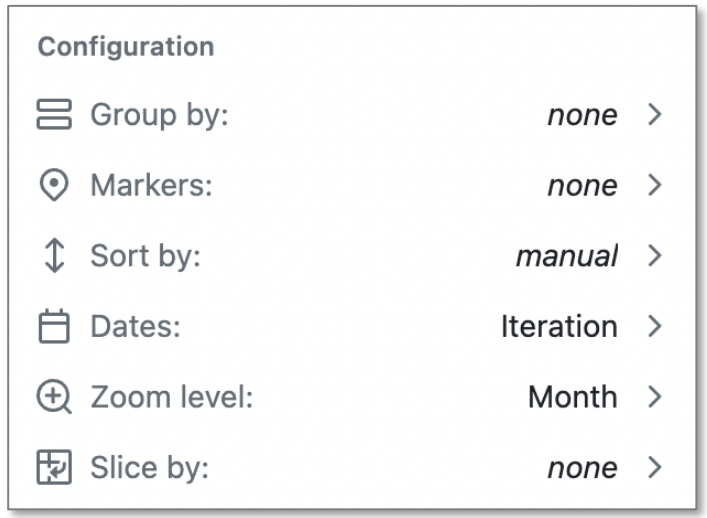
GitHub Projects Layout Options

Roadmap layout is great for For planning based on a timeline



GitHub Projects Layout Options

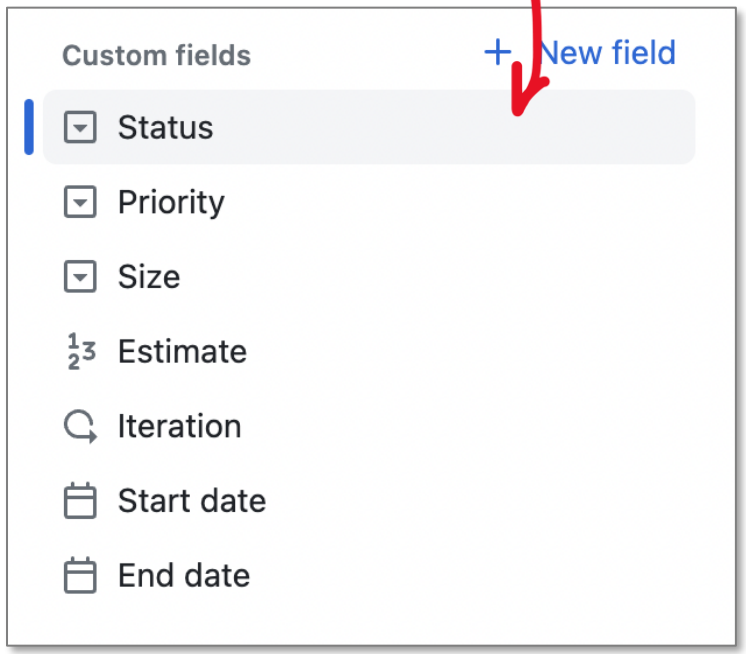
For your layouts you can **configure your data** based on multiple options



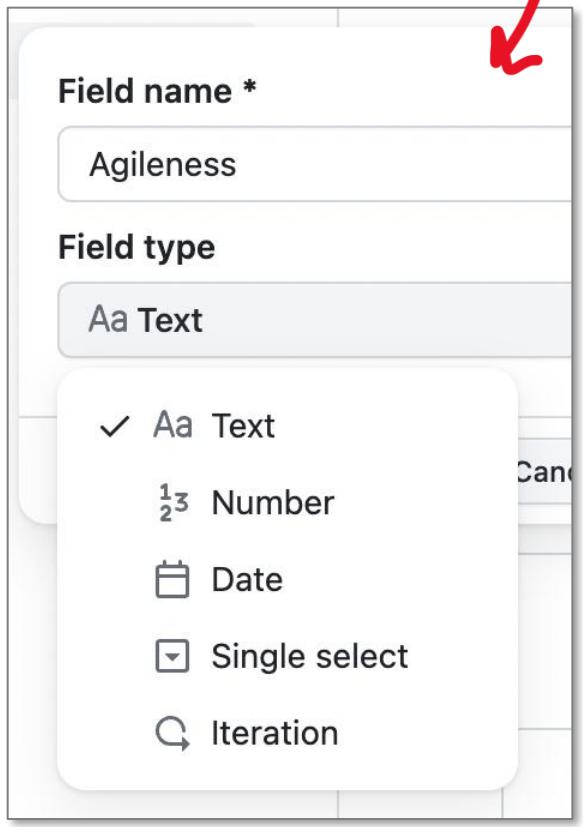
- **Fields:** Add custom data fields to each project item (e.g., status, assignees).
- **Column** by: Organize board columns based on specific fields (e.g., status).
- **Group by:** Group items into rows based on a chosen field (e.g., assignee, label).
- **Sort by:** Define the order of items within columns or groups (e.g., by priority).
- **Fields sum:** Show a summary or total of a particular field for all items in a column/group.
- **Slice by:** Filter visible items on the board by a specific field (e.g., milestone, label).
- **Zoom Level (Roadmap only):** Adjust the view of the roadmap timeline (daily, monthly, quarterly).
- **Dates (Roadmap only):** Set and view start and end dates for each item on the roadmap.

GitHub Projects Configuration Options

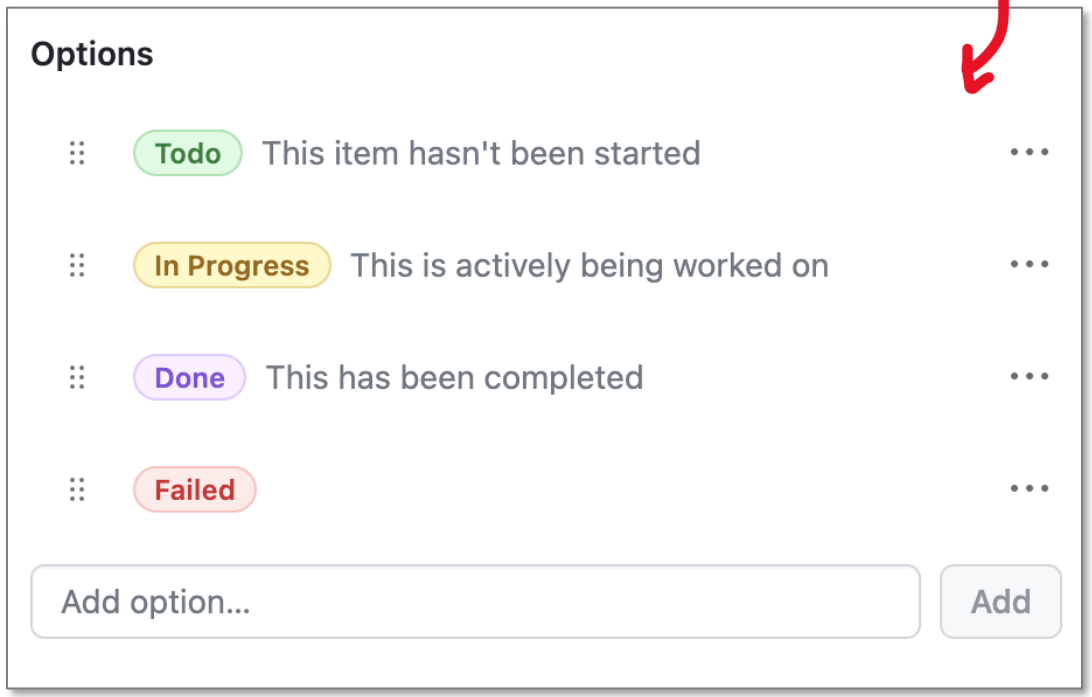
Your GitHub Project Items have **custom fields**



You can create **new fields**



And provide **options** to easily fill in the fields.



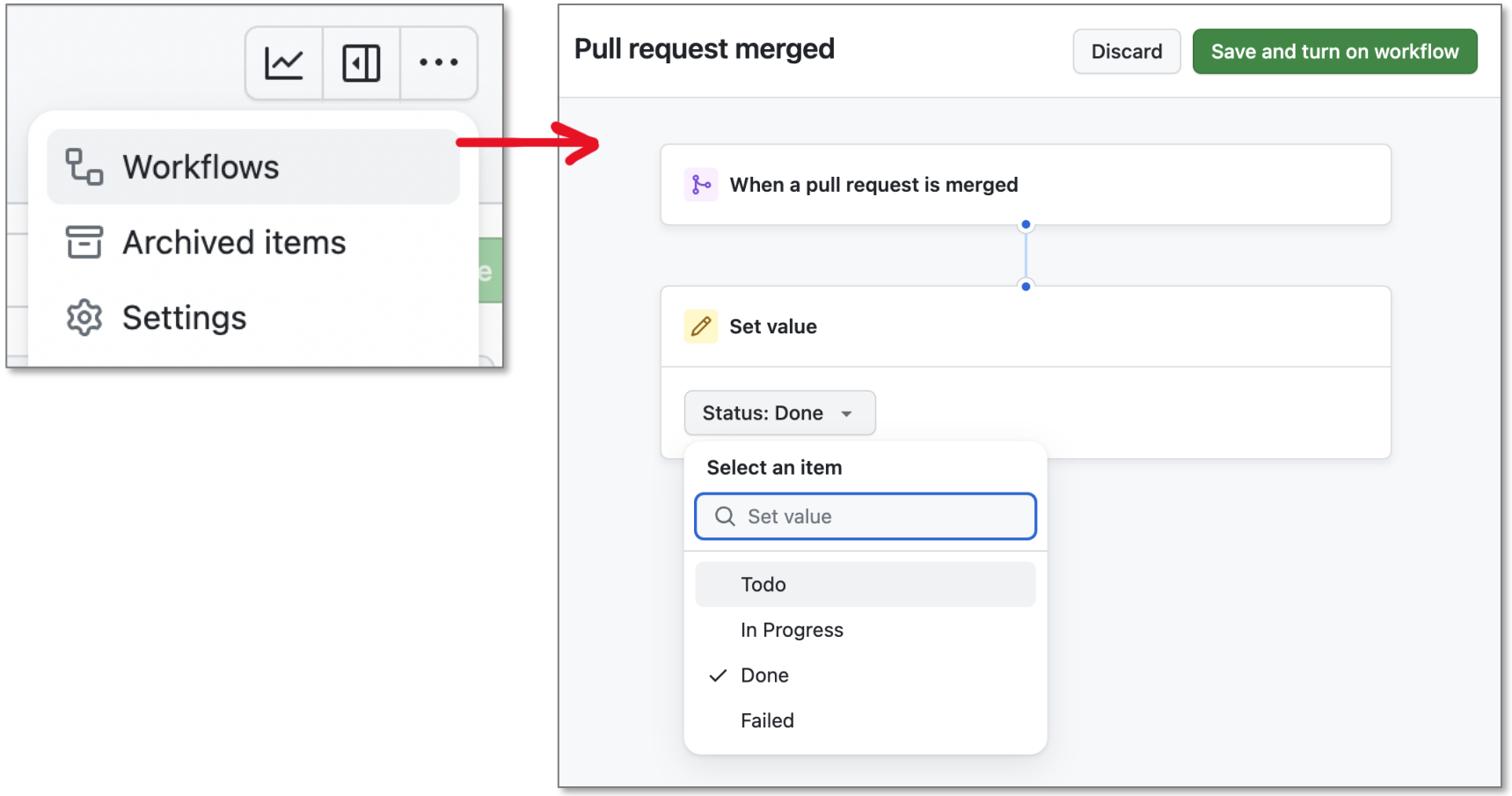
Projects vs Projects Classic

	GitHub Projects	GitHub Projects *Classic*
Interface	New, more dynamic interface with tables, boards, and views	Traditional board view similar to Trello
Flexibility	Highly customizable with different views and fields	Limited customization options
Automation	Supports automated workflows and field updates	Basic automation capabilities
Reporting	Enhanced reporting features and insights	Basic reporting and tracking
Integration	Deeper integration with GitHub issues and pull requests	Basic integration with GitHub repositories

GitHub Projects offers more advanced and flexible project management tools compared to GitHub Classic Projects, which is simpler and more straightforward.

Project Workflows

GitHub built-in Project Workflows allows you to **automate what happens based on specific events**

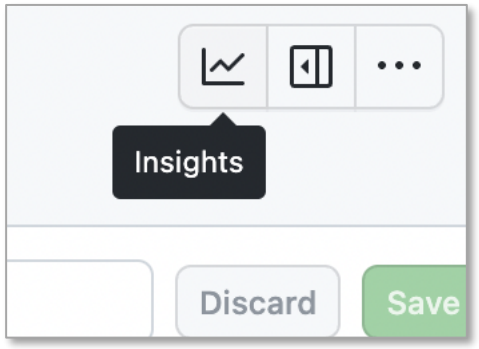


Built-in Workflows

- Item added to project
- Item reopened
- Item closed
- Code changes requested
- Code review approved
- Pull request merged
- Auto-archive items
- Auto-add to project

For more advanced automation of GitHub Projects, you can use GitHub Actions

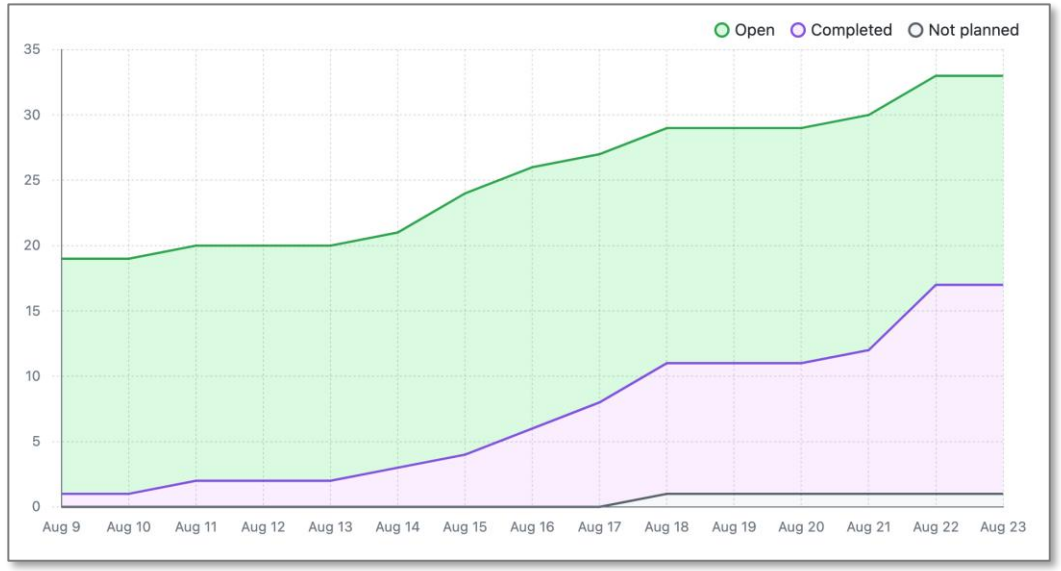
Project Insights



Project Insights lets you **create charts about your GitHub Project**.

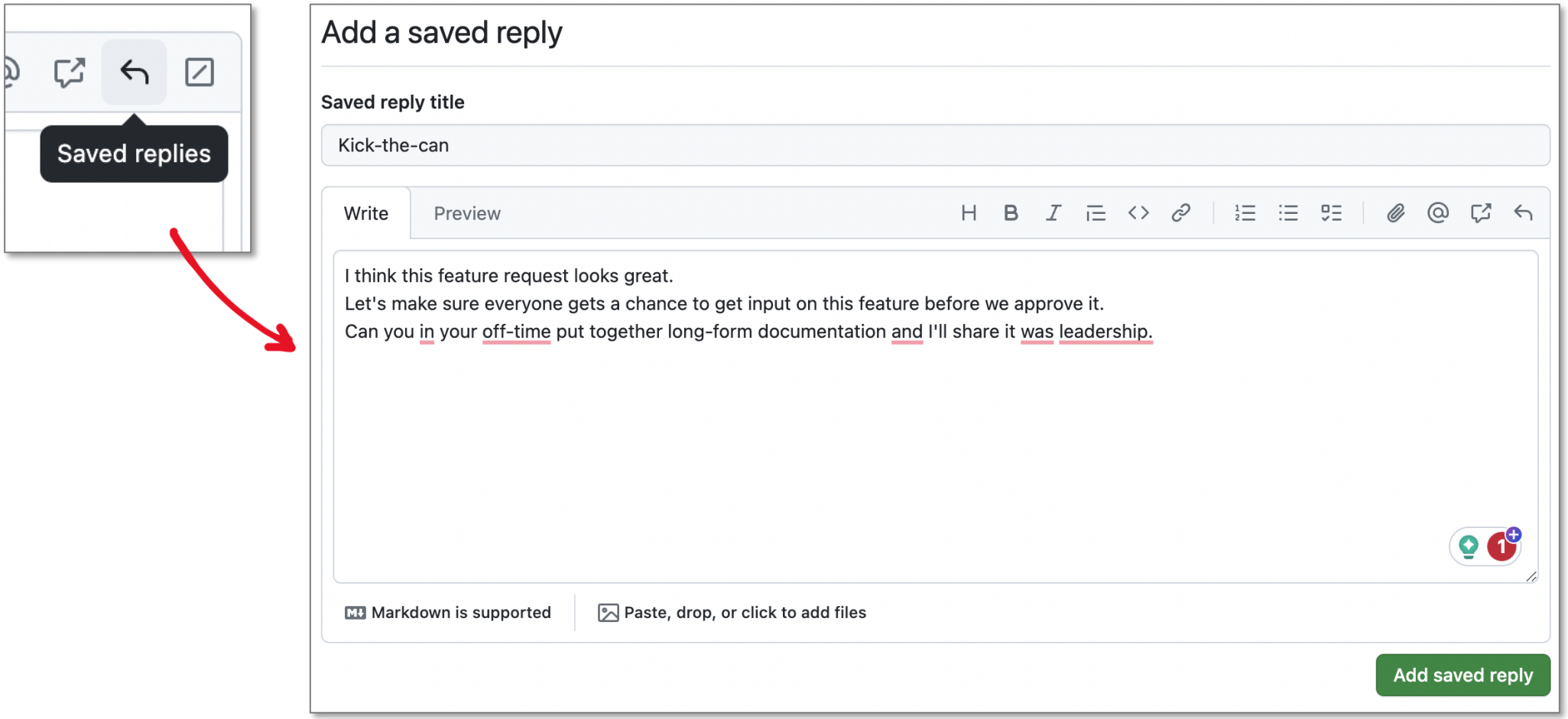
Current Charts
eg. how many items are assigned to each individuals

Historical Charts
track changes to the state of your project items

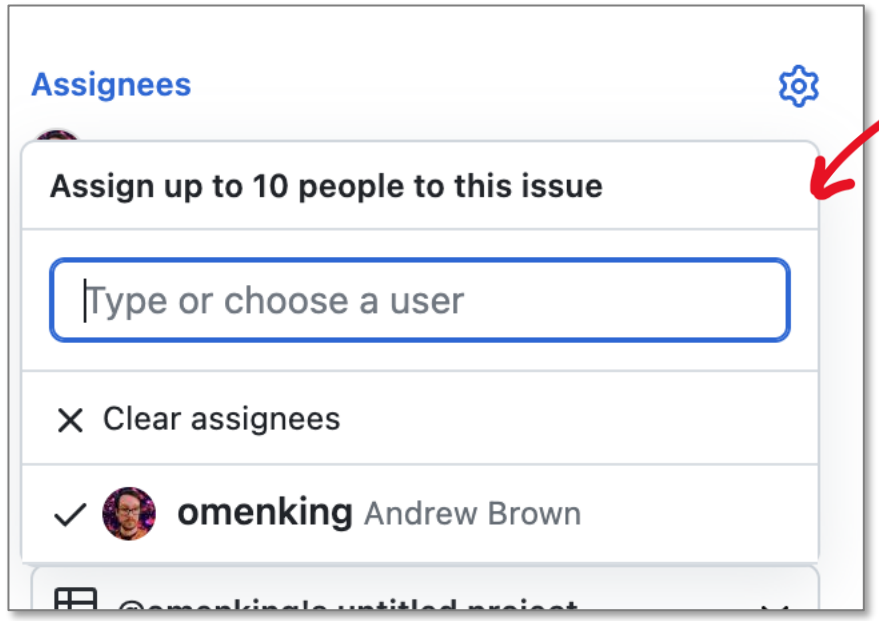


Managing Saved Replies

When commenting on an issue or pull request, you can **add a saved reply that you've already set up**.

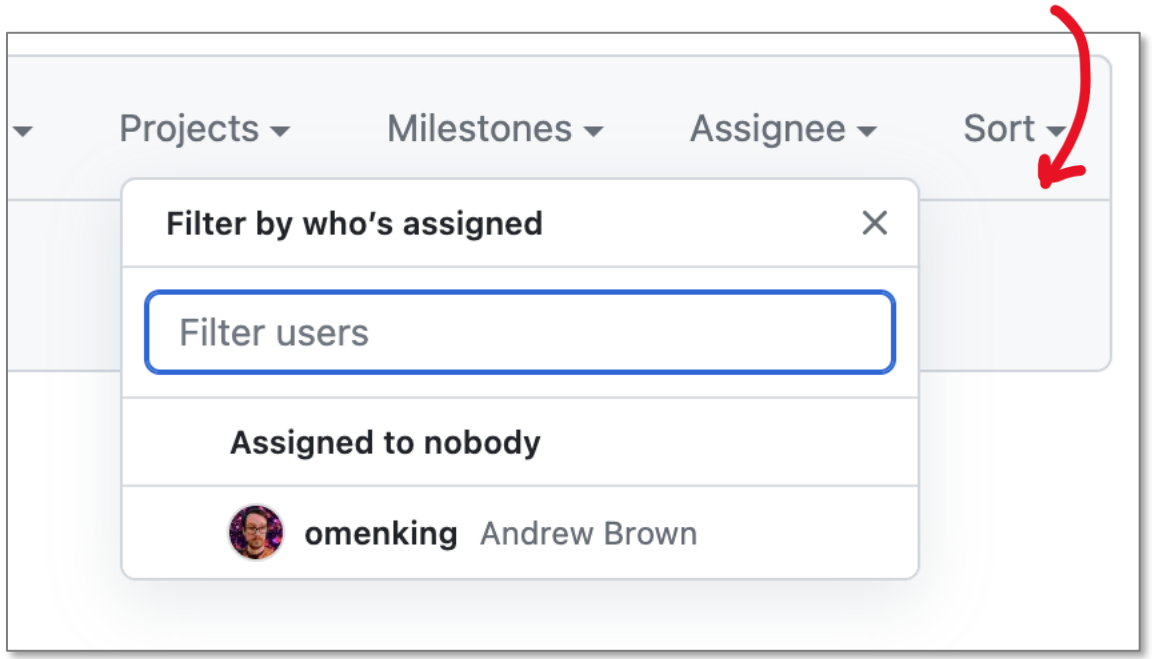


Assigning Issues and Pull Requests



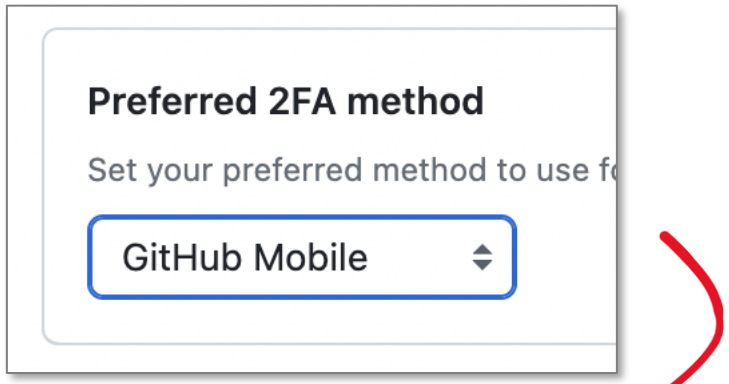
You can **assign issues** and pull requests to specific users.
That's it.

You can **filter** Issues and PRs based on assignees.



Securing Account with 2FA

2-Factor Authentication (2FA) adds an extra layer of security by requiring a second device (e.g. phone) to confirm your identity during sign-in or sensitive actions.



Choose your preferred **UFA (User-Facing Authentication)** method:

- GitHub Mobile
- SMS/Text message
- Authenticator app

You can download third-party authenticator apps via your phone

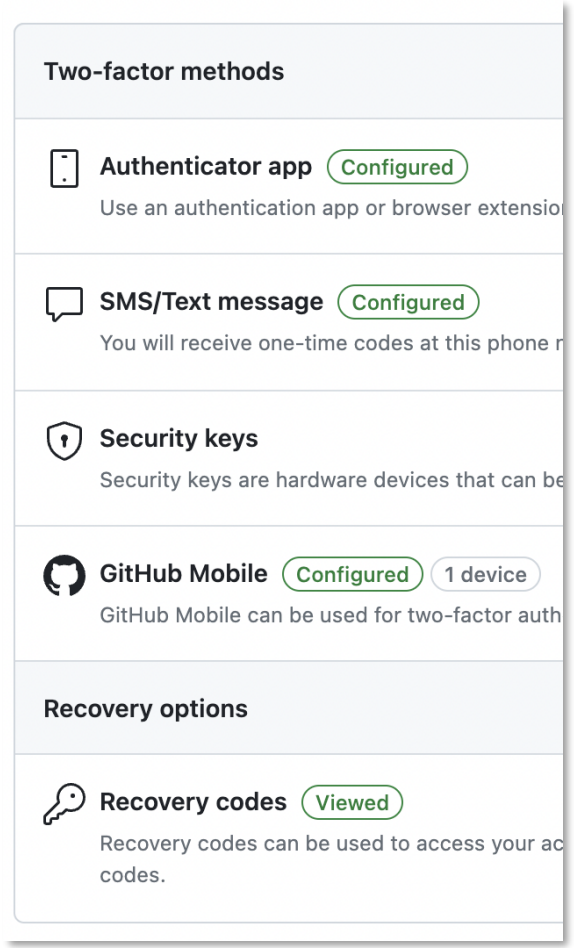
- Authy
- Google Authenticator
- Microsoft Authenticator

Get UFA code sent as text message

Use **hardware security keys** like YubiKeys

GitHub's Mobile app

Recovery codes are long secret codes you need to print and store somewhere safe in case you get locked out.



Access Permissions – Personal Accounts

Personal account has **two permission levels**:

- **Repository owner –** the person who owns the account, that's you!
  - Repositories owned by personal accounts have one owner
  - Ownership permissions can't be shared with another personal account.
- **Collaborators** – people you add as collaborators

Repository owner actions — has full control of the repository

- Invite collaborators
- Change the visibility of the repository
- Limit interactions with the repository
- Rename a branch, including the default branch
- Merge a pull request on a protected branch, even if there are no approving reviews
- Delete the repository
- Manage the repository's topics
- Manage security and analysis settings for the repository
- Enable the dependency graph for a private repository
- Delete and restore packages
- Customize the repository's social media preview
- Create a template from the repository
- Control access to Dependabot alerts
- Dismiss Dependabot alerts in the repository
- Manage data use for a private repository
- Define code owners for the repository
- Archive the repository
- Create security advisories
- Display a sponsor button
- Allow or disallow auto-merge for pull requests
- Manage deploy keys
- Manage webhooks

Access Permissions – Personal Accounts

Collaborators actions — can pull (read) the contents of the repository and push (write) changes to the repository

- Fork the repository
- Rename a branch other than the default branch
- Create, edit, and delete comments on commits, pull requests, and issues in the repository
- Create, assign, close, and re-open issues in the repository
- Manage labels for issues and pull requests in the repository
- Manage milestones for issues and pull requests in the repository
- Mark an issue or pull request in the repository as a duplicate
- Create, merge, and close pull requests in the repository
- Enable and disable auto-merge for a pull request
- Apply suggested changes to pull requests in the repository
- Create a pull request from a fork of the repository
- Submit a review on a pull request that affects the mergeability of the pull request
- Create and edit a wiki for the repository
- Create and edit releases for the repository
- Act as a code owner for the repository
- Publish, view, or install packages
- Remove themselves as collaborators on the repository

Access Permissions – Personal Accounts

Repository owner actions

has full control of the repository

VS

Collaborators actions

can pull (read) the contents of the repository and push (write) changes to the repository

Access Permissions – Organization Accounts

Organization members have the permission levels

- **Owner**
  - complete administrative access to your organization
  - This role should be limited, but to no less than two people, in your organization
- **Billing manager** — can manage billing settings
- **Member roles** — the default role for everyone else

<Orgnaziation roles confict, this makes no sense maybe ignore this slide for now>

EMUs

Enterprise Managed Users (EMUs) allows you to **manage the lifecycle and authentication of your users on GitHub.com** from an **external identity management system**, or IdP.

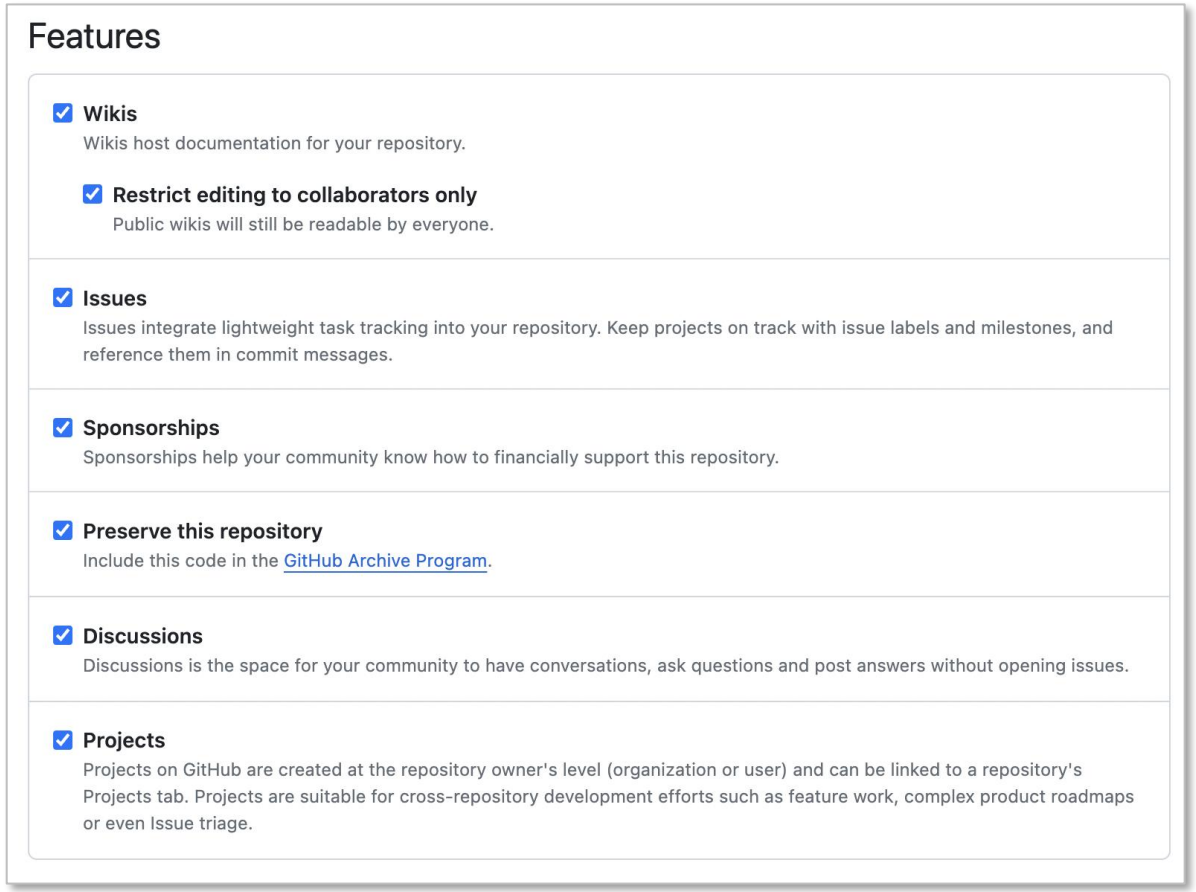
- GitHub partners with some developers of identity management systems to provide a "paved-path" integration with EMUs
- These IdPs mostly provide authentication using SAML.
- Microsoft Entra ID (aka Azure AD) also offers OIDC for authentication.
- The IdP applications provision users with System for Cross-domain Identity Management (SCIM).

Partner IdP	SAML	OCID	SCIM
Microsoft Entra ID	✓	✓	✓
Okta	✓	✗	✓
PingFederate	✓	✗	✓

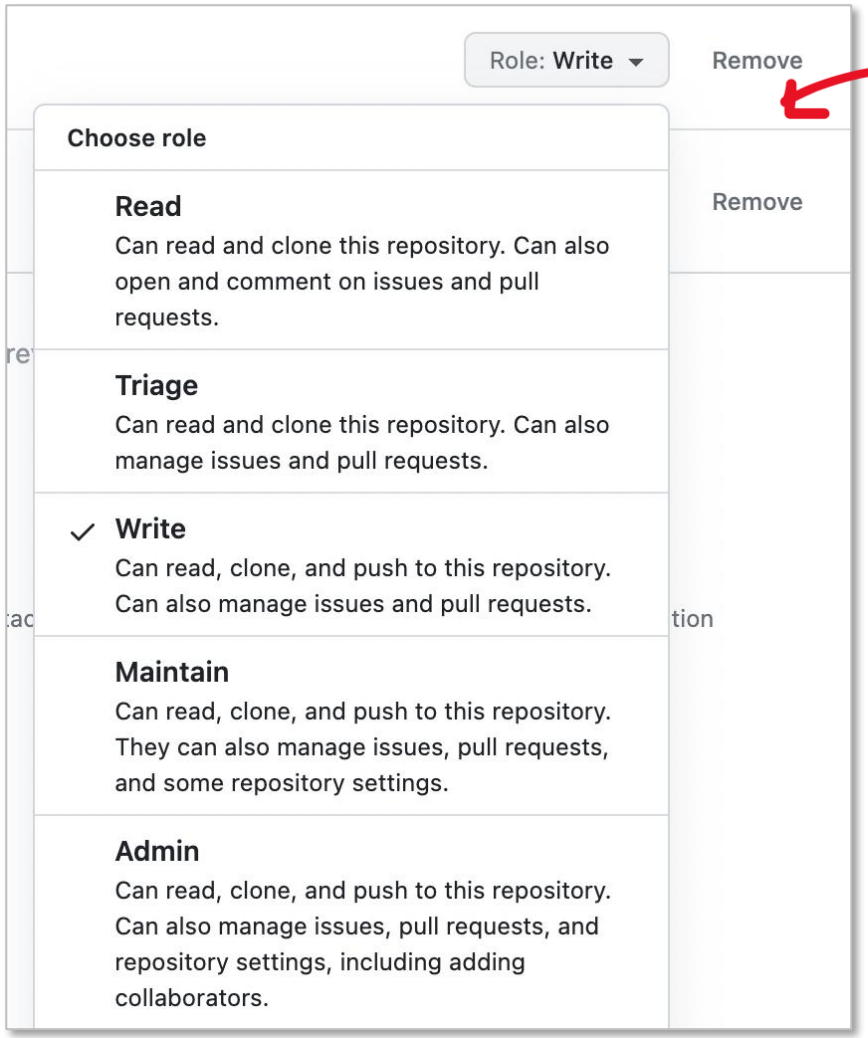
Managing Features

GitHub Repo have **multiple features we can enable or disable:**

- Wikis
- Issues
- Sponsorships
- Preserve this repo
  - Goes into GitHub Arctic Code Vault
- Discussions
- Projects
- and possibly more…



Repo permission levels



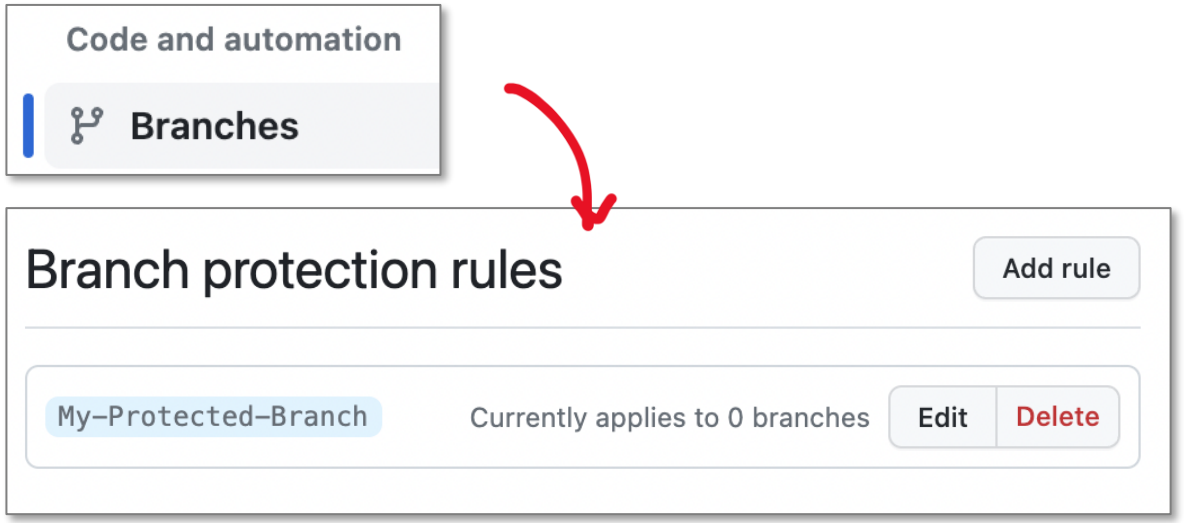
When you add a collaborator (and they accept the invite) you can choose from **predefined roles** with different level of access

- **Read**:
  - View and clone the repository.
  - Open and comment on issues and pull requests.
  - Download releases.
- **Triage**:
  - Manage issues and pull requests without write access.
  - Label and assign issues and pull requests.
  - Close and reopen issues and pull requests.
- **Maintain**:
  - Push to the repository.
  - Manage issues, pull requests, labels, and project boards.
  - Create and publish releases.
  - Configure repository settings for non-sensitive fields (like assigning collaborators).
- **Admin**:
  - Full control over the repository.
  - Push to, delete, or archive the repository.
  - Change repository settings, including sensitive fields like security or billing.
  - Manage collaborators and their permissions.
  - Access to all functionalities provided in the Maintain role.

Enterprise accounts can have custom roles.

Branch Protection Rules

Branch protection rules are used to **enforce certain workflows or requirements** before changes can be merged into a branch.



Protect matching branches rules

- Require a pull request before merging
- Require status checks to pass before merging
- Require conversation resolution before merging
- Require signed commits
- Require linear history
- Require deployments to succeed before merging
- Lock branch
- Do not allow bypassing the above settings

Rules applied to everyone including administrators

- Allow force pushes
- Allow deletions

Security Tab features and options

The security tab will present a **repo security checklist** of security options to configure

Security policy
Create markdown file that explains how security vulnerabilities should be reported

Security advisories
privately discuss, fix, and publish information about security vulnerabilities in your public repository

Private vulnerability reporting
Allow your community to privately report potential security vulnerabilities to maintainers and repository

Dependabot alerts
A bot that alerts of you of vulnerabilities due to out-of-date dependencies.
Dependabot can automatically create PRs to update dependencies for you to approve.

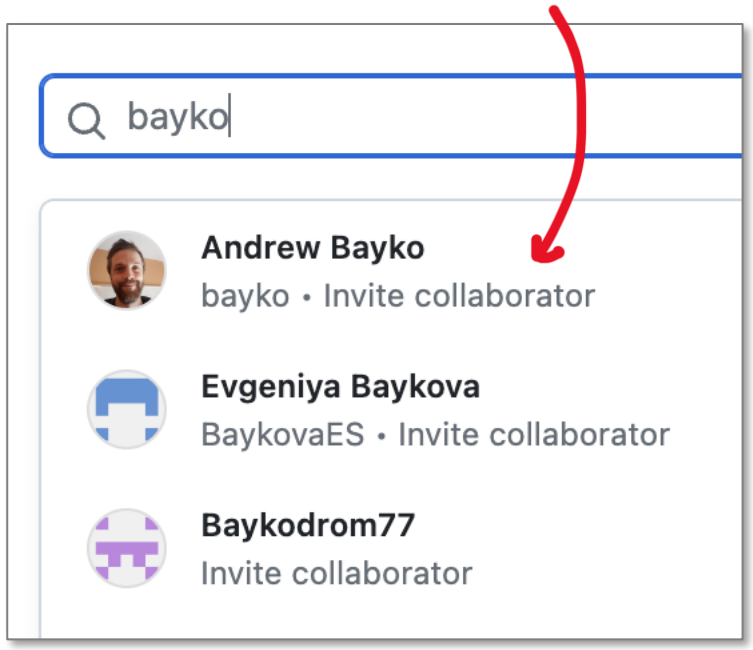
Code scanning alerts
Automatically detect common vulnerabilities and coding errors. Via CodeQL or third-party tools

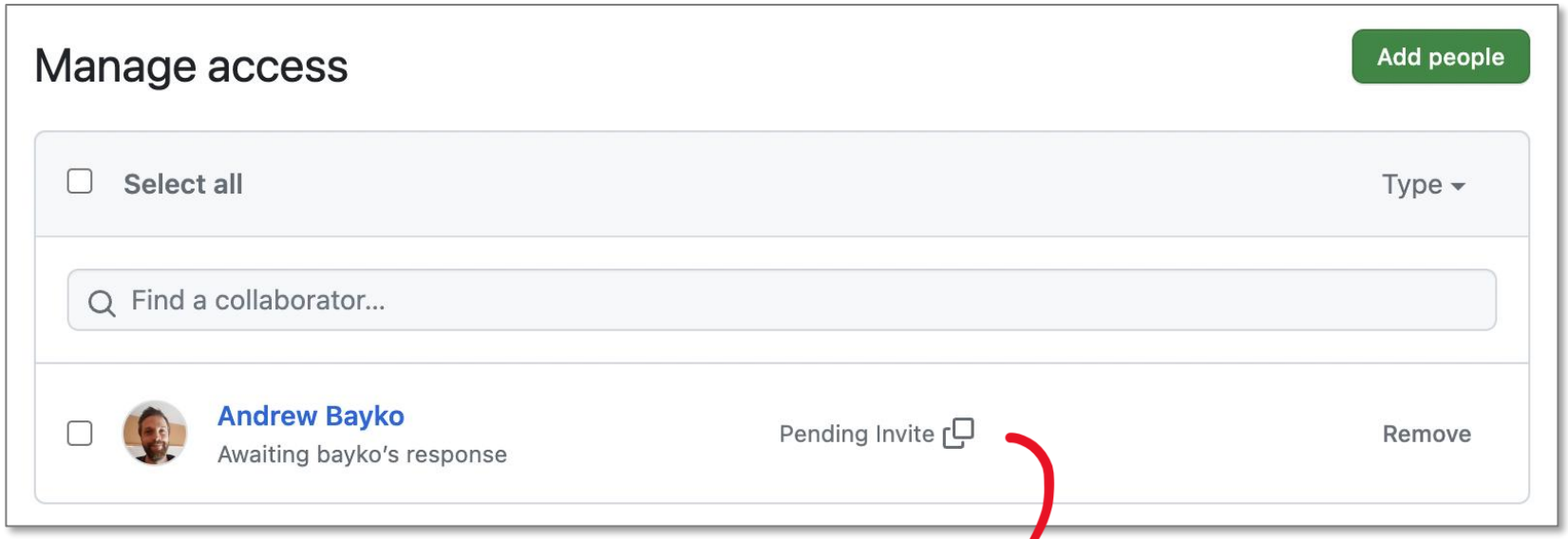
Secret scanning alerts
Get notified when a secret is pushed to this repository

Managing Collaborators

Collaborators allows you let other GitHub users have access to your repo based on the permission levels you provide

You search their **username** to invite



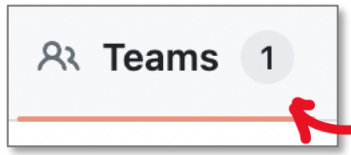


https://github.com/<username/org>/<repo-name>/invitations

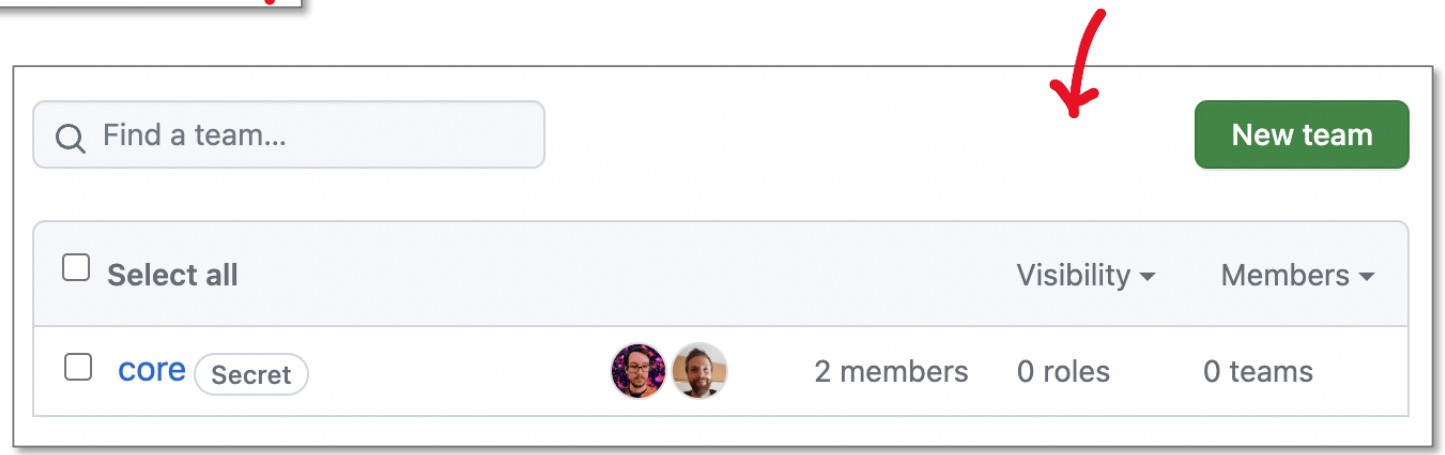
The user need to **accept** the invite.

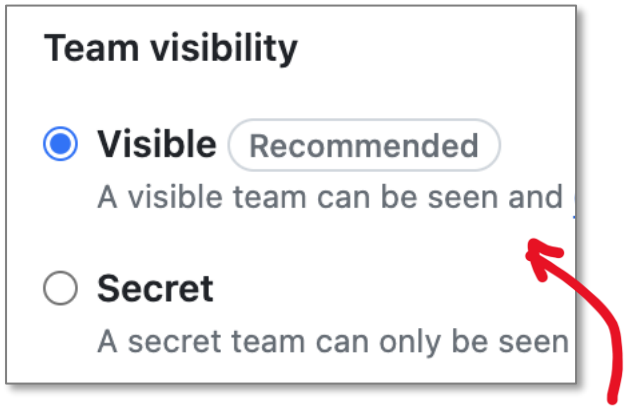
Its best to share the link directly and manually remind them to accept

Organization Teams Tab



An organization can group organization members in **teams**.





Teams can be **public** or **private**

- Teams can be assigned to projects and issue
- Request reviews from teams
- Teams can be mentioned in discussions, issues, and pull requests
- Control team access to repositories
- New members can be added to teams, instantly giving them access to all relevant repositories and discussions

GitHub Connect

GitHub Connect enhances GitHub Enterprise Server (GHES) by allowing your GHES instance **access some of GitHub.com cloud only offerings**

You can **individually enable these features**:

- Automatic user license sync
- Dependabot
- GitHub.com actions
- Server Statistics
- Unified search
- Unified contributions

GitHub Internal Repos

GitHub Enterprise Cloud accounts have a third special GitHub Repo type called **Internal**

- All enterprise members have **read permissions** to the internal repository
- Internal repositories are not visible to people outside of the enterprise, including outside collaborators on organization repositories.
- Unless your enterprise uses Enterprise Managed Users (EMUs), members of the enterprise can fork any internal repository owned by an organization in the enterprise.

Internal repositories are the **default setting** for all new repositories created in an organization owned by an enterprise account.

GitHub Advanced Security

GitHub makes **extra security features available** to customers under a GitHub Advanced Security (GHAS) license.

GitHub Advanced Security is available for enterprise accounts on:

- GitHub Enterprise Cloud
- GitHub Enterprise Server
- And some features of for public repositories on GitHub.com

GitHub Advanced Security license provides the following additional features:

	Public repository	Private repository without Advanced Security	Private repository with Advanced Security
Code scanning	✓	✗	✓
Secret scanning	✓	✗	✓
Dependency review	✓	✗	✓

SAML and SCIM

SAML and SCIM is **available for both** GitHub Enterprise Cloud and GitHub Enterprise Server

SAML

Security Assertion Markup Language (SAML) is an open standard for exchanging authentication and authorization between an identity provider and a service provider.
An important use case for SAML is **Single-Sign-On via web browser**.

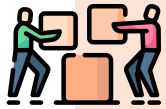
SCIM

System for Cross-domain Identity Management (SCIM) is an open standard for automating the exchange of user identity information across different identity management systems. A key use case for SCIM is to enable scalable and automated user provisioning and deprovisioning, often integrated with enterprise identity services.

SAML is about **securely transmitting user authentication and authorization data** for single sign-on purposes.

vs

SCIM is about **managing user identities across different systems** to simplify account maintenance and provisioning



They can complement each other in an ecosystem where you need both federated identity for authentication and a standardized way to manage user accounts across various systems.