ConductorOne

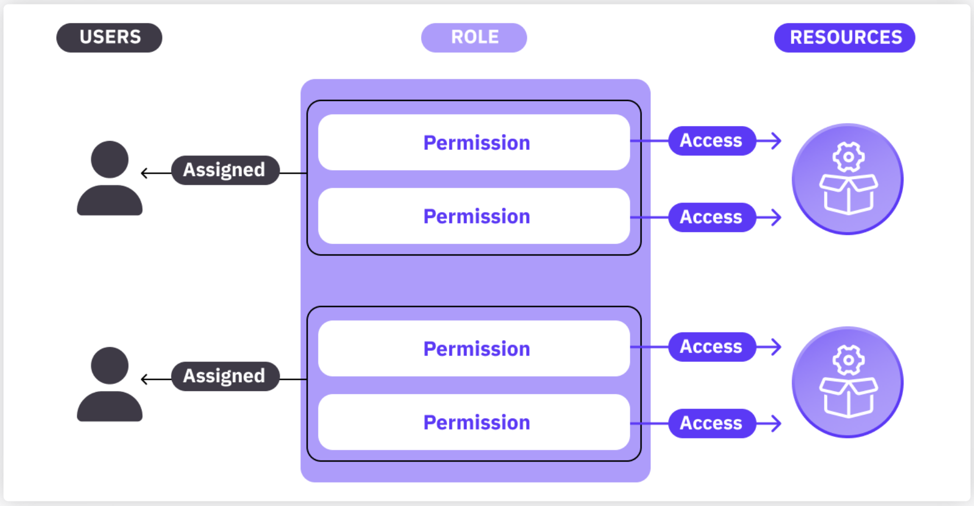
Everything You Wanted to Know about GitHub Access Control

GitHub is the largest and most popular software development platform, providing services from Git version control to bug tracking, CI/CD, and task management. With over 100 million users developing all kinds of software, proper access control on all levels is crucial to the maintenance, security, and integrity of all the code on the platform. That's why GitHub provides an extensive system to manage access control across repositories, teams, and organizations of all sizes.

This article explores everything you need to know about GitHub access control in order to properly manage your GitHub accounts and repositories on all levels.

Understanding GitHub's Authorization/Permission Model

GitHub uses a role-based access control (RBAC) model. With this model, to perform any action on GitHub, a user must have appropriate permissions that grant them the necessary access to a particular resource at the given level. Permissions are collectively assigned to users as roles:



There are different roles for different types of GitHub accounts. For a repository owned by a personal account, the only two available roles are owner and collaborator:

→ An owner has complete control over the repository, with the ability to perform dangerous, destructive actions (like archiving or deleting the repository) and add collaborators.

→ A collaborator has read/write access to the repository's content, with the ability to manage issues, merge pull requests, and create releases, for example.

If you need more granular control, you have to create a GitHub organization, where you'll have access to more organization- and repository-level roles.

At the organizational level, the following roles are available:

→ An owner has complete control over the organization, including all of its repositories and assigned users. To ensure the organization's continuity, it's recommended that it have at least two owners.

→ A member is the base, non-administrative role with default access to repos and projects; its permissions are configurable.

→ A moderator has extended moderation-related permissions in addition to base member role permissions, like blocking non-member users, managing interaction limits, and hiding disruptive comments in public repos.

→ The billing manager role can manage the organization's billing settings, such as the current plan, sponsorships, payment details, or history.

→ The GitHub App manager role can manage some or all GitHub Apps registered by the organization on top of the basic, required member role's permissions.

→ The outside collaborator role allows limited access to selected organization repositories. It has to be further controlled with repository-level permissions.

You can also configure the permissions of individual members, outside collaborators, and entire teams in relation to selected repositories by assigning them one of the repository-level roles:

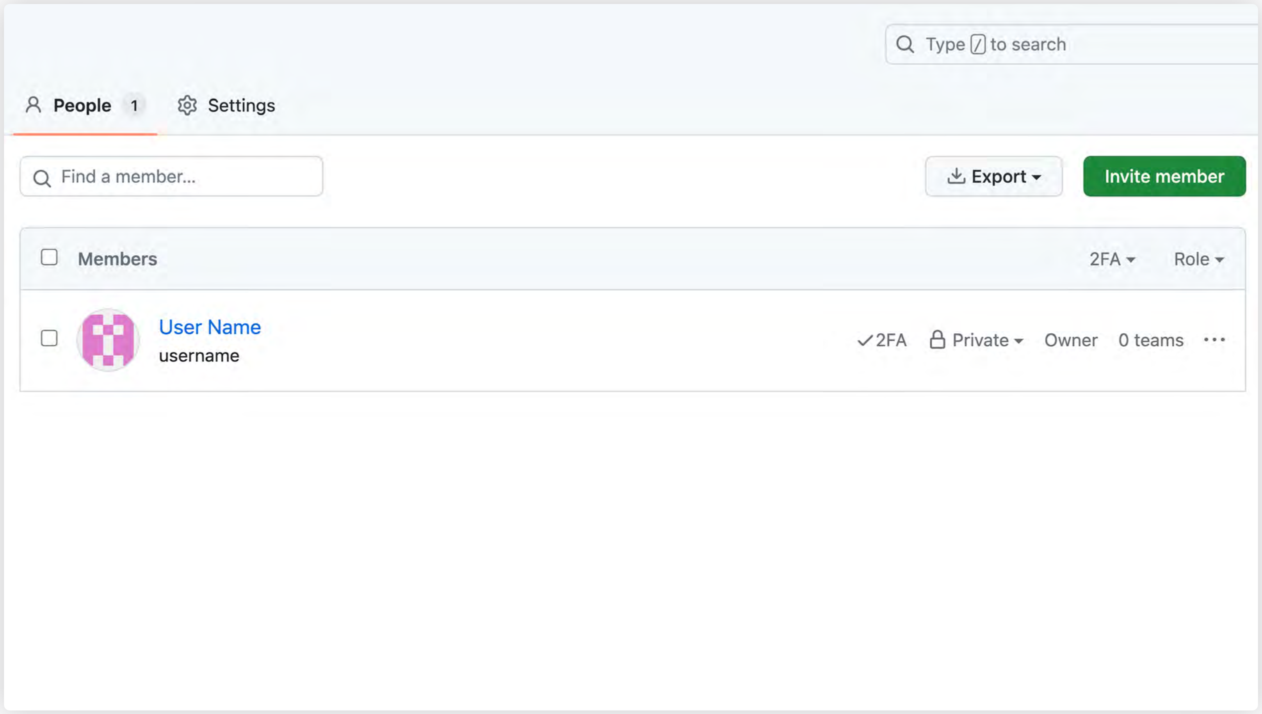
- **Read** provides read-only access to the repo's content and related resources (issues, pull requests, and so on).
- **Triage** has additional permissions to better proactively manage issues, pull requests, and more.
- **Write** has further write permissions to the repo's content to perform actions such as merging pull requests or creating releases.
- **Maintain** has broad access to the repository, with the exception of dangerous or destructive actions.
- **Admin** has full access to the repository, including permissions to change security settings or delete the repository.

Managing Authorization and Access Control in GitHub Organizations

If you have the necessary permissions, you can manage the roles of other users in the dedicated settings panels of the entire organization or individual repo. If you want to do this effectively, you'll need a strong understanding of GitHub's access control system and its UI dashboard.

General User and Team Management

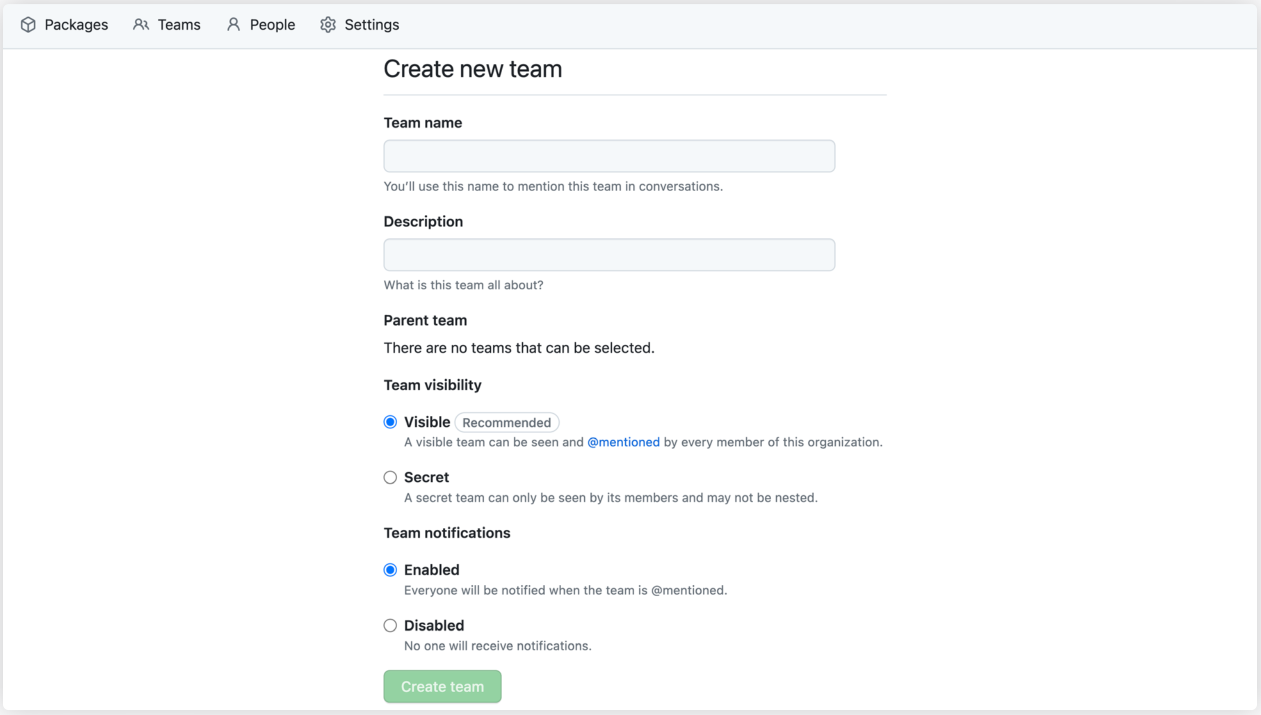
Managing your GitHub organization begins with inviting members. From the main organization dashboard, go to the **People** tab and click **Invite member**. Then, from the respective dropdown menu, you can convert any member to an outside collaborator, further manage their access to select repositories, or completely remove them from the organization:



Members		2FA		Role		
User Name username		✓2FA	Private	Owner	0 teams	···

In addition to managing members individually, you can also assign them to teams and manage their repository access and permissions collectively.

To create a new team, head to the **Teams** tab and click **New team**, then fill in the relevant details:



Create new team

Team name

You'll use this name to mention this team in conversations.

Description

What is this team all about?

Parent team

There are no teams that can be selected.

Team visibility

[✓ Visible] Recommended
A visible team can be seen and @mentioned by every member of this organization.

Secret
A secret team can only be seen by its members and may not be nested.

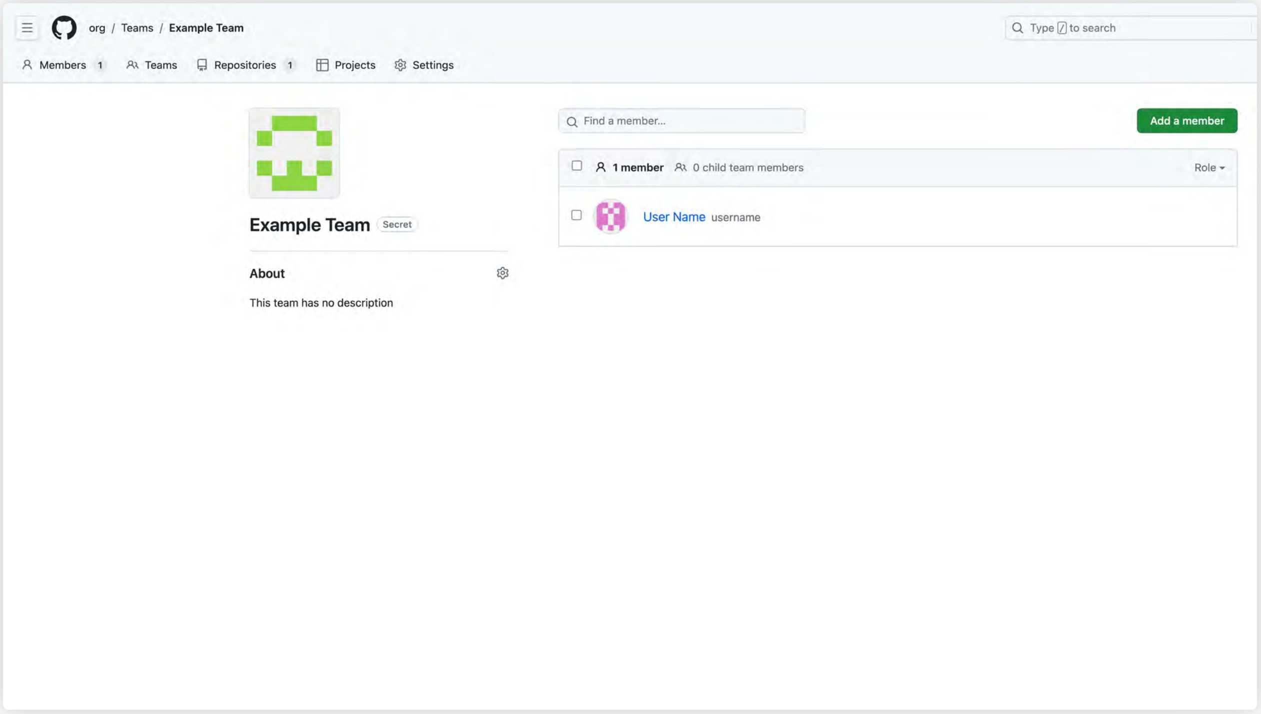
Team notifications

[✓ Enabled]
Everyone will be notified when the team is @mentioned.

Disabled
No one will receive notifications.

Create team

Once you create a team, you'll be redirected to its dedicated dashboard:



Example Team Secret

About

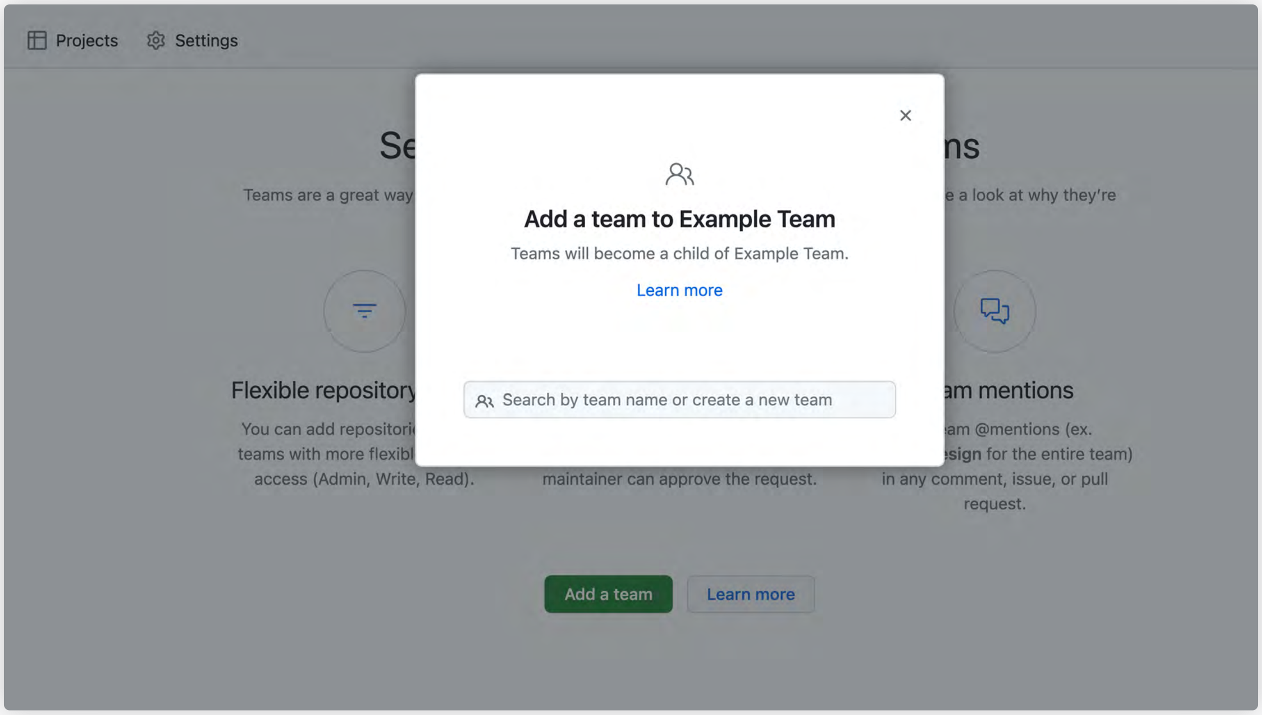
This team has no description

1 member 0 child team members

User Name username

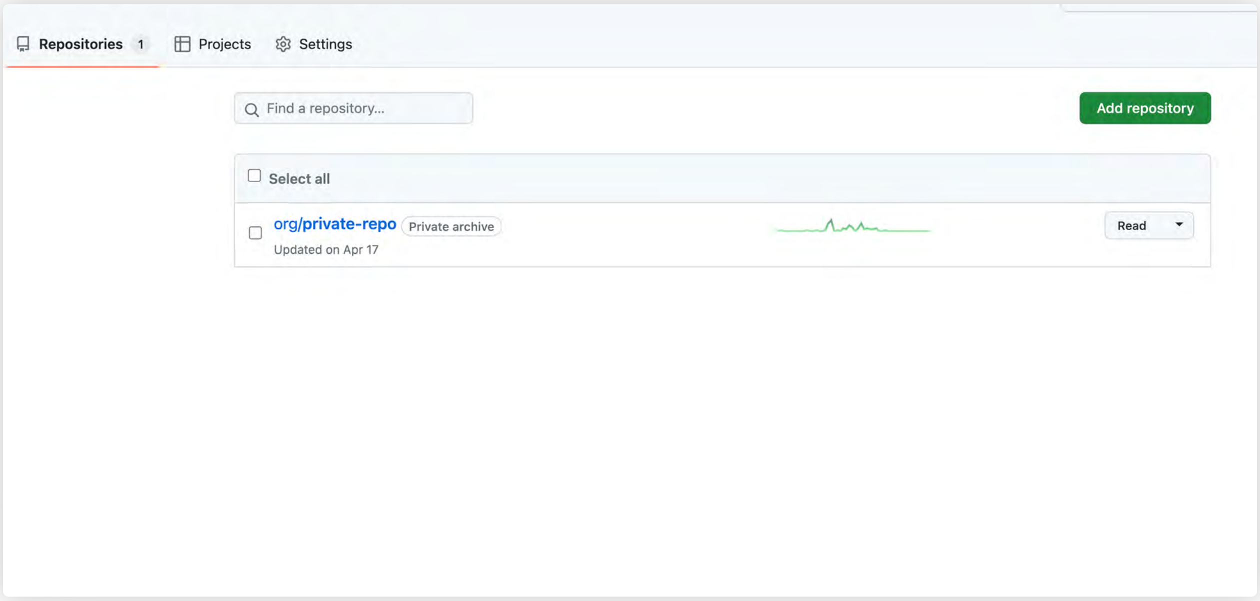
From here, you can manage the team, including its details, members, and repository-level roles. You can add a member to the team by clicking **Add a member**. This can be someone who is already an organization member or someone you are inviting to become a member of both the team and the entire organization.

In the **Teams** tab, you can also add an entire child team by clicking **Add a team**. This can be used to better organize and manage access control:



Projects
Settings
Add a team to Example Team
Teams will become a child of Example Team.
Learn more
Search by team name or create a new team
Flexible repository
You can add repositori teams with more flexibl access (Admin, Write, Read).
maintainer can approve the request.
am mentions
am @mentions (ex. sign for the entire team) in any comment, issue, or pull request.
Add a team
Learn more

You can assign a repository-level role to a team to provide repository access. To do this, go to the **Repositories** panel and click **Add repository**:

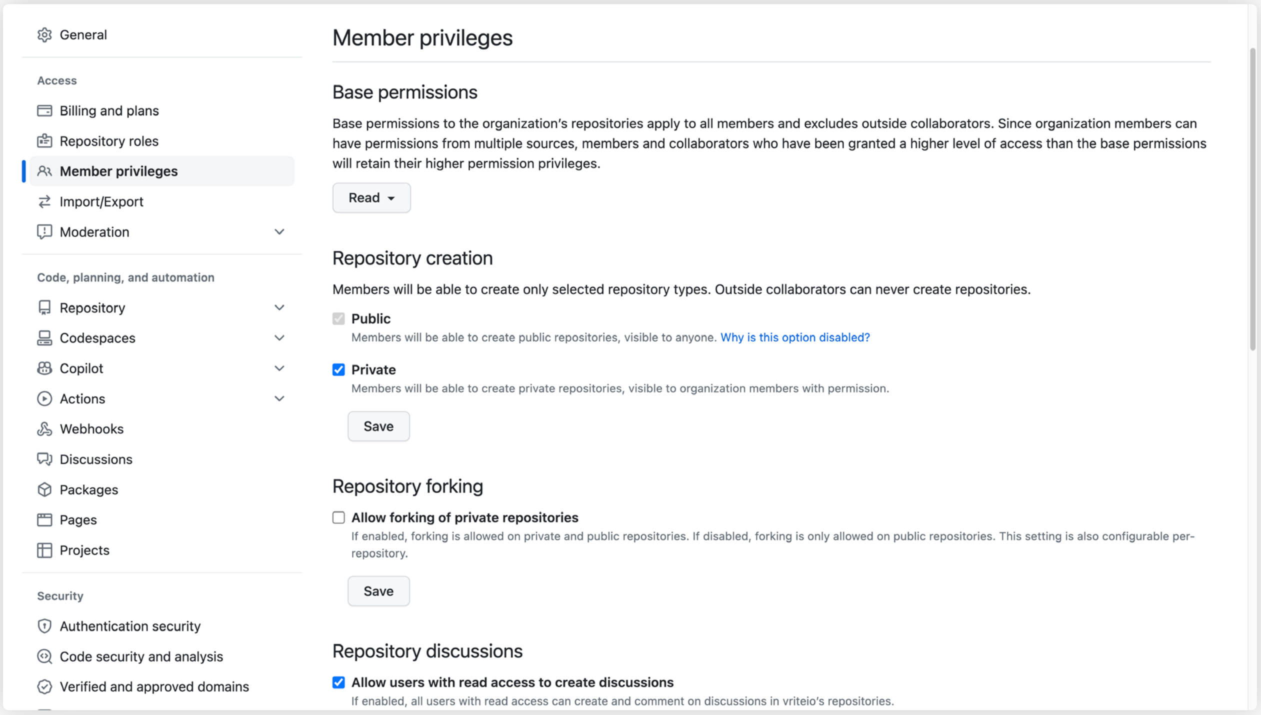


For each repository you add, you'll have to choose a role to assign to the team.

Organization-Level Access Control

The member role serves as the base role for the organization's users. GitHub provides extra organization-level tools to enhance control over the privileges associated with this role.

You can configure various options for the member role in the organization's **Settings** tab under the **Member privileges** section. Most importantly, this includes base permissions, meaning the level of access each member will possess across all repositories. Additionally, you can customize access to various actions within repositories, projects, issues, and more:



Member privileges

Base permissions

Base permissions to the organization's repositories apply to all members and excludes outside collaborators. Since organization members can have permissions from multiple sources, members and collaborators who have been granted a higher level of access than the base permissions will retain their higher permission privileges.

Read

Repository creation

Members will be able to create only selected repository types. Outside collaborators can never create repositories.

[✓] Public
Members will be able to create public repositories, visible to anyone. Why is this option disabled?

[✓] Private
Members will be able to create private repositories, visible to organization members with permission.

Repository forking

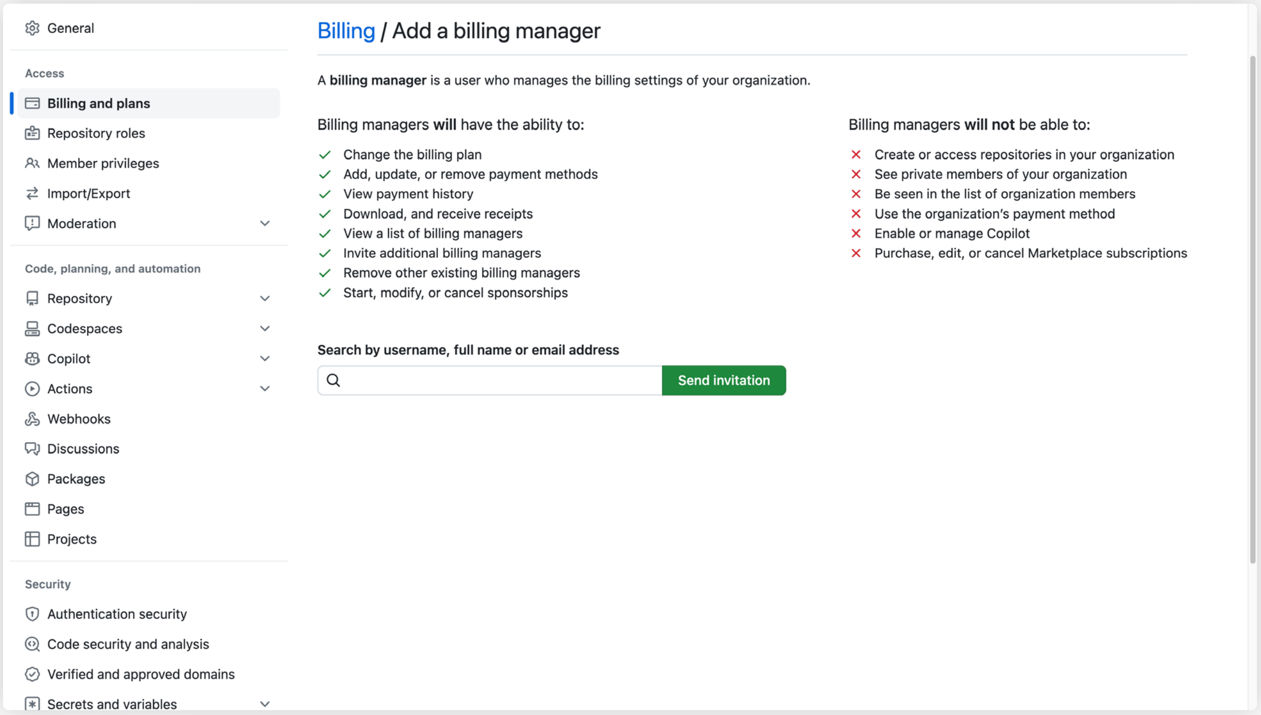
[ ] Allow forking of private repositories
If enabled, forking is allowed on private and public repositories. If disabled, forking is only allowed on public repositories. This setting is also configurable per-repository.

Repository discussions

[✓] Allow users with read access to create discussions
If enabled, all users with read access can create and comment on discussions in vriteio's repositories.

You can also assign other organization-level roles from the **Settings** tab.

To assign billing managers, go to the **Billing and plans > Billing managers** section and click **Invite**:



Billing / Add a billing manager

A **billing manager** is a user who manages the billing settings of your organization.

Billing managers **will** have the ability to:

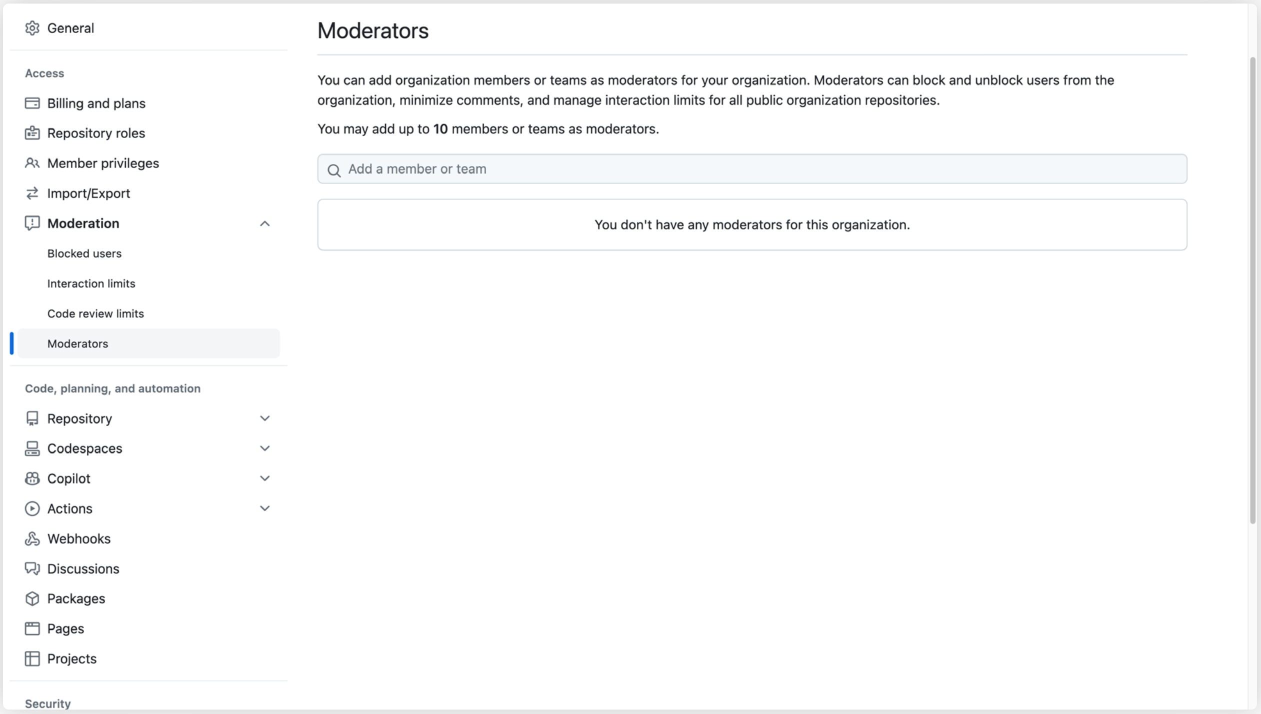
- ✓ Change the billing plan
- ✓ Add, update, or remove payment methods
- ✓ View payment history
- ✓ Download, and receive receipts
- ✓ View a list of billing managers
- ✓ Invite additional billing managers
- ✓ Remove other existing billing managers
- ✓ Start, modify, or cancel sponsorships

Billing managers **will not** be able to:

- ✗ Create or access repositories in your organization
- ✗ See private members of your organization
- ✗ Be seen in the list of organization members
- ✗ Use the organization's payment method
- ✗ Enable or manage Copilot
- ✗ Purchase, edit, or cancel Marketplace subscriptions

Search by username, full name or email address

You can assign existing users or entire teams as moderators in the **Moderation > Moderators** section:



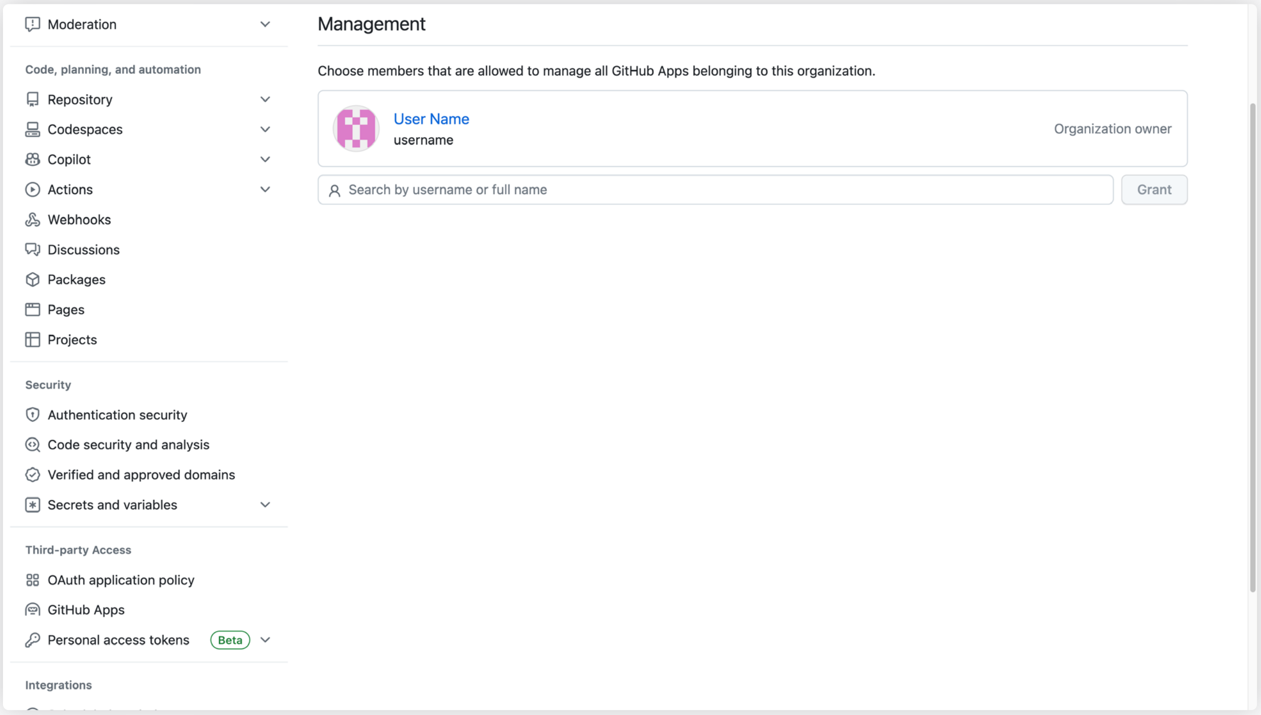
Moderators

You can add organization members or teams as moderators for your organization. Moderators can block and unblock users from the organization, minimize comments, and manage interaction limits for all public organization repositories.

You may add up to **10** members or teams as moderators.

You don't have any moderators for this organization.

Lastly, you can assign the GitHub App manager role to existing organization members from the **Developer Settings > GitHub Apps > Management** section:



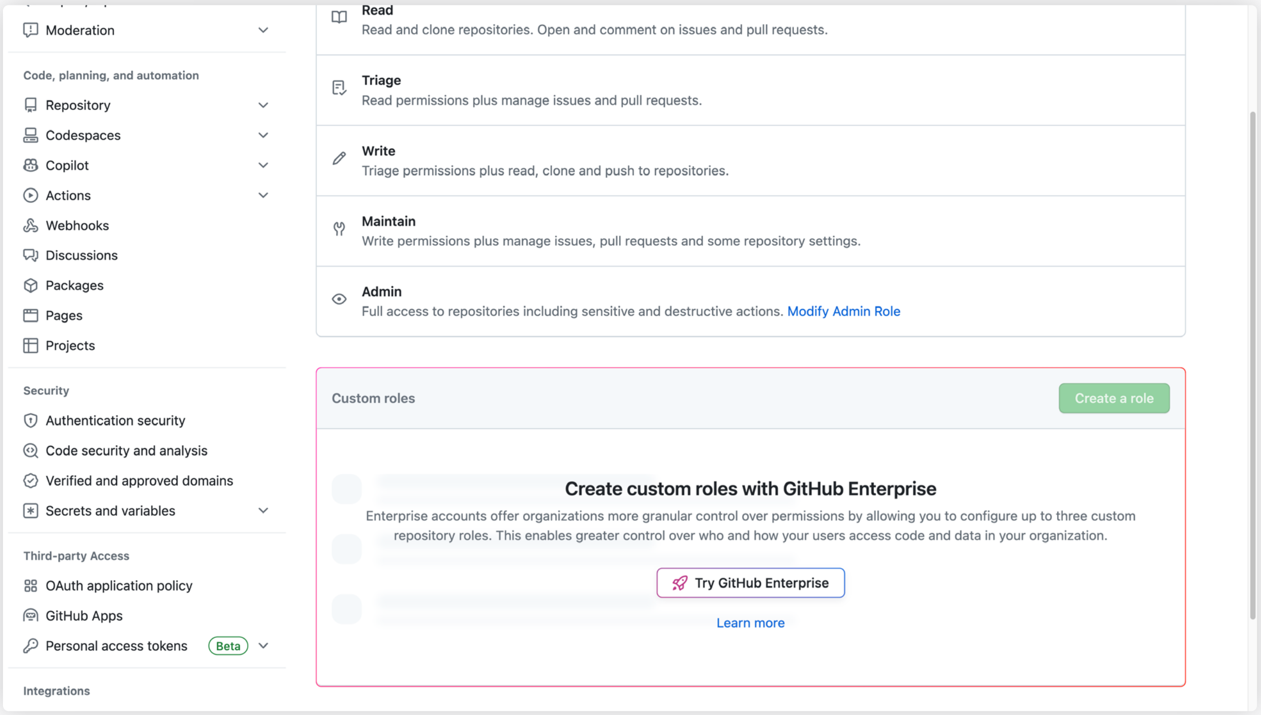
Management

Choose members that are allowed to manage all GitHub Apps belonging to this organization.

User Name
username
Organization owner

Search by username or full name
Grant

Among the different settings for managing your organization, the **Repository roles** section is particularly useful. It lets you see all the predefined repository-level roles. If you have a GitHub Enterprise account, it also lets you create custom repository roles for even more fine-grained control:



Read
Read and clone repositories. Open and comment on issues and pull requests.

Triage
Read permissions plus manage issues and pull requests.

Write
Triage permissions plus read, clone and push to repositories.

Maintain
Write permissions plus manage issues, pull requests and some repository settings.

Admin
Full access to repositories including sensitive and destructive actions. Modify Admin Role

Custom roles
Create a role

Create custom roles with GitHub Enterprise

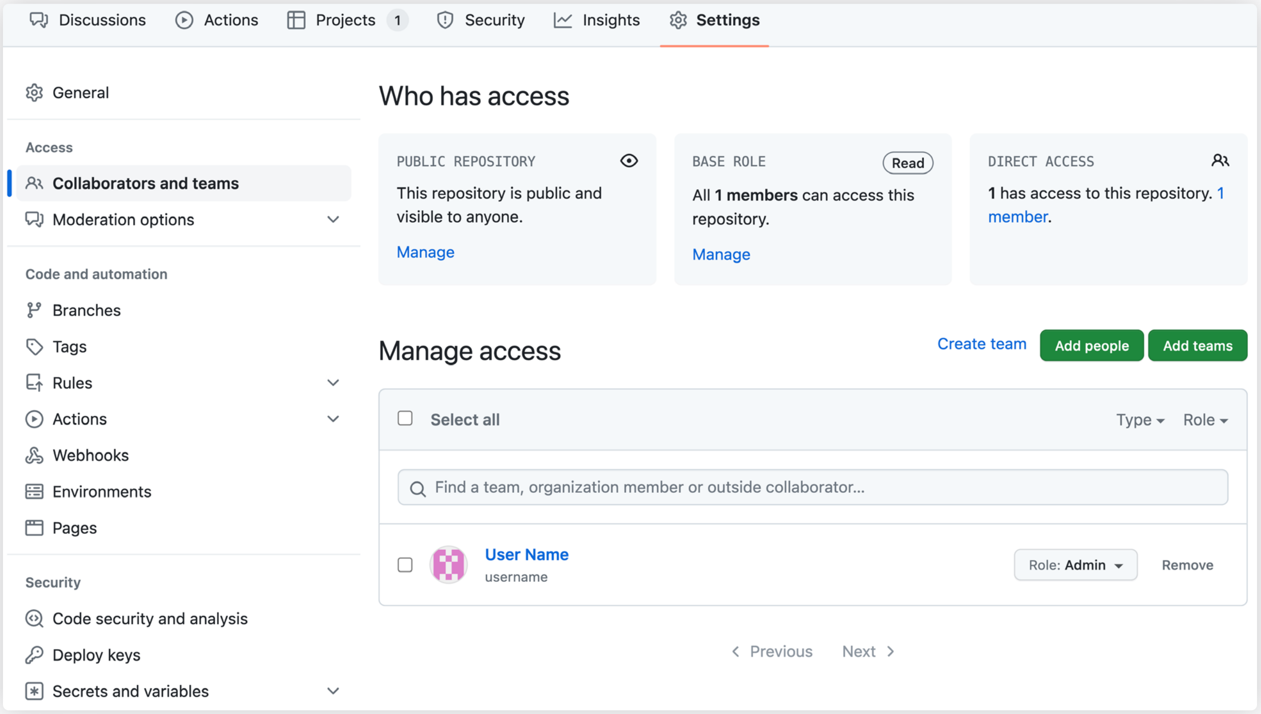
Enterprise accounts offer organizations more granular control over permissions by allowing you to configure up to three custom repository roles. This enables greater control over who and how your users access code and data in your organization.

Try GitHub Enterprise
Learn more

Custom repository roles aren't the only advantage of the GitHub Enterprise plan when it comes to access control. The plan also gives you access to more authentication security settings, including SAML Single Sign-On (SSO), custom enterprise policies, and an IP allow list—all certified for compliance.

Repository-Level Access Control

In addition to configuring repository-level roles through settings for members or teams, you can also do this at the level of a given repository:



Who has access

PUBLIC REPOSITORY
This repository is public and visible to anyone.
Manage

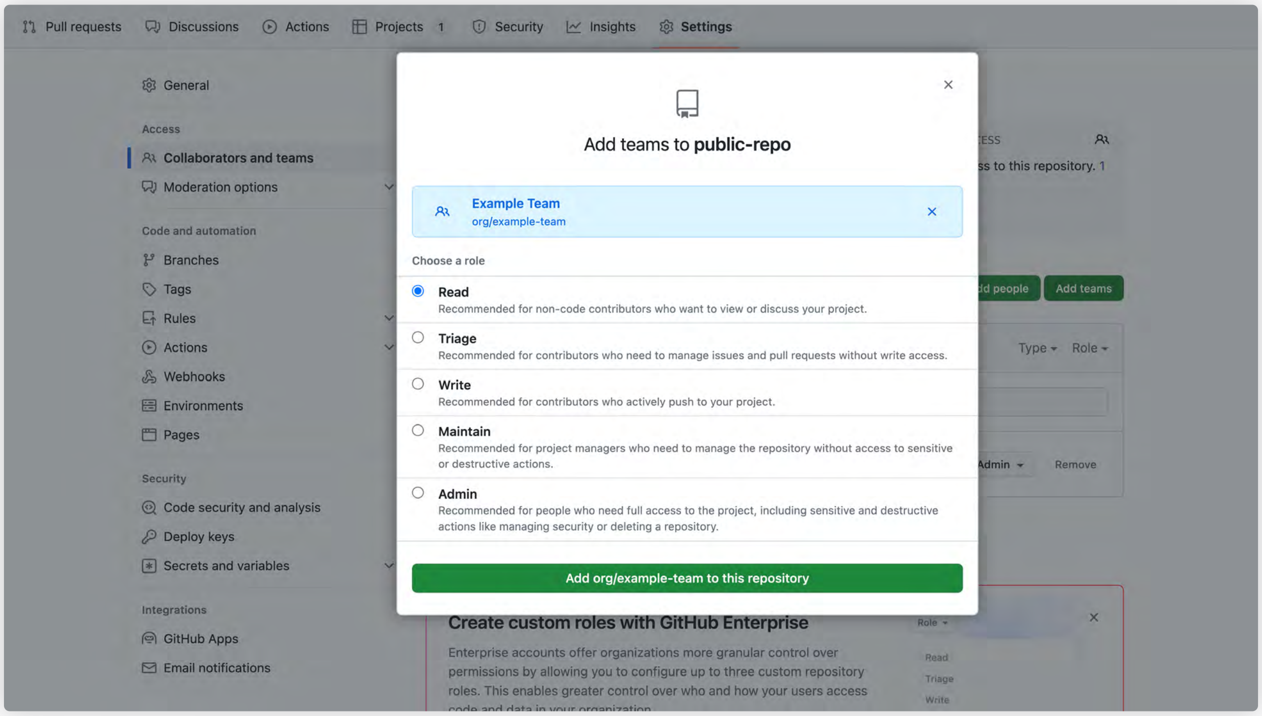
BASE ROLE Read
All 1 members can access this repository.
Manage

DIRECT ACCESS
1 has access to this repository. 1 member.

Manage access

User Name
username
Role: Admin
Remove

In the repository dashboard, under **Settings > Collaborators and teams**, you can use the **Add people** and **Add teams** buttons and manage repository-level roles accordingly for members and outside collaborators or teams:



Add teams to **public-repo**

Example Team
org/example-team

Choose a role

[✓ Read]
Recommended for non-code contributors who want to view or discuss your project.

Triage
Recommended for contributors who need to manage issues and pull requests without write access.

Write
Recommended for contributors who actively push to your project.

Maintain
Recommended for project managers who need to manage the repository without access to sensitive or destructive actions.

Admin
Recommended for people who need full access to the project, including sensitive and destructive actions like managing security or deleting a repository.

Add org/example-team to this repository

Create custom roles with GitHub Enterprise

Enterprise accounts offer organizations more granular control over permissions by allowing you to configure up to three custom repository roles. This enables greater control over who and how your users access

This provides you with an alternative way to manage access control, reassuring you that the changes you make affect only the selected repository.

Best Practices for Managing GitHub Access Control

When working with the access control features available on GitHub, you should consider the following best practices to ensure that your projects are secure and your organizational workflow is running efficiently.

Follow the Principle of Least Privilege

The principle of least privilege is a security best practice that recommends that users be granted the minimum level of access required to perform their tasks. This ensures that users do not have access to resources they don't need, reducing the risk of accidental or malicious misuse of resources.

You should consider administering roles and permissions at the team level rather than at the individual level, which makes implementing the principle of least privilege easier. For example, you can create a team for developers and assign them the "write" access level for the repository. Similarly, you can create a team for testers and assign them the "read" access level for the repository. This ensures that developers have the ability to push changes to the repository, while testers can only view the repository and clone it to their local machine.

Use Proper Roles for Users

You should assign the appropriate roles to users based on their responsibilities. Organizations generally have well-defined teams with specific responsibilities. You can reflect this structure by creating teams on GitHub and assigning the same roles and permissions to them as they have in the organization. For example, architects can be assigned the “maintain” access level for the repository and the “owner” role for the organization, while project leads can be assigned “triage” access to their respective repositories and the “member” role for the organization.

Set Repository Visibility and Access Control

You need to choose the right visibility settings when creating your repository, as this determines who can access it. GitHub provides three different visibility settings for repositories: public, private, and internal. The internal visibility setting is only available for organization accounts.

→ **Public**: Public repositories are visible to everyone on the internet. Anyone can view the repository and clone it to their local machine. However, only users with the appropriate roles and permissions can push changes to the repository.

→ **Private**: Private repositories are only visible to the account owner, users the owner shares access with, and users with the appropriate roles and permissions in the organization.

→ **Internal**: Internal repositories are visible to all members of the organization. An internal repo is not visible to users outside the organization. However, users outside the organization can be granted access to the repository by assigning them the outside collaborator role.

You should consider the sensitivity of your project when choosing the visibility setting for your repository. For example, if you are working on an open source project, you should choose the public visibility setting. On the other hand, if you are working on a proprietary project, you should choose the private or internal visibility setting.

Use Branch Protection Rules

Branch protection rules allow you to control the actions that can be performed on a branch. You can configure branch protection rules to prevent users from pushing changes directly to the “main” branch. You can also configure policies to ensure that any merge to the “main” branch is done via a pull request and that the pull request is approved by a reviewer before it can be merged.

Handle Credentials with Secrets

You should avoid hard-coding credentials in your codebase, as it can lead to the accidental exposure of sensitive information. Instead, you can use secrets, which are encrypted environment variables specifically designed to securely store and manage credentials within your GitHub workflows. Using secrets ensures that sensitive credentials remain protected and won’t be exposed to unauthorized users.

It’s good practice to store secrets at the organization level and use access policies to control who can access them. This ensures that secrets are not duplicated between repositories and are centrally managed.

Remove Inactive Users

You should regularly audit your team list and remove users who are no longer active or part of the project to maintain security and clarity in the team structure. You can use the audit log to identify inactive users and remove them from your team.

Strengths of GitHub’s Authorization Model

Having a good understanding of GitHub’s access control model, its implementation, and its usage allows you to better understand its strengths and limitations.

Having granular controls and the ability to configure individual user access to selected repositories allows you to manage an organization at any scale. GitHub’s RBAC model, its built-in roles, and its implementation of teams and sub-teams provide enough control and flexibility so that you can model the access control settings after the way your organization is structured.

Large organizations with hundreds of members can use teams and sub-teams to manage general access at the team level. Individual teams can then manage their own members, allowing for easier control within a focused scope. On the other hand, smaller organizations can manage individual members’ roles directly and take advantage of the “outside collaborator” role for onboarding new or temporary members.

Further access control configuration at the repository level facilitates collaboration and various repository-bound workflows that are available on GitHub. From managing pull requests and issues to using GitHub projects or CI/CD, proper access control makes sure only trusted members have access to dangerous or destructive actions.

Limitations and Challenges of GitHub’s Authorization Permission Model

That said, GitHub’s model does have its limitations.

GitHub’s RBAC implementation provides many tools that allow you to have granular control over the roles of every member of the organization. However, with such flexibility comes increased complexity, which can quickly become a challenge, especially when managing a large organization. From the organization-level roles, through teams and sub-teams, down to individual members’ permissions to select repositories, having to manage and delegate tasks through all the levels can quickly get out of hand.

While complexity may only affect large organizations, the limitations of GitHub’s RBAC implementation can be experienced at any scale. Even though GitHub has a concept of permissions, you’re limited to assigning predefined roles for the most part. The few configuration options GitHub provides may not be enough in case you need to create custom roles (for example, for contractors or community managers) to accurately model access control in your organization. Such a level of control is reserved only for GitHub Enterprise accounts.

However, custom roles aren't the only advantage of GitHub Enterprise Cloud. If your organization has to adhere to special compliance and security standards, no account type other than Enterprise can fit your needs. This vastly limits your options and can be especially problematic for smaller, specialized organizations, which must adhere to various standards but have little need for other features of GitHub Enterprise. Still, in such cases, Enterprise Cloud, being certified for compliance (with annual SOC 1 Type 2 and SOC 2 Type 2 reports) and providing extended authentication security features (like SAML SSO), is your only option.

Conclusion

You should now know how GitHub's RBAC model works and how you can make the best use of it in your GitHub organization.

Implementing role-based access control as advanced as GitHub's might seem challenging, but it doesn't have to be. With ConductorOne, an identity security automation platform, you can easily add modern, user-focused access controls with streamlined self-service requests and just-in-time provisioning to any app.

ConductorOne has a feature-rich GitHub integration that enables you to provision just-in-time access to your organization's repositories. Using the integration, you can easily manage user access to your repositories using the same intuitive UI that you use for managing access to other resources within your organization. Take a tour or book a demo to learn more.