About protected branches

In this article

You can protect important branches by setting branch protection rules, which define whether collaborators can delete or force push to the branch and set requirements for any pushes to the branch, such as passing status checks or a linear commit history.

Protected branches are available in public repositories with GitHub Free and GitHub Free for organizations, and in public and private repositories with GitHub Pro, GitHub Team, GitHub Enterprise Cloud, and GitHub Enterprise Server.

About branch protection rules

You can enforce certain workflows or requirements before a collaborator can push changes to a branch in your repository, including merging a pull request into the branch, by creating a branch protection rule. Actors may only be added to bypass lists when the repository belongs to an organization.

By default, each branch protection rule disables force pushes to the matching branches and prevents the matching branches from being deleted. You can optionally disable these restrictions and enable additional branch protection settings.

By default, the restrictions of a branch protection rule don't apply to people with admin permissions to the repository or custom roles with the "bypass branch protections" permission. You can optionally apply the restrictions to administrators and roles with the "bypass branch protections" permission, too. For more information, see "Managing custom repository roles for an organization".

You can create a branch protection rule in a repository for a specific branch, all branches, or any branch that matches a name pattern you specify with `fnmatch` syntax. For example, to protect any branches containing the word `release`, you can create a branch rule for `*release*`. For more information about branch name patterns, see "Managing a branch protection rule."

You can configure a pull request to merge automatically when all merge requirements are met. For more information, see "Automatically merging a pull request."

About branch protection settings

For each branch protection rule, you can choose to enable or disable the following settings.

- Require pull request reviews before merging

- Require status checks 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
- Restrict who can push to matching branches
- Allow force pushes
- Allow deletions

For more information on how to set up branch protection, see "Managing a branch protection rule."

Require pull request reviews before merging

Repository administrators or custom roles with the "edit repository rules" permission can require that all pull requests receive a specific number of approving reviews before someone merges the pull request into a protected branch. You can require approving reviews from people with write permissions in the repository or from a designated code owner.

If you enable required reviews, collaborators can only push changes to a protected branch via a pull request that is approved by the required number of reviewers with write permissions.

If a person with admin permissions chooses the **Request changes** option in a review, then that person must approve the pull request before the pull request can be merged. If a reviewer who requests changes on a pull request isn't available, anyone with write permissions for the repository can dismiss the blocking review.

Even after all required reviewers have approved a pull request, collaborators cannot merge the pull request if there are other open pull requests that have a head branch pointing to the same commit with pending or rejected reviews. Someone with write permissions must approve or dismiss the blocking review on the other pull requests first.

If a collaborator attempts to merge a pull request with pending or rejected reviews into the protected branch, the collaborator will receive an error message.

```
remote: error: GH006: Protected branch update failed for refs/heads/main.
remote: error: Changes have been requested.
```

Optionally, you can choose to dismiss stale pull request approvals when commits are pushed that affect the diff in the pull request. GitHub records the state of the diff at the point when a pull request is approved. This state represents the set of changes that the reviewer approved. If the diff changes from this state (for example, because a contributor pushes new changes to the pull request branch or clicks **Update branch**, or because a related pull request is merged into the target branch), the approving review is dismissed as stale, and the pull request cannot be merged until someone approves the work again. For information about the base branch, see "About pull requests."

Optionally, you can restrict the ability to dismiss pull request reviews to specific people or teams. For more information, see "Dismissing a pull request review."

Optionally, you can choose to require reviews from code owners. If you do, any pull

request that affects code with a code owner must be approved by that code owner before the pull request can be merged into the protected branch.

Optionally, you can require that the most recent reviewable push must be approved by someone other than the person who pushed it. This means at least one other authorized reviewer has approved any changes. For example, the "last reviewer" can check that the latest set of changes incorporates feedback from other reviews, and does not add new, unreviewed content.

For complex pull requests that require many reviews, requiring an approval from someone other than the last person to push can be a compromise that avoids the need to dismiss all stale reviews: with this option, "stale" reviews are not dismissed, and the pull request remains approved as long as someone other than the person who made the most recent changes approves it. Users who have already reviewed a pull request can reapprove after the most recent push to meet this requirement. If you are concerned about pull requests being "hijacked" (where unapproved content is added to approved pull requests), it is safer to dismiss stale reviews.

> **Note:** If you select **Dismiss stale pull request approvals when new commits are pushed** and/or **Require approval of the most recent reviewable push**, manually creating the merge commit for a pull request and pushing it directly to a protected branch will fail, unless the contents of the merge exactly match the merge generated by GitHub for the pull request.
>
> In addition, with these settings, approving reviews will be dismissed as stale if the merge base introduces new changes after the review was submitted. The merge base is the commit that is the last common ancestor between the topic branch and the base branch. If the merge base changes, the pull request cannot be merged until someone approves the work again.

Require status checks before merging

Required status checks ensure that all required CI tests are passing before collaborators can make changes to a protected branch. Required status checks can be checks or statuses. For more information, see "About status checks."

You can use the commit status API to allow external services to mark commits with an appropriate status. For more information, see "Commit statuses" in the REST API documentation.

After enabling required status checks, all required status checks must pass before collaborators can merge changes into the protected branch. After all required status checks pass, any commits must either be pushed to another branch and then merged or pushed directly to the protected branch.

Any person or integration with write permissions to a repository can set the state of any status check in the repository, but in some cases you may only want to accept a status check from a specific GitHub App. When you add a required status check, you can select an app that has recently set this check as the expected source of status updates. If the status is set by any other person or integration, merging won't be allowed. If you select "any source", you can still manually verify the author of each status, listed in the merge box.

You can set up required status checks to either be "loose" or "strict." The type of required status check you choose determines whether your branch is required to be up to date with the base branch before merging.

Type of required status check	Setting	Merge requirements	Considerations
Strict	The **Require branches to be up to date before merging** checkbox is checked.	The branch **must** be up to date with the base branch before merging.	This is the default behavior for required status checks. More builds may be

			required, as you'll need to bring the head branch up to date after other collaborators update the target branch.
Loose	The **Require branches to be up to date before merging** checkbox is **not** checked.	The branch **does not** have to be up to date with the base branch before merging.	You'll have fewer required builds, as you won't need to bring the head branch up to date after other collaborators merge pull requests. Status checks may fail after you merge your branch if there are incompatible changes with the base branch.
Disabled	The **Require status checks to pass before merging** checkbox is **not** checked.	The branch has no merge restrictions.	If required status checks aren't enabled, collaborators can merge the branch at any time, regardless of whether it is up to date with the base branch. This increases the possibility of incompatible changes.

For troubleshooting information, see "Troubleshooting required status checks."

Require conversation resolution before merging

Requires all comments on the pull request to be resolved before it can be merged to a protected branch. This ensures that all comments are addressed or acknowledged before merge.

Require signed commits

When you enable required commit signing on a branch, contributors can only push commits that have been signed and verified to the branch. For more information, see "About commit signature verification."

> **Note:** If a collaborator pushes an unsigned commit to a branch that requires commit signatures, the collaborator will need to rebase the commit to include a verified signature, then force push the rewritten commit to the branch.

You can always push local commits to the branch if the commits are signed and verified. However, you cannot merge pull requests into the branch on GitHub Enterprise Server. You can merge pull requests locally. For more information, see "Checking out pull requests locally."

Require linear history

Enforcing a linear commit history prevents collaborators from pushing merge commits to the branch. This means that any pull requests merged into the protected branch must use a squash merge or a rebase merge. A strictly linear commit history can help teams reverse changes more easily. For more information about merge methods, see "About pull request merges."

Before you can require a linear commit history, your repository must allow squash merging or rebase merging. For more information, see "Configuring pull request merges."

Require deployments to succeed before merging

You can require that changes are successfully deployed to specific environments before a branch can be merged. For example, you can use this rule to ensure that changes are successfully deployed to a staging environment before the changes merge to your default branch.

Lock branch

Locking a branch ensures that no commits can be made to the branch. By default, a forked repository does not support syncing from its upstream repository. You can enable **Allow fork syncing** to pull changes from the upstream repository while preventing other contributions to the fork's branch.

Do not allow bypassing the above settings

By default, the restrictions of a branch protection rule do not apply to people with admin permissions to the repository or custom roles with the "bypass branch protections" permission in a repository.

You can enable this setting to apply the restrictions to admins and roles with the "bypass branch protections" permission, too. For more information, see "Managing custom repository roles for an organization".

Restrict who can push to matching branches

When you enable branch restrictions, only users, teams, or apps that have been given permission can push to the protected branch. You can view and edit the users, teams, or apps with push access to a protected branch in the protected branch's settings. When status checks are required, the people, teams, and apps that have permission to push to a protected branch will still be prevented from merging into the branch when the required checks fail. People, teams, and apps that have permission to push to a protected branch will still need to create a pull request when pull requests are required.

Optionally, you can apply the same restrictions to the creation of branches that match the rule. For example, if you create a rule that only allows a certain team to push to any branches that contain the word `release`, only members of that team would be able to create a new branch that contains the word `release`.

You can only give push access to a protected branch, or give permission to create a matching branch, to users, teams, or installed GitHub Apps with write access to a repository. People and apps with admin permissions to a repository are always able to push to a protected branch or create a matching branch.

Allow force pushes

By default, GitHub Enterprise Server blocks force pushes on all protected branches. When you enable force pushes to a protected branch, you can choose one of two groups who can force push:

1. Allow everyone with at least write permissions to the repository to force push to the branch, including those with admin permissions.
2. Allow only specific people or teams to force push to the branch.

If someone force pushes to a branch, the force push may mean commits that other collaborators based their work on are removed from the history of the branch. People may have merge conflicts or corrupted pull requests. Force pushing can also be used to delete branches or point a branch to commits that were not approved in a pull request.

Enabling force pushes will not override any other branch protection rules. For example, if a branch requires a linear commit history, you cannot force push merge commits to that branch.

You cannot enable force pushes for a protected branch if a site administrator has blocked force pushes to all branches in your repository. For more information, see "Enforcing repository management policies in your enterprise."

If a site administrator has blocked force pushes to the default branch only, you can still enable force pushes for any other protected branch.

Allow deletions

By default, you cannot delete a protected branch. When you enable deletion of a protected branch, anyone with at least write permissions to the repository can delete the branch.