

# Creating rulesets for a repository

## In this article

Introduction

Creating a branch or tag ruleset

Granting bypass permissions for your ruleset

Choosing which branches or tags to target

Selecting branch or tag protections

Adding metadata restrictions

Finalizing your ruleset and next steps

Using fnmatch syntax

Using regular expressions for commit metadata

You can add rulesets to a repository to control how people can interact with specific branches and tags.

### Who can use this feature

Anyone with read access to a repository can view the repository's rulesets. People with admin access to a repository, or a custom role with the "edit repository rules" permission, can create, edit, and delete rulesets for a repository and view ruleset insights. For more information, see "[About custom repository roles](#)."

Rulesets are available in public repositories with GitHub Free and GitHub Free for organizations, and in public and private repositories with GitHub Pro, GitHub Team, and GitHub Enterprise Cloud. For more information, see "[GitHub's plans](#)."

## Introduction

You can create rulesets to control how users can interact with selected branches and tags in a repository. When you create a ruleset, you can allow certain users to bypass the rules in the ruleset. This can be users with certain permissions, specific teams, or GitHub Apps. For more information on rulesets, see "[About rulesets](#)."

You can import a ruleset from another repository or organization using a JSON file. This can be useful if you want to apply the same ruleset to multiple repositories or organizations. For more information, see "[Managing rulesets for repositories in your organization](#)."


You can also create rulesets for all repositories in an organization. For more information, see "[Creating rulesets for repositories in your organization](#)."

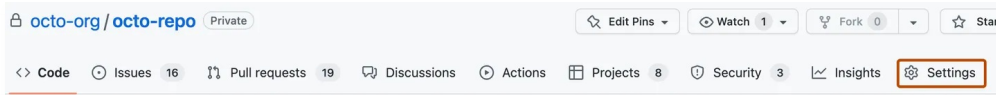
To create a ruleset, complete the following procedures:

- [Creating a branch or tag ruleset](#)
- [Granting bypass permissions for your ruleset](#)
- [Choosing which branches or tags to target](#)
- [Selecting branch or tag protections](#)
- [Adding metadata restrictions](#)

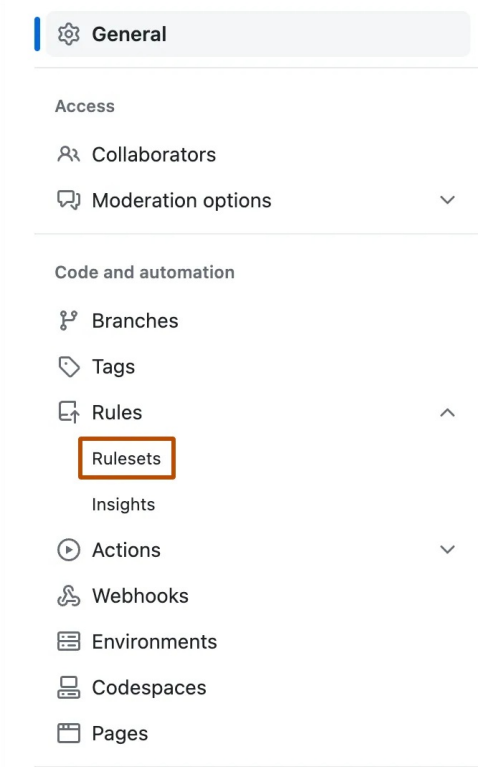
- [Finalizing your ruleset and next steps](#)


## Creating a branch or tag ruleset [🔗](#)

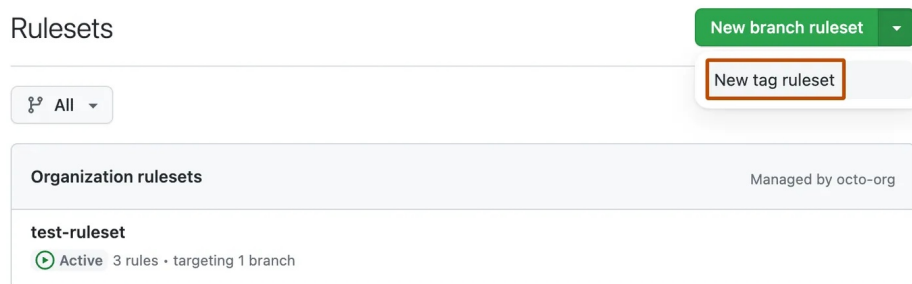
- 1 On GitHub.com, navigate to the main page of the repository.
- 2 Under your repository name, click  **Settings**. If you cannot see the "Settings" tab, select the ... dropdown menu, then click **Settings**.







- 3 In the left sidebar, under "Code and automation," click **Rules**, then click **Rulesets**.



- 4 You can create a ruleset targeting branches, or a ruleset targeting tags.
  - To create a ruleset targeting branches, click **New branch ruleset**.
  - To create a ruleset targeting tags, select , then click **New tag ruleset**.



- 5 In the "General" section, type a name for the ruleset, then select  **Disabled** and click one of the following enforcement statuses:

-  **Active:** your ruleset will be enforced upon creation.
-  **Evaluate:** your ruleset will not be enforced, but you will be able to monitor which actions would or would not violate rules on the "Rule Insights" page. For more information, see "[Viewing insights for rulesets](#)."
-  **Disabled:** your ruleset will not be enforced or evaluated.

## Granting bypass permissions for your ruleset

You can grant certain roles, teams, or apps bypass permissions for your ruleset. The following are eligible for bypass access:

- Repository admins or organization owners
- The maintain or write role, or custom repository roles based on the write role
- Teams
- GitHub Apps

- 1 To grant bypass permissions for the ruleset, in the "Bypass list" section, click + **Add bypass**.
- 2 In the "Add bypass" modal dialog that appears, search for the role, team, or app you would like to grant bypass permissions, then select the role, team, or app from the "Suggestions" section and click **Add Selected**.
- 3 Optionally, to grant bypass to an actor without allowing them to push directly to a repository, select **Always** ▾, then click **For pull requests only**.

The selected actor is now required to open a pull request to make changes to a repository, creating a clear digital trail with their changes. The actor can then choose to bypass any branch protections and merge that pull request.

## Choosing which branches or tags to target

To target branches or tags, in the "Target branches" or "Target tags" section, select **Add a target**, then select how you want to include or exclude branches or tags. You can use `fnmatch` syntax to include or exclude branches or tags based on a pattern. For more information, see "[Using fnmatch syntax](#)."

You can add multiple targeting criteria to the same ruleset. For example, you could include the default branch, include any branches matching the pattern `*feature*`, and then specifically exclude a branch matching the pattern `not-a-feature`.

## Selecting branch or tag protections

In the "Branch protections" or "Tag protections" section, select the rules you want to include in the ruleset. When you select a rule, you may be able to enter additional settings for the rule. For more information on the rules, see "[Available rules for rulesets](#)."

**Notes:** If you select **Require status checks before merging**, in the "Additional settings" section:

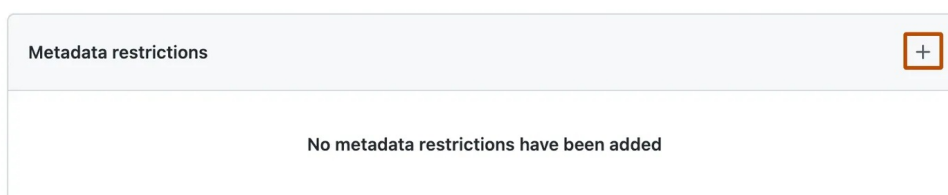
- You can enter the name of each status check you would like to require. To finish adding the status check as a requirement, you must click +.
- If you select **Require branches to be up to date before merging**, you must define a check for the protection to take effect.


## Adding metadata restrictions

### Notes:

- Adding metadata restrictions can impact the experience of people contributing to your repository. Before you enact a ruleset with metadata restrictions, you can select the "Evaluate" enforcement status for your ruleset to test the effects of any metadata restrictions without impacting contributors. For more information on metadata restrictions, see ["Available rules for rulesets."](#)
- Metadata restrictions are intended to increase consistency between commits in your repository. They are not intended to replace security measures such as requiring code review via pull requests.
- If you squash merge a branch, all commits on that branch must meet any metadata requirements for the base branch.

- 1 Optionally, in the "Metadata restrictions" section, to add a rule to control the metadata of commits being pushed to the branch or tag, click +.



Metadata restrictions 

No metadata restrictions have been added

- 2 Configure the settings for the metadata restriction rule, then click **Add**. You can add multiple restrictions to the same ruleset.

For most requirements, such as "Must start with a matching pattern," the pattern you enter is interpreted literally, and wildcards are not supported. For example, the `*` character only represents the literal `*` character.

For more complex patterns, you can select "Must match a given regex pattern" or "Must not match a given regex pattern," then use regular expression syntax to define the matching pattern. For more information, see ["Using regular expressions for commit metadata."](#)

Anyone who views the rulesets for a repository will be able to see the description you provide.

## Finalizing your ruleset and next steps

To finish creating your ruleset, click **Create**. If the enforcement status of the ruleset is set to "Active", the ruleset takes effect immediately.

You can view insights for the ruleset to see how the rules are affecting your contributors. If the enforcement status is set to "Evaluate", you can see which actions would have passed or failed if the ruleset was active. For more information on insights for rulesets, see ["Managing rulesets for a repository."](#)

## Using `fnmatch` syntax

You can use `fnmatch` syntax to define patterns to target the names of branches, tags, and repositories when you create a ruleset.

You can use the `*` wildcard to match any string of characters. Because GitHub uses the

`File::FNM_PATHNAME` flag for the `File.fnmatch` syntax, the `*` wildcard does not match directory separators ( `/` ). For example, `qa/*` will match all branches beginning with `qa/` and containing a single slash, but will not match `qa/foo/bar` . You can include any number of slashes after `qa` with `qa/**/*` , which would match, for example, `qa/foo/bar/foobar/hello-world` . You can also extend the `qa` string with `qa**/**/*` to make the rule more inclusive.

For more information about syntax options, see the [fnmatch documentation](#).

**Note:** Although GitHub supports `File::FNM_PATHNAME` in `fnmatch` syntax, `File::FNM_EXTGLOB` is not supported.

## Using regular expressions for commit metadata [↗](#)

When you add metadata restrictions, you can use regular expression syntax to define patterns that the relevant metadata, such as the commit message or the branch or tag name, must or must not match.

Rulesets support RE2 syntax. For more information, see Google's [syntax guide](#). To validate your expressions, you can use the validator on [regex101.com](#), selecting the "Golang" flavor in the left sidebar.

By default, regular expressions in metadata restrictions do not consider multiple lines of text. For example, if you have a multiline commit message, the pattern `^ABC` will be a match if the first line of the message starts with `ABC` . To match multiple lines of the message, start your expression with `(?m)` .

The negative lookahead assertion, denoted `?!` , is not supported. However, for cases where you need to look for a given string that is not followed by another given string, you can use the positive lookahead assertion, denoted `?` , combined with the "Must not match a given regex pattern" requirement.

**Note:** If you require contributors to sign off on commits, this may interfere with your regular expression patterns. When someone signs off, GitHub adds a string like `Signed-off-by: #AUTHOR-NAME <#AUTHOR-EMAIL>` to the commit message. For more information, see "[Managing the commit signoff policy for your organization](#)."

### Useful regular expression patterns [↗](#)

The following examples provide useful patterns for commit metadata. To use these patterns, set **Requirement** to "Must match a given regex pattern".

#### Ensure branch names are compatible with Windows [↗](#)

You can use the following pattern to ensure that branch names only include numbers, lowercase letters, and the characters `-` and `_` . This ensures branch names are compatible with operating systems that do not use case-sensitive file systems by default.

Text



```
\A[0-9a-z-_$]
```

Matches: `my-branch`

Does not match: `myBranch`

#### Ensure tag names use semantic versioning [↗](#)

You can use the following pattern to ensure tag names conform to semantic versioning. For more information, see the documentation on [semver.org](https://semver.org).

Text



```
^(0|[1-9]\d*)\.(0|[1-9]\d*)\.(0|[1-9]\d*)(?:-((?:0|[1-9]\d*|\d*[a-zA-Z-][0-9a-zA-Z-]*)?(?:\.(?:0|[1-9]\d*|\d*[a-zA-Z-][0-9a-zA-Z-]*)*)?)?(?:\+[0-9a-zA-Z-](?:\.[0-9a-zA-Z-](?:0|[1-9]\d*|\d*[a-zA-Z-][0-9a-zA-Z-]*)*)?)?)?$
```

Matches: 1.2.3 , 10.20.30 , 1.1.2-prerelease+meta

Does not match: 1.2 , 1.2-SNAPSHOT

### Limit length of lines in commit messages [🔗](#)

The [Pro Git book](#) recommends limiting the first line of a commit message to around 50 characters.

You can use the following pattern to ensure the first line in a commit message contains 50 characters or fewer.

Text



```
\A.{1,50}$
```

### Ensure commit messages start with a resolution and issue number [🔗](#)

You can use the following pattern to ensure that commit messages contain the word `Resolves:` or `Fixes:` , followed by a string like `#1234` .

Text



```
^(Resolves|Fixes): \#[0-9]+$
```

Matches: Fixes: #1234

Does not match: Add conditional logic to foo.bar

### Enforce conventional commits [🔗](#)

You can use the following pattern to ensure that commit messages conform to the Conventional Commits specification. For more information, see [conventionalcommits.org](https://conventionalcommits.org).

Text



```
^(build|chore|ci|docs|feat|fix|perf|refactor|revert|style|test){1}(\(([\w\-\.\.]+\))?)?(!)?:([\w ])+([\s\S]*)
```

Matches: feat: allow provided config object to extend other configs

Does not match: Add conditional logic to foo.bar

## Legal

