About push protection

Push protection blocks contributors from pushing secrets to a repository and generates an alert whenever a contributor bypasses the block. Push protection can be applied at the repository, organization, and user account level.

Who can use this feature?

Push protection for repositories and organizations is available for the following repository types:

- User-owned public repositories for free
- Private and internal repositories in organizations using GitHub Enterprise Cloud with GitHub Advanced Security enabled
- User namespace repositories belonging to Enterprise Managed Users

About push protection

Push protection is a secret scanning feature that is designed to prevent sensitive information, such as secrets or tokens, from being pushed to your repository in the first place. Unlike secret scanning, which detects secrets after they have been committed, push protection proactively scans your code for secrets during the push process and blocks the push if any are detected.

Push protection helps you avoid the risks associated with exposed secrets, like unauthorized access to resources or services. With this feature, developers get immediate feedback and can address potential issues before they become a security concern.

You can enable push protection:

- At repository/organization level, if you are a repository administrator or an organization owner. You will see alerts in the **Security** tab of your repository when a contributor to the repository bypasses push protection.
- For your account on GitHub, as a user. This type of push protection is referred to as "push protection for users". It protects you from pushing secrets to *any* public repository on GitHub, but no alerts are generated.

For information about the secrets and service providers supported by push protection, see "Supported secret scanning patterns."

How push protection works

Push protection works:

- From the command line. See "Working with push protection from the command line."
- In the GitHub UI. See "Working with push protection in the GitHub UI."
- On files uploaded onto the repository on GitHub.
- From the REST API. See "Working with push protection from the REST API."

Once enabled, if push protection detects a potential secret during a push attempt, it will block the push and provide a detailed message explaining the reason for the block. You will need to review the code in question, remove any sensitive information, and reattempt the push.

By default, anyone with write access to the repository can choose to bypass push protection by specifying one of the bypass reasons outlined in the table. If a contributor bypasses a push protection block for a secret, GitHub:

- Creates an alert in the **Security** tab of the repository.
- Adds the bypass event to the audit log.
- Sends an email alert to organization or personal account owners, security managers, and repository administrators who are watching the repository, with a link to the secret and the reason why it was allowed.

This table shows the behavior of alerts for each way a user can bypass a push protection block.

Bypass reason	Alert behavior
It's used in tests	GitHub creates a closed alert, resolved as "used in tests"
It's a false positive	GitHub creates a closed alert, resolved as "false positive"
I'll fix it later	GitHub creates an open alert

If you want greater control over which contributors can bypass push protection and which pushes containing secrets should be allowed, you can enable delegated bypass for push protection. Delegated bypass lets you configure a designated group of reviewers to oversee and manage requests to bypass push protection from contributors pushing to the repository. For more information, see "About delegated bypass for push protection."

You can also bypass push protection using the REST API. For more information, see "REST API endpoints for secret scanning."

About the benefits of push protection

- **Preventative security**: Push protection acts as a frontline defense mechanism by scanning code for secrets at the time of the push. This preventative approach helps to catch potential issues before they are merged into your repository.
- **Immediate feedback**: Developers receive instant feedback if a potential secret is detected during a push attempt. This immediate notification allows for quick remediation, reducing the likelihood of sensitive information being exposed.
- **Reduced risk of data leaks**: By blocking commits that contain sensitive information, push protection significantly reduces the risk of accidental data leaks. This helps in safeguarding against unauthorized access to your infrastructure, services, and data.
- **Efficient secret management**: Instead of retrospectively dealing with exposed secrets, developers can address issues at the source. This makes secret management more efficient and less time-consuming.
- **Integration with CI/CD pipelines**: Push Protection can be integrated into your Continuous Integration/Continuous Deployment (CI/CD) pipelines, ensuring that every push is scanned for secrets before it gets deployed. This adds an extra layer of security to your DevOps practices.
- **Ability to detect custom patterns**: Organizations can define custom patterns for detecting secrets unique to their environment. This customization ensures that push Protection can effectively identify and block even non-standard secrets.
- **Delegated bypass for flexibility**: For cases where false positives occur or when certain patterns are necessary, the delegated bypass feature allows designated users to approve

specific pushes. This provides flexibility without compromising overall security.

Every user across GitHub can also enable push protection for themselves within their individual settings. Enabling push protection for your user account means that your pushes are protected whenever you push to a public repository on GitHub, without relying on that repository to have push protection enabled. For more information, see "Push protection for users."

Customizing push protection

Once push protection is enabled, you can customize it further:

Integrate with CI/CD pipelines

Integrate push protection with your Continuous Integration/Continuous Deployment (CI/CD) pipelines to ensure that it runs scans during automated processes. This typically involves adding steps in your pipeline configuration file to call GitHub's APIs or using GitHub Actions.

Define custom patterns

Define custom patterns that push protection can use to identify secrets and block pushes containing these secrets. For more information, see "Defining custom patterns for secret scanning."

Configure delegated bypass

Define contributors who can bypass push protection and add an approval process for other contributors. For more information, see "About delegated bypass for push protection."

Further reading

- "Enabling push protection for your repository"
- "Working with push protection from the command line"
- "Working with push protection in the GitHub UI"
- "Defining custom patterns for secret scanning"
- "About delegated bypass for push protection"

Help and support