About commit signature verification

In this article

Using GPG, SSH, or S/MIME, you can sign tags and commits locally. These tags or commits are marked as verified on GitHub so other people can be confident that the changes come from a trusted source.

About commit signature verification

You can sign commits and tags locally, to give other people confidence about the origin of a change you have made. If a commit or tag has a GPG, SSH, or S/MIME signature that is cryptographically verifiable, GitHub marks the commit or tag "Verified" or "Partially verified."

For most individual users, GPG or SSH will be the best choice for signing commits. S/MIME signatures are usually required in the context of a larger organization. SSH signatures are the simplest to generate. You can even upload your existing authentication key to GitHub to also use as a signing key. Generating a GPG signing key is more involved than generating an SSH key, but GPG has features that SSH does not. A GPG key can expire or be revoked when no longer used. GitHub shows commits that were signed with such a key as "Verified" unless the key was marked as compromised. SSH keys don't have this capability.

Commits and tags have the following verification statuses, depending on whether you have enabled vigilant mode. By default vigilant mode is not enabled. For information on how to enable vigilant mode, see "Displaying verification statuses for all of your commits."

Signing commits differs from signing off on a commit. For more information about signing off on commits, see "Managing the commit signoff policy for your repository."

Default statuses

Status	Description
Verified	The commit is signed and the signature was successfully verified.

Unverified	The commit is signed but the signature could not be verified.
No verification status	The commit is not signed.

Signature verification for rebase and merge

When using the **Rebase and Merge** option on a pull request, it's important to note that the commits in the head branch are added to the base branch without commit signature verification. When you use this option, GitHub creates a modified commit, using the data and content of the original commit. This means that GitHub didn't truly create this commit, and can't therefore sign it as a generic system user. GitHub doesn't have access to the committer's private signing keys, so it can't sign the commit on the user's behalf.

A workaround for this is to rebase and merge locally, and then push the changes to the pull request's base branch.

For more information, see "About merge methods on GitHub."

Statuses with vigilant mode enabled

Status	Description
Verified	The commit is signed, the signature was successfully verified, and the committer is the only author who has enabled vigilant mode.
Partially verified	The commit is signed, and the signature was successfully verified, but the commit has an author who: a) is not the committer and b) has enabled vigilant mode. In this case, the commit signature doesn't guarantee the consent of the author, so the commit is only partially verified.
Unverified	Any of the following is true: - The commit is signed but the signature could not be verified. - The commit is not signed and the committer has enabled vigilant mode. - The commit is not signed and an author has enabled vigilant mode.

Repository administrators can enforce required commit signing on a branch to block all commits that are not signed and verified. For more information, see "About protected branches."

You can check the verification status of your signed commits or tags on GitHub and view why your commit signatures might be unverified. For more information, see "Checking your commit and tag signature verification status."

GitHub will automatically use GPG to sign commits you make using the web interface. Commits signed by GitHub will have a verified status. You can verify the signature locally using the public key available at https://github.com/web-flow.gpg. The full fingerprint of the key is `5DE3 E050 9C47 EA3C F04A 42D3 4AEE 18F8 3AFD EB23`.

You can optionally choose to have GitHub GPG sign commits you make in GitHub Codespaces. For more information about enabling GPG verification for your codespaces, see "Managing GPG verification for GitHub Codespaces."

GPG commit signature verification

You can use GPG to sign commits with a GPG key that you generate yourself.

GitHub uses OpenPGP libraries to confirm that your locally signed commits and tags are cryptographically verifiable against a public key you have added to your account on GitHub.com.

To sign commits using GPG and have those commits verified on GitHub, follow these steps:

1. Check for existing GPG keys
2. Generate a new GPG key
3. Add a GPG key to your GitHub account
4. Tell Git about your signing key
5. Sign commits
6. Sign tags

SSH commit signature verification

You can use SSH to sign commits with an SSH key that you generate yourself. For more information, see the Git reference documentation for `user.Signingkey`. If you already use an SSH key to authenticate with GitHub, you can also upload that same key again for use as a signing key. There's no limit on the number of signing keys you can add to your account.

GitHub uses ssh_data, an open source Ruby library, to confirm that your locally signed commits and tags are cryptographically verifiable against a public key you have added to your account on GitHub.com.

> **Note:** SSH signature verification is available in Git 2.34 or later. To update your version of Git, see the Git website.

To sign commits using SSH and have those commits verified on GitHub, follow these steps:

1. Check for existing SSH keys
2. Generate a new SSH key
3. Add a SSH signing key to your GitHub account
4. Tell Git about your signing key
5. Sign commits
6. Sign tags

S/MIME commit signature verification

You can use S/MIME to sign commits with an X.509 key issued by your organization.

GitHub uses the Debian ca-certificates package, the same trust store used by Mozilla browsers, to confirm that your locally signed commits and tags are cryptographically verifiable against a public key in a trusted root certificate.

> **Note:** S/MIME signature verification is available in Git 2.19 or later. To update your version of Git, see the Git website.

To sign commits using S/MIME and have those commits verified on GitHub, follow these steps:

1. Tell Git about your signing key
2. Sign commits
3. Sign tags

You don't need to upload your public key to GitHub.

Signature verification for bots

Organizations and GitHub Apps that require commit signing can use bots to sign commits. If a commit or tag has a bot signature that is cryptographically verifiable, GitHub marks the commit or tag as verified.

Signature verification for bots will only work if the request is verified and authenticated as the GitHub App or bot and contains no custom author information, custom committer information, and no custom signature information, such as Commits API.

Further reading

- "Signing commits"
- "Signing tags"
- "Troubleshooting commit signature verification"