Article version: Free, Pro, and Team

GitHub.com / Authentication / Keeping your account and data secure /
Removing sensitive data from a repository

Removing sensitive data from a repository

If you commit sensitive data, such as a password or SSH key into a Git repository, you can remove it from the history. To entirely remove unwanted files from a repository's history you can use either the `git filter-branch` command or the BFG Repo-Cleaner open source tool.

In this article

The `git filter-branch` command and the BFG Repo-Cleaner rewrite your repository's history, which changes the SHAs for existing commits that you alter and any dependent commits. Changed commit SHAs may affect open pull requests in your repository. We recommend merging or closing all open pull requests before removing files from your repository.

You can remove the file from the latest commit with `git rm`. For information on removing a file that was added with the latest commit, see "Removing files from a repository's history."

Warning: Once you have pushed a commit to GitHub, you should consider any data it contains to be compromised. If you committed a password, change it! If you committed a key, generate a new one.

This article tells you how to make commits with sensitive data unreachable from any branches or tags in your GitHub repository. However, it's important to note that those commits may still be accessible in any clones or forks of your repository, directly via their SHA-1 hashes in cached views on GitHub, and through any pull requests that reference them. You can't do anything about existing clones or forks of your repository, but you can permanently remove cached views and references to the sensitive data in pull requests on GitHub by contacting GitHub Support or GitHub Premium Support.

Purging a file from your repository's history

Using the BFG

The BFG Repo-Cleaner is a tool that's built and maintained by the open source community. It provides a faster, simpler alternative to `git filter-branch` for removing unwanted data. For example, to remove your file with sensitive data and leave your latest commit untouched, run:

```
$ bfg --delete-files YOUR-FILE-WITH-SENSITIVE-DATA
```

To replace all text listed in `passwords.txt` wherever it can be found in your repository's history, run:

```
$ bfg --replace-text passwords.txt
```

After the sensitive data is removed, you must force push your changes to GitHub.

```
$ git push --force
```

See the BFG Repo-Cleaner's documentation for full usage and download instructions.

Using filter-branch

Warning: If you run `git filter-branch` after stashing changes, you won't be able to retrieve your changes with other stash commands. Before running `git filter-branch`, we recommend unstashing any changes you've made. To unstash the last set of changes you've stashed, run `git stash show -p | git apply -R`. For more information, see Git Tools Stashing.

To illustrate how `git filter-branch` works, we'll show you how to remove your file with sensitive data from the history of your repository and add it to `.gitignore` to ensure that it is not accidentally re-committed.

1 If you don't already have a local copy of your repository with sensitive data in its history, clone the repository to your local computer.

```
$ git clone https://github.com/YOUR-USERNAME/YOUR-REPOSITORY
> Initialized empty Git repository in /Users/YOUR-FILE-PATH/YOUR-REPOSITORY/.git/
> remote: Counting objects: 1301, done.
> remote: Compressing objects: 100% (769/769), done.
> remote: Total 1301 (delta 724), reused 910 (delta 522)
> Receiving objects: 100% (1301/1301), 164.39 KiB, done.
> Resolving deltas: 100% (724/724), done.
```

2 Navigate into the repository's working directory.

```
$ cd YOUR-REPOSITORY
```

3 Run the following command, replacing `PATH-TO-YOUR-FILE-WITH-SENSITIVE-DATA` with the **path to the file you want to remove, not just its filename**. These arguments will:

- Force Git to process, but not check out, the entire history of every branch and tag
- Remove the specified file, as well as any empty commits generated as a result
- **Overwrite your existing tags**

```
$ git filter-branch --force --index-filter \
  "git rm --cached --ignore-unmatch PATH-TO-YOUR-FILE-WITH-SENSITIVE-DATA" \
  --prune-empty --tag-name-filter cat -- --all
  > Rewrite 48dc599c80e20527ed902928085e7861e6b3cbe6 (266/266)
  > Ref 'refs/heads/main' was rewritten
```

Note: If the file with sensitive data used to exist at any other paths (because it was moved or renamed), you must run this command on those paths, as well.

4 Add your file with sensitive data to `.gitignore` to ensure that you don't accidentally commit it again.

```
$ echo "YOUR-FILE-WITH-SENSITIVE-DATA" >> .gitignore
$ git add .gitignore
$ git commit -m "Add YOUR-FILE-WITH-SENSITIVE-DATA to .gitignore"
> [main 051452f] Add YOUR-FILE-WITH-SENSITIVE-DATA to .gitignore
>  1 files changed, 1 insertions(+), 0 deletions(-)
```

5 Double-check that you've removed everything you wanted to from your repository's history, and that all of your branches are checked out.

6 Once you're happy with the state of your repository, force-push your local changes to overwrite your GitHub repository, as well as all the branches you've pushed up:

```
$ git push origin --force --all
> Counting objects: 1074, done.
> Delta compression using 2 threads.
> Compressing objects: 100% (677/677), done.
> Writing objects: 100% (1058/1058), 148.85 KiB, done.
> Total 1058 (delta 590), reused 602 (delta 378)
> To https://github.com/YOUR-USERNAME/YOUR-REPOSITORY.git
>  + 48dc599...051452f main -> main (forced update)
```

7 In order to remove the sensitive file from your tagged releases, you'll also need to force-push against your Git tags:

```
$ git push origin --force --tags
> Counting objects: 321, done.
> Delta compression using up to 8 threads.
> Compressing objects: 100% (166/166), done.
> Writing objects: 100% (321/321), 331.74 KiB | 0 bytes/s, done.
> Total 321 (delta 124), reused 269 (delta 108)
> To https://github.com/YOUR-USERNAME/YOUR-REPOSITORY.git
>  + 48dc599...051452f main -> main (forced update)
```

8 Contact GitHub Support or GitHub Premium Support, asking them to remove cached views and references to the sensitive data in pull requests on GitHub.

9 Tell your collaborators to rebase, *not* merge, any branches they created off of your old (tainted) repository history. One merge commit could reintroduce some or all of the tainted history that you just went to the trouble of purging.

10 After some time has passed and you're confident that `git filter-branch` had no unintended side effects, you can force all objects in your local repository to be dereferenced and garbage collected with the following commands (using Git 1.8.5 or newer):

```
$ git for-each-ref --format="delete %(refname)" refs/original | git update-ref --stdin
$ git reflog expire --expire=now --all
$ git gc --prune=now

> Counting objects: 2437, done.
> Delta compression using up to 4 threads.
> Compressing objects: 100% (1378/1378), done.
> Writing objects: 100% (2437/2437), done.
> Total 2437 (delta 1461), reused 1802 (delta 1048)
```

Note: You can also achieve this by pushing your filtered history to a new or empty repository and then making a fresh clone from GitHub.

Avoiding accidental commits in the future

There are a few simple tricks to avoid committing things you don't want committed:

- Use a visual program like GitHub Desktop or gitk to commit changes. Visual programs generally make it easier to see exactly which files will be added, deleted, and modified with each commit.
- Avoid the catch-all commands `git add .` and `git commit -a` on the command line—use `git add filename` and `git rm filename` to individually stage files, instead.
- Use `git add --interactive` to individually review and stage changes within each file.
- Use `git diff --cached` to review the changes that you have staged for commit. This is the exact diff that `git commit` will produce as long as you don't use the `-a` flag.

Further reading

- `git filter-branch` man page
- Pro Git: Git Tools - Rewriting History

Did this doc help you?

Privacy policy

Help us make these docs great!

All GitHub docs are open source. See something that's wrong or unclear? Submit a pull request.

Make a contribution

Or, learn how to contribute.

© 2020 GitHub, Inc. Terms Privacy Security Status Help Contact GitHub Pricing Developer API Training About