


We have updated our privacy policy, you can read our new policy [here](https://itextpdf.com/privacy-policy). (<https://itextpdf.com/privacy-policy>).

How to completely remove a file from a Git repository

Posted: Jul 3 2017 - 11:05 Tags:  git

Intro

Have you already committed an SSH private key, a password file or a config file with sensitive data to your repository before? In case you did not, I would recommend to first try this out before you continue reading this blogpost.

For the rest of us: DON'T PANIC! Take a deep breath, get up from your desk, walk around for a few minutes. Ready? Okay, let's get started!

The goal is to completely wipe a file out of existence in a Git repository, to cover all tracks of your horrible mistake. Do you want to be the person who committed AWS keys to a public GitHub repository (<http://blog.roundingpegs.com/how-i-avoid-committing-passwords-to-github/>), only to find out 24 hours later that ~USD2000 has been spent mining bitcoins?

Several methods

Simply `git rm passwords.txt` won't do it, as the file will still be there in all previous commits. Depending on where the file is, you can use several methods. Hereby an overview per scenario in increasing order of complexity.

Notes:

- *All of these methods assume that you are familiar with console commands.*
- *They are written for Linux, but should work on OS X and even on Windows if you use Git Bash.*
- *If you have already pushed your changes, then things might become complicated.*
- *If you're a single developer, just go for it. But if you work in a team then first talk it over with them.*
- *If your code (with unwanted file) is already out in the open (GitHub, BitBucket,...) then you might be out of luck. But keep reading until the end.*

Scenario 1: the file is in the last commit and you have not yet pushed

1. You want to keep the file locally

Amend the last commit to remove the file from the repository, and add it to `.gitignore`, to prevent it from being added by accident again.

```
git rm --cached $FILE
echo $FILE >> .gitignore
git add .gitignore
git commit --amend --no-edit
git reflog expire --expire=now --all && git gc --prune=now --aggressive
```

The `git reflog expire` and `git gc` commands force a garbage collection, to keep the file from dangling somewhere in your repository.

2. You do not want to keep the file locally

Just amend the last commit.

```
git rm $FILE
git commit --amend --no-edit
git reflog expire --expire=now --all && git gc --prune=now --aggressive
```

Scenario 2: the file is further down in the history and you have not yet pushed

Solution 1: BFG Repo-Cleaner

Download 'BFG Repo-Cleaner' here (<https://rtyley.github.io/bfg-repo-cleaner/>). This tool claims to work 10-720x faster than any other method, but you cannot specify a subdirectory, it will delete all files with the same name in any directory.

Normally BFG Repo-Cleaner protects your most recent commit, but if you know what you are doing (and you know, right?) then you give it the option `--no-blob-protection`, which is sort of like saying, do what I want and don't protect me from mistakes.

```
java -jar bfg.jar --delete-files $FILE --no-blob-protection .
rm $FILE
git reflog expire --expire=now --all && git gc --prune=now --aggressive
```

Solution 2: interactive rebase

An interactive rebase lets you go back in history and redo commits, as if they were correct in the first place. Sounds like cheating? Maybe. But you want to get rid of that file, and you have the power to do it.

This command has a subshell inside `$(...)` : it finds the first commit where the file was added, even taking renames of the file into account. Then the command outside `$(...)` starts an interactive rebase at the parent (`~`) of that commit.

```
git rebase --interactive \
$(git log --follow --find-renames=40% --diff-filter=A --format=%H -- $FILE)~
```

Edit the `git-rebase-todo` file: change the first command from pick to edit.

```
edit 7b0a4be987 Add a new table width tests
pick 38737174a7 Avoid FontProgram#getBaseName method usage when it's not appropriate, update doc
umentation
pick 66f95dd41b Generalize reading of bytes for woff format conversion

# Rebase d1613dff58..66f95dd41b onto d1613dff58 (3 commands)
#
# Commands:
# p, pick = use commit
# r, reword = use commit, but edit the commit message
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
# f, fixup = like "squash", but discard this commit's log message
# x, exec = run command (the rest of the line) using shell
# d, drop = remove commit
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out
```

Then save and close the editor. The rebase will roll back the commits to the commit that added the unwanted file.

```
git rm $FILE
echo $FILE >> .gitignore
git add .gitignore
git commit --amend --no-edit
git rebase --continue
git reflog expire --expire=now --all && git gc --prune=now --aggressive
```

Solution 3: Git filter-branch

This runs a script specified in `--tree-filter` (f.ex: delete a certain file) on every commit. This is really SLOW! That's because it has to checkout every commit, run the script, commit, and move on to the next commit. Use this when there are tags or merge commits between the offending commit and HEAD, or

when the offending file exists in multiple branches. In other words, use as a last resort!

In this script I use the same trick to find the first commit that adds the unwanted file, and then I run the filter only on the *parent* of that commit, up to the `HEAD` of the current branch. `git filter-branch` always makes a backup and prefixes the original branch with `original`. The `git for-each-ref` command cleans up those backup branches, because we really want to get rid of that pesky file. `--tag-name-filter cat` will make sure that any tags will move with their commits.

```
git filter-branch -f \
  --prune-empty \
  --tag-name-filter cat \
  --tree-filter 'rm -f $FILE' \
  $(git log --follow --find-renames=40% --diff-filter=A --format=%H -- $FILE)~..HEAD
git for-each-ref --format="%(refname)" refs/original/ | xargs -n 1 git update-ref -d
git reflog expire --expire=now --all && git gc --prune=now --aggressive
```

If you want to do it for the entire repository (you really shouldn't!):

```
git filter-branch -f \
  --prune-empty \
  --tag-name-filter cat \
  --tree-filter 'rm -f $FILE' \
  -- --all
git for-each-ref --format="%(refname)" refs/original/ | xargs -n 1 git update-ref -d
git reflog expire --expire=now --all && git gc --prune=now --aggressive
```

Scenario 3: you have already pushed

- I can't repeat this enough: first consult with the rest of your team!!! You will make their life miserable if you don't.
- Secondly, make a backup of the repository before you do anything else.
- Then: MAKE A BACKUP!!!
- I don't know if I have mentioned this already, but are you really, really sure that you have made a backup?
- Use one of the methods described above to remove the file.
- Force push, but know what you are doing and be aware of the consequences because there is no way to go back if you don't have a backup. `--force-with-lease` should actually be the default there, because it first checks that you won't overwrite other people's work. See also the Atlassian blog (<https://developer.atlassian.com/blog/2015/04/force-with-lease/>) for an excellent explanation.

```
git push --force-with-lease origin $BRANCH
```

Scenario 4: the commits are already on GitHub

In this case, the risk exists that someone can still access the unwanted file, even after a force push.

There are two ways to do that:

- if they made a fork or a clone of the repository: a force push will only update our repository, it will not update forks/clones.
- if they happen to know the exact hash of the commit that added the file (maybe they wrote it down or the web page is still in their browser cache), via the URL:

`https://github.com/$USER/$REPO/commit/$COMMIT`

(`https://github.com/$USER/$REPO/commit/$COMMIT`)

GitHub may garbage collect cached views after some time, but this is not something to rely on.

The best thing to do, is to contact GitHub support (<https://help.github.com/articles/removing-sensitive-data-from-a-repository/>) and tell them the repository and the offending commit, they will then manually delete the cached views.

What to expect next?

In the follow-up blogpost, I will explain some best practices used at iText Software to handle sensitive files needed in our projects.

Authors

amedee.vangasse@itextpdf.com



GET PRODUCTS

Product Overview (<https://itextpdf.com/itext7suite>)

Downloads (<https://developers.itextpdf.com/downloads>)

Uninstall iText (<https://developers.itextpdf.com/question/how-can-i-uninstall-itext>)

HOW TO BUY

Explore Pricing (<https://itextpdf.com/Pricing>)

Request a Quote (<https://itextpdf.com/Request-A-Quote>)

Find a Partner (<https://itextpdf.com/partner>)

CUSTOMERS

Customer Portal (<https://itextpdf.com>)

Customer Support (<https://itextpdf.com/support>)

FAQ (<https://itextpdf.com/customer-faq>)

DEVELOPERS

Developer Portal (<https://developers.itextpdf.com>)

Q&A (<https://developers.itextpdf.com/frequently-asked-developer-questions-7>)

Release Notes (<https://itextpdf.com/posts/release-note-and-changelogs>)

API Docs (<https://developers.itextpdf.com/apis>)

COMPANY

About us (<https://itextpdf.com/about>)

Blog (<https://itextpdf.com/posts/general>)

Press Releases (<https://itextpdf.com/posts/press-and-news>)

Events (<https://itextpdf.com/posts/events>)

Awards (<https://itextpdf.com/awards>)

Legal (<https://itextpdf.com/legal>)

Contact Us (<https://itextpdf.com/contact>)

Jobs (<https://itextpdf.com/jobs>)

Newsletters (<https://itextpdf.com/itext-newsletters>)

Asia, Oceania

3 Fraser Street
Duo Tower #08-21
Singapore 189352
sales.isa@itextpdf.com (<mailto:sales.isa@itextpdf.com>)
Tel: +65 6828 1430 (tel:+6568281430)
Fax: +65 6828 1401
201510495G

Europe, Middle East, Africa

Business Center "De Punt"
Kerkstraat 108
9050 Gentbrugge (Ghent)
Belgium
sales.isb@itextpdf.com (<mailto:sales.isb@itextpdf.com>)
Tel: +32 9 298 02 31 (tel:+3292980231)
Fax: +32 9 270 33 75
0838.649.627

North America, South America

265 Medford St, Suite 401
Somerville, MA 02143
USA
sales.isc@itextpdf.com (<mailto:sales.isc@itextpdf.com>)
Tel: +1 617 982 2646 (tel:+16479822646)
Fax: +1 617 982 2647
26-4352833

© 2018,

iText Group NV | Privacy Policy (<https://itextpdf.com/privacy-policy>) | Cookie Policy
(<https://itextpdf.com/cookies-policy>) | Terms of Use (<https://itextpdf.com/end-user-license-agreement>)



(<https://www.flickr.com/photos/itextinaction/>)



(https://www.youtube.com/channel/UC6kL1_Vm712V3XDM1_RSY8w)



(<https://www.pinterest.com/itext/>)



(<https://www.facebook.com/iTextPDF/>)



(<https://plus.google.com/+itext>)



(<http://www.slideshare.net/iTextPDF>)



(<https://www.linkedin.com/company-beta/281131/>)

(<https://twitter.com/iText>)