Make the current commit the only (initial) commit in a Git repository?

I currently have a local Git repository, which I push to a Github repository.

The local repository has ~10 commits, and the Github repository is a synchronised duplicate of this.

What I'd like to do is remove ALL the version history from the local Git repository, so the current contents of the repository appear as the only commit (and therefore older versions of files within the repository are not stored).

I'd then like to push these changes to Github.

I have investigated Git rebase, but this appears to be more suited to removing specific versions. Another potential solution is to delete the local repo, and create a new one - though this would probably create a lot of work!

ETA: There are specific directories / files that are untracked - if possible I would like to maintain the untracking of these files.

git github git-commit

edited Feb 19 '17 at 20:54

jsroyal
435 6 17

asked Mar 13 '12 at 11:41

kaese
2,969 7 21 34

6 See also stackoverflow.com/questions/435646/… ("How do I combine the first two commits of a Git repository?") – Anonymoose Mar 13 '12 at 11:56

Possibly related: How do I combine the first two commits of a Git repository?. – user456814 May 15 '14 at 8:59

and this: How to squash all git commits into one? – ryenus Nov 17 '17 at 4:15

13 Answers

Here's the brute-force approach. It also removes the configuration of the repository.

Note: This does NOT work if the repository has submodules! If you are using submodules, you should use e.g. interactive rebase

Step 1: remove all history (**Make sure you have backup, this cannot be reverted**)

```
rm -rf .git
```

Step 2: reconstruct the Git repo with only the current content

```
git init
git add .
git commit -m "Initial commit"
```

Step 3: push to GitHub.

```
git remote add origin <github-uri>
git push -u --force origin master
```

edited Sep 6 '17 at 9:13

community wiki
6 revs, 3 users 76%
Fred Foo

3 Thanks larsmans - I have opted to use this as my solution. Though initialising the Git repo loses record of untracked files in the old repo, this is probably a simpler solution for my problem. – kaese Mar 13 '12 at 12:14

5 @kaese: I think your `.gitignore` should handle those, right? – Fred Foo Mar 13 '12 at 12:44

39 Save your .git/config before, and restore it after. – lalebarde Apr 19 '14 at 8:35

Just tried this and it worked. – Dan Dascalescu Aug 28 '15 at 20:59

15 Be careful with this if you are trying to remove sensitive data: the presence of only a single commit in the newly pushed master branch is **misleading - the history will still exist** it just won't be accessible from that branch. If you have tags, for example, which point to older commits, these commits will be accessible. In fact, for anyone with a bit of git foo, I'm sure that after this git push, they will still be able to recover all history from the GitHub repository - and if you have other branches or tags, then they don't even need much git foo. – Robert Muil May 3 '16 at 9:05

The only solution that works for me (and keeps submodules working) is

```
git checkout --orphan newBranch
git add -A  # Add all files and commit them
git commit
git branch -D master  # Deletes the master branch
git branch -m master  # Rename the current branch to master
git push -f origin master  # Force push master branch to github
git gc --aggressive --prune=all     # remove the old files
```

Deleting `.git/` always causes huge issues when I have submodules. Using `git rebase --root` would somehow cause conflicts for me (and take long since I had a lot of history).

edited Dec 29 '16 at 14:46 — MatthewG — 3,098 14 21

answered Oct 27 '12 at 18:16 — Zeelot — 4,254 1 9 4

34 this should be the correct answer! just add a `git push -f origin master` as the last op and sun will shine again on your fresh repo! :) – gru Jan 7 '14 at 13:30

4 @JonePolvora git fetch; git reset --hard origin/master stackoverflow.com/questions/4785107/… – echo May 2 '14 at 17:41

2 after doing this, will the repo free space? – Inuart Aug 28 '14 at 15:32

2 @Brad: I was wondering the same thing. Seems that --orphan only gets the last commit so that the remaining ones are thrown away when the branch is deleted. Reference: git-scm.com/docs/git-checkout – Hirnhamster Nov 23 '14 at 10:24

3 I believe you should add @JasonGoemaat 's suggestion as the last line to your answer. Without `git gc --aggressive --prune all` the whole point of losing history would be missed. – Tuncay Göncüoğlu Oct 22 '16 at 12:19

This is my favoured approach:

```
git branch new_branch_name $(echo "commit message" | git commit-tree HEAD^{tree})
```

This will create a new branch with one commit that adds everything in HEAD. It doesn't alter anything else, so it's completely safe.

answered Mar 22 '13 at 13:53 — dan_waterworth — 4,363 19 36

2 Best approach! Clear, and do the work. Additionally, i rename the branch with a lot of changes from "master" to "local-work" and "new_branch_name" to "master". In master, do following: git -m local-changes git branch -m local-changes git checkout new_branch_name git branch -m master< – Valtoni Boaventura Feb 9 '15 at 12:46

This looks really short and sleek, the only thing I don't understand or haven't seen yet is HEAD^{tree}, could somebody explain? Apart from that I'd read this as "create new branch from given commit, created by creating a new commit-object with given commit-message from ___" – TomKeegasi Feb 15 '17 at 8:10

2 The definitive place to look for answers to questions about git reference syntax is in the `git-rev-parse` docs. What's happening here is `git-commit-tree` requires a reference to a tree (a snapshot of the repo), but `HEAD` is a revision. To find the tree associated with a commit we use the `<rev>^{<type>}` form. – dan_waterworth Feb 15 '17 at 20:48

Nice answer. Works well. Finally say `git push --force <remote> new_branch_name:<remote-branch>` – Felipe Alvarez Dec 11 '17 at 22:01

The other option, which could turn out to be a lot of work if you have a lot of commits, is an interactive rebase (assuming your git version is >=1.7.12): `git rebase --root -i`

- Change "pick" to "reword" for the first commit
- Change "pick" to "fixup" every other commit

Save and close. Git will start rebasing.

At the end you would have a new root commit that is a combination of all the ones that came after it.

The advantage is that you don't have to delete your repository and if you have second thoughts you always have a fallback.

If you really do want to nuke your history, reset master to this commit and delete all other branches.

edited May 21 '14 at 21:59

answered Mar 14 '12 at 20:24 — Carl — 27.6k 6 60 89

Variant of *larsmans*'s proposed method:

Save your untrackfiles list:

```
git ls-files --others --exclude-standard > /tmp/my_untracked_files
```

Save your git configuration:

```
mv .git/config /tmp/
```

Then perform larsmans's first steps:

```
rm -rf .git
git init
git add .
```

Restore your config:

```
mv /tmp/config .git/
```

Untrack you untracked files:

```
cat /tmp/my_untracked_files | xargs -0 git rm --cached
```

Then commit:

```
git commit -m "Initial commit"
```

And finally push to your repository:

```
git push -u --force origin master
```

edited Jan 30 '15 at 22:47

emotality
8,690 4 22 42

answered Apr 19 '14 at 8:58

lalebarde
781 9 27

You could use shallow clones (git > 1.9):

```
git clone --depth depth remote-url
```

Further reading: http://blogs.atlassian.com/2014/05/handle-big-repositories-git/

edited Apr 4 '16 at 11:12

answered Apr 4 '16 at 11:05

Matthias M
2,916 4 37 44

1 Such clone can not be pushed to a new repository. – Seweryn Niemiec Nov 9 '16 at 20:53

Create a branch, copy all the contents to it, commit it and then delete the master branch:

```
git checkout --orphan newBranch; git add -A ;git commit -am 'first commit' ;git branch -D
master;git branch -m master; git push -f origin master; git gc --aggressive --prune=all
```

edited Jul 11 '17 at 9:05

zb226
5,407 3 24 50

answered Mar 12 '17 at 14:53

pratik_bhavsar
59 1 3

This answer is identically to that from Zeelot – Matthias M Mar 27 at 7:42

The method below is exactly reproducible, so there's no need to run clone again if both sides were consistent, just run the script on the other side too.

```
git log -n1 --format=%H >.git/info/grafts
git filter-branch -f
rm .git/info/grafts
```

If you then want to clean it up, try this script:

http://sam.nipl.net/b/git-gc-all-ferocious

I wrote a script which "kills history" for each branch in the repository:

http://sam.nipl.net/b/git-kill-history

see also: http://sam.nipl.net/b/confirm

edited Jan 29 '15 at 3:48

answered Feb 6 '13 at 16:46

Sam Watkins
4,035 1 23 26

```
git for-each-ref --format='git update-ref -d %(refname)' \
        refs/{heads,tags} | sh -x
current=$(git commit-tree -m 'Initial commit' `git write-tree`)
git update-ref -m 'Initial commit' `git symbolic-ref HEAD` $current
```

```
git update-ref -m "initial commit" $(git symbolic-ref HEAD) $current
```

This will remove all local branches and tags, make a single no-history commit with the state of your current checkout on whatever branch is current, and leave everything else about your repo untouched. You can then force-push to your remotes as you like.

answered Apr 5 '14 at 1:42

jthill
25k 2 36 71

> What I'd like to do is remove ALL the version history from the local Git repository, so the current contents of the repository appear as the only commit (and therefore older versions of files within the repository are not stored).

A more conceptual answer:

git automatically garbage collects old commits if no tags/branches/refs point to them. So you simply have to remove all tags/branches and create a new orphan commit, associated with any branch - by convention you would let the branch `master` point to that commit.

The old, unreachable commits will then never again be seen by anyone unless they go digging with low-level git commands. If that is enough for you, I would just stop there and let the automatic GC do it's job whenever it wishes to. If you want to get rid of them right away, you can use `git gc` (possibly with `--aggressive --prune=all`). For the remote git repository, there's no way for you to force that though, unless you have shell access to their file system.

answered Apr 4 '16 at 11:27

AnoE
4,623 1 8 19

I solved a similar issue by just deleting the `.git` folder from my project and reintegrating with version control through IntelliJ. Note: The `.git` folder is hidden. You can view it in the terminal with `ls -a`, and then remove it using `rm -rf .git`.

answered Apr 11 '17 at 18:54

JB Lovell
73 5

thats what he's doing in step 1: rm -rf .git ? – nights Jul 10 '17 at 1:45

> To remove the last commit from git, you can simply run

```
git reset --hard HEAD^
```

> If you are removing multiple commits from the top, you can run

```
git reset --hard HEAD~2
```

> to remove the last two commits. You can increase the number to remove even more commits.

More info here.

Git tutoturial here provides help on how to purge repository:

> you want to remove the file from history and add it to the .gitignore to ensure it is not accidentally re-committed. For our examples, we're going to remove Rakefile from the GitHub gem repository.

```
git clone https://github.com/defunkt/github-gem.git

cd github-gem

git filter-branch --force --index-filter \
  'git rm --cached --ignore-unmatch Rakefile' \
  --prune-empty --tag-name-filter cat -- --all
```

> Now that we've erased the file from history, let's ensure that we don't accidentally commit it again.

```
echo "Rakefile" >> .gitignore

git add .gitignore

git commit -m "Add Rakefile to .gitignore"
```

> If you're happy with the state of the repository, you need to force-push the changes to overwrite the remote repository

```
git push origin master --force
```

edited Jun 5 '13 at 5:48

answered Jun 5 '13 at 5:41

octoback
12.3k 25 94 162

3 Remove files or commits from the repository has absolutely no relation with the question (which asks to remove history, a completely different thing). The OP wants a clean history but wants to preserve current state of the repository. – Victor Schröder May 22 '15 at 9:07

this does not produce the result asked in the question. you are discarding all changes after the commit you keep last and losing all changes since then, but the question asks to keep current files and drop history. – Tuncay Göncüoğlu Oct 22 '16 at 12:28

For that use Shallow Clone command git clone --depth 1 URL - It will clones only the current HEAD of the repository

answered Aug 23 '17 at 15:55

kkarki
1