Revert to a commit by a SHA hash in Git? [duplicate]

Asked 13 years, 2 months ago Modified 1 year, 2 months ago Viewed 672k times



This question already has answers here:

725

How do I revert a Git repository to a previous commit? (41 answers)



I'm not clear on how git revert works. For example, I want to revert to a commit six commits behind the head, reverting all the changes in the intermediary commits in between.

Say its <u>SHA</u> hash is 56e05fced214c44a37759efa2dfc25a65d8ae98d . Then why can't I just do something like:

git revert 56e05fced214c44a37759efa2dfc25a65d8ae98d

git

Share Edit Follow Flag

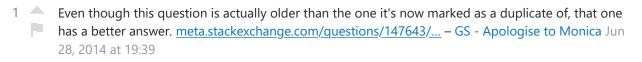
Closed 8 years ago.

edited Jan 26, 2018 at 15:54

asked Dec 12, 2009 at 23:34



JP Silvashy **46.2k** • 48 • 150 • 222



- This question and the top answer here may confuse git users. Just to help understand the terminology, you don't *revert to* a commit. You can either *reset to* a commit (which is like going back in time using time machine) or *revert* a commit (which is like pulling out a commit as if it never existed however it does preserve the revert info in history, allowing you to revert a revert if you wanted to) Note also that you shouldn't use the m flag and type a commit message if you get conflicts in the process. The auto message git provides is more informative when looking back in history. alexrogers Mar 12, 2015 at 12:30
- @alexrogins what does pulling out a commit as if it never existed mean? Not sure what 'revert a revert' refers to either appreciate the comment though, good info, just looking for more detail on your perspective. Joe Jan 23, 2018 at 17:15
- @Joe as in if you add a line of code then commit that line, if you were to revert it you would be undoing that line of code (wherever it was first written in history, doesn't have to be the last commit). That then makes a revert commit. If you revert that revert commit then you're essentially undoing the undo (i.e. redoing the original line again) alexrogers Jan 24, 2018 at 21:59

9 Answers

Sorted by: Highest score (default)



1320

If you want to commit on top of the current HEAD with the exact state at a different commit, undoing all the intermediate commits, then you can use reset to create the correct state of the index to make the commit.



```
# Reset the index and working tree to the desired tree
# Ensure you have no uncommitted changes that you want to keep
git reset --hard 56e05fced

# Move the branch pointer back to the previous HEAD
git reset --soft "HEAD@{1}"
git commit -m "Revert to 56e05fced"
```

•

Share Edit Follow Flag

edited Apr 21, 2021 at 16:06

metal

6,087 • 1 • 36 • 48

answered Dec 12, 2009 at 23:51





- When I did this I ended up with a bunch of Untracked Files in the working tree. However looking at the history I could see that those files did have a corresponding delete commit in that "Revert to SHA" commit. So after git reset —hard at the end, you can do git clean —f —d to clean up any untracked files that lingered about. Also, thank you so much this helped me solve a crisis! nzifnab Apr 27, 2012 at 19:33
- do I have to do the git reset --soft HEAD@{1} unconditionally? I mean always with a value of 1? deprecated Sep 17, 2013 at 9:23 /
- @vemv Yes, unless you want to throw away commits on the tip of the branch. git reset 56e05fced adds another entry to the reflog (run git reflog), so git reset --soft HEAD@{1} simply moves the pointer back to the HEAD prior to calling git reset 56e05fced. Using a higher number (e.g. git reset --soft HEAD@{2}) would append the new commit on a previous commit. That is, increasing the number would essentially throw away N-1 commits where N is the number you replace 1 with. 0b10011 Sep 18, 2013 at 18:37
- 5 @Tom HEAD@{1} should be quoted as 'HEAD{@1}' otherwise, it won't work for me (possibly every zsh users) jilen Apr 1, 2015 at 1:56 ▶



What git-revert does is create a commit which undoes changes made in a given commit, creating a commit which is reverse (well, reciprocal) of a given commit. Therefore

178



git revert <SHA-1>

should and does work.



If you want to rewind back to a specified commit, and you can do this because this part of history was not yet published, you need to use git-reset, not git-revert:

```
git reset --hard <SHA-1>
```

(Note that --hard would make you lose any non-committed changes in the working directory).

Additional Notes

By the way, perhaps it is not obvious, but everywhere where documentation says <commit> or <commit-ish> (or <object>), you can put an SHA-1 identifier (full or shortened) of commit.

Share Edit Follow Flag





answered Dec 13, 2009 at 9:54



302k • 65 • 219 • 230



9 In the case that you're history has already been pushed to a remote before you did the hard reset, you would need to force push the newly reset branch with git push -f , but Be Warned that this could possibly unintentionally delete other users' commits, and if not delete new commits, then it will force other users to resynchronize their work with the reset branch, so make sure this is OK with your collaborators first. – user456814 Jun 28, 2014 at 17:25 🖍

4 This seems to be the best answer. It also tells clearly the difference between git revert and git reset. - kta May 14, 2018 at 3:14



It reverts the said commit, that is, adds the commit opposite to it. If you want to checkout an earlier revision, you do:

91



git checkout 56e05fced214c44a37759efa2dfc25a65d8ae98d



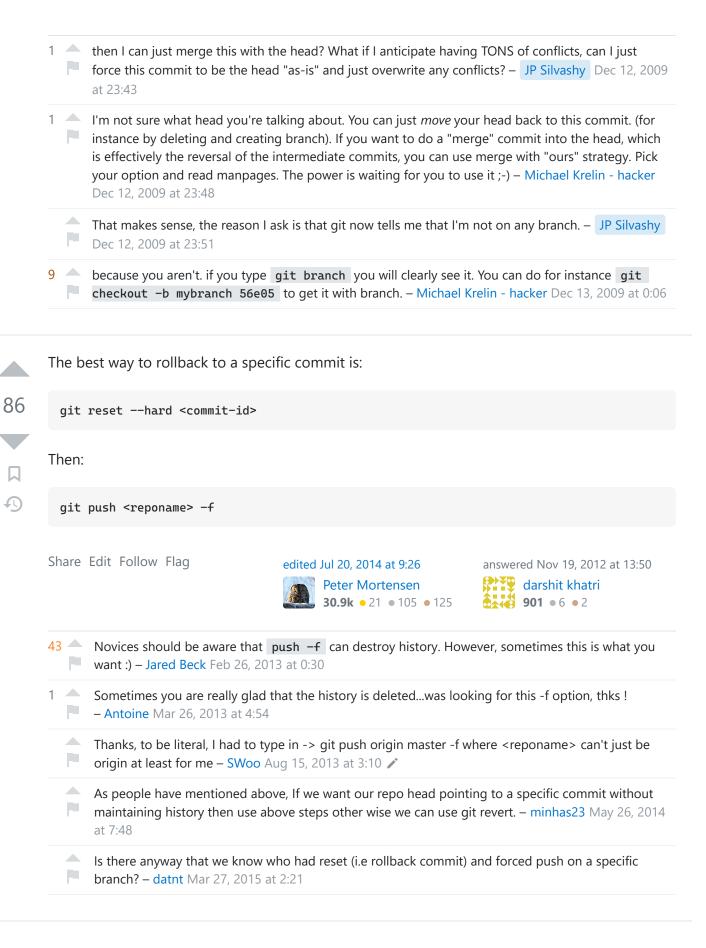
Share Edit Follow Flag

edited Jul 20, 2014 at 9:25 Peter Mortensen **30.9k** • 21 • 105 • 125 answered Dec 12, 2009 at 23:40



Michael Krelin - hacker **136k** • 24 • 192 • 173

4 of 10





If your changes have already been pushed to a **public**, **shared** remote, and you want to revert all commits between HEAD and <sha-id>, then you can pass a commit range to git revert,





git revert 56e05f..HEAD



and it will revert all commits between 56e05f and HEAD (excluding the start point of the range, 56e05f).

Share Edit Follow Flag



answered Dec 19, 2011 at 13:15
Flueras Bogdan

9,047 • 8 • 31 • 30

- Note that if you're reverting a few hundred commits, this could take a while because you have to commit each revert individually. splicer Jun 27, 2013 at 6:14
- 9 @splicer you don't have to revert each commit individually, you can either pass the --no-edit option to avoid having to make individual commit messages, or you can use --no-commit to commit the reversions all at once. user456814 Jun 28, 2014 at 17:40
 - @Cupcake you are right HEAD..56e05f doesn't work for me but 56e05f..HEAD did the trick
 Inder Kumar Rathore Aug 19, 2014 at 19:13
- This is by far my preferred way of rolling back, no matter if you pushed it or not. I added this to my global ~/.gitconfig under the aliases section: rollback = "!git revert --no-commit \$1..HEAD #" so now I can just intuitively do \$ git rollback a1s2d3 DannyB Feb 8, 2017 at 20:22
- This seems super close to what I want, but I had about 30 commits to revert, but about half way through it fails on a merge commit with error: Commit

 6b3d9b3e05a9cd9fc1dbbebdd170bf083de02519 is a merge but no -m option was given.

 fatal: revert failed any suggestions? I tried adding -m but wasn't quite sure how that would work with this Brad Parks Jan 28, 2020 at 13:03



Updated:

62

If there were no merge commits in between, this answer provides a is simpler method: https://stackoverflow.com/a/21718540/541862



But if there was one or more merge commits, that answer won't work, so stick to this one (that works in all cases).

Ori

Original answer:

```
# Create a backup of master branch
git branch backup_master

# Point master to '56e05fce' and
# make working directory the same with '56e05fce'
git reset --hard 56e05fce

# Point master back to 'backup_master' and
# leave working directory the same with '56e05fce'.
git reset --soft backup_master

# Now working directory is the same '56e05fce' and
# master points to the original revision. Then we create a commit.
git commit -a -m "Revert to 56e05fce"

# Delete unused branch
git branch -d backup_master
```

The two commands git reset --hard and git reset --soft are magic here. The first one changes the working directory, but it also changes head (the current branch) too. We fix the head by the second one.

Share Edit Follow Flag

edited Dec 2, 2021 at 3:34

Pedro A

3,695 • 3 • 33 • 55

answered Mar 22, 2013 at 5:09





This is more understandable:

5

```
git checkout 56e05fced -- .
git add .
git commit -m 'Revert to 56e05fced'
```



And to prove that it worked:

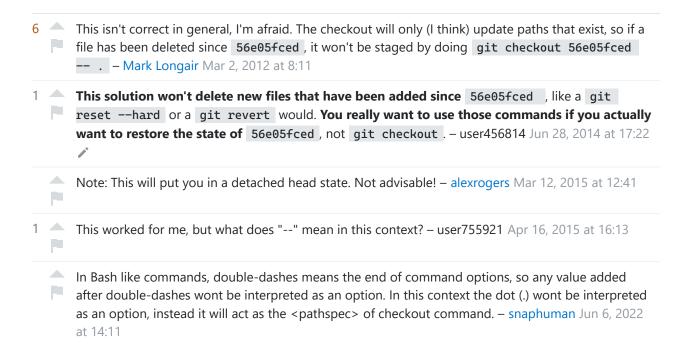
9

git diff 56e05fced

Share Edit Follow Flag

answered Jun 23, 2011 at 16:29





Should be as simple as:

1

git reset --hard 56e05f



That'll get you back to that specific point in time.

Share Edit Follow Flag

answered Jan 4, 2013 at 0:15



10.1k • 7 • 45 • 66

4 ...and is also very dangerous as will wipe all the history since including other peoples work. **Beware**of this one! – alexrogers Mar 12, 2015 at 12:42



This might work:

-2

git checkout 56e05f
echo ref: refs/heads/master > .git/HEAD
git commit



A)

Share Edit Follow Flag

answered Jun 25, 2012 at 22:36



This basically does the same thing as **git reset --hard 56e05f**, except this is less safe and more hacky. You might as well use <u>Charle's solution</u> or <u>Jakub's solution</u>. – user456814 Jun 28, 2014 at 17:14