How do I undo the most recent local commits in Git?

Asked 15 years, 6 months ago Modified 10 days ago

Viewed 15.4m times

26820

I accidentally committed the wrong files to Git but haven't pushed the commit to the server yet.

How do I undo those commits from the *local* repository?

git version-control git-commit undo

Share Follow

edited Oct 21 at 17:20

community wiki
94 revs, 65 users 11%
Peter Mortensen

801 You know what git needs? `git undo`, that's it. Then the reputation git has for handling mistakes made by us mere mortals disappears. Implement by pushing the current state on a git stack before executing any `git` command. It would affect performance, so it would be best to add a config flag as to whether to enable it. – Yimin Rong Mar 20, 2018 at 1:45

63 @YiminRong That can be done with Git's `alias` feature: git-scm.com/book/en/v2/Git-Basics-Git-Aliases – Edric Oct 5, 2018 at 14:50

165 For VsCode users , just type ctrl +shift +G and then click on three dot ,ie , more options and then click on undo Last Commit – ashad Apr 8, 2019 at 12:15

38 @YiminRong Undo *what* exactly? There are dozens of very different functional cases where "undoing" means something **completely** different. I'd bet adding a new fancy "magic wand" would only confuse things more. – Romain Valeri Mar 24, 2020 at 14:27

61 @YiminRong Not buying it. People would still fumble and undo things not to be undone. But more importantly, `git reflog` is already close to what you describe, but gives the user more control on what's to be (un)done. But please, no, "undo" does not work the same everywhere, and people would *expect* many different things for the feature to achieve. Undo last commit? Undo last action? If last action was a push, undo how exactly, (reset and push) or (revert and push)? – Romain Valeri Mar 25, 2020 at 13:23

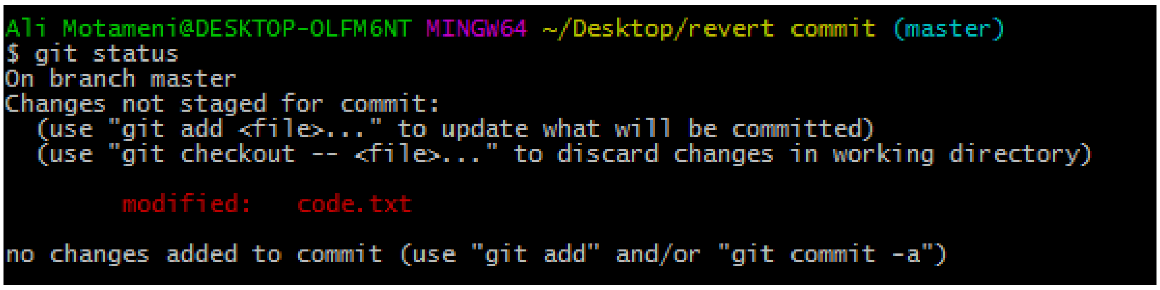
101 Answers

Sorted by: Highest score (default)

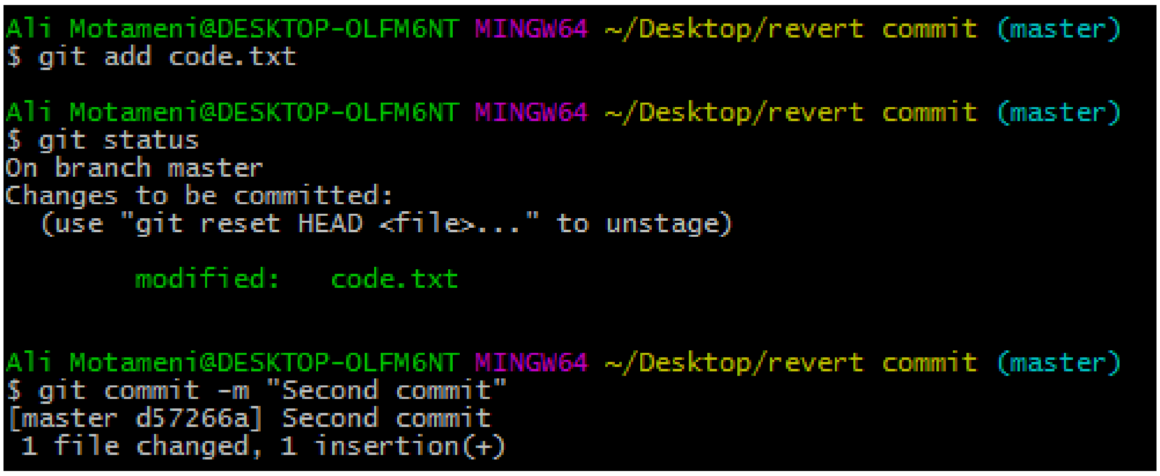
Prev 1 2 3 4 Next

103

Think we have ***code.txt*** file. We make some changes on it and commit. **We can undo this commit in three ways**, but first you should know what is the staged file... An staged file is a file that ready to commit and if you run `git status` this file will be shown with green color and if this is not staged for commit will be shown with red color:


```
Ali Motameni@DESKTOP-OLFM6NT MINGW64 ~/Desktop/revert commit (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   code.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

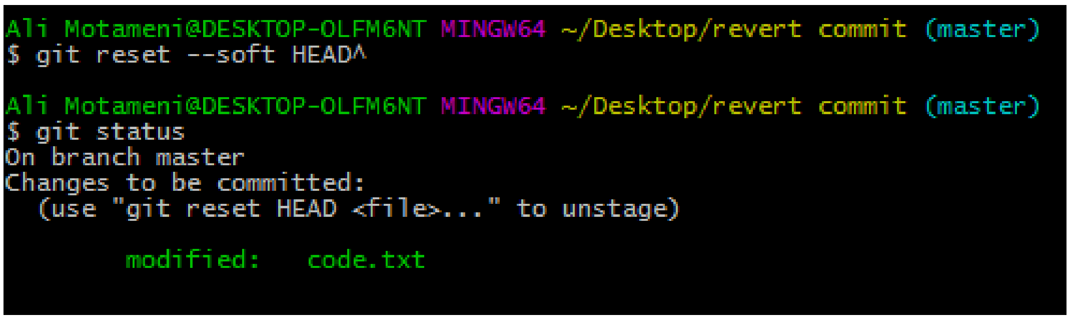
It means if you commit your change, your changes on this file is not saved. You can add this file in your stage with `git add code.txt` and then commit your change:


```
Ali Motameni@DESKTOP-OLFM6NT MINGW64 ~/Desktop/revert commit (master)
$ git add code.txt

Ali Motameni@DESKTOP-OLFM6NT MINGW64 ~/Desktop/revert commit (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   code.txt


Ali Motameni@DESKTOP-OLFM6NT MINGW64 ~/Desktop/revert commit (master)
$ git commit -m "Second commit"
[master d57266a] Second commit
 1 file changed, 1 insertion(+)
```

Undo last commit:

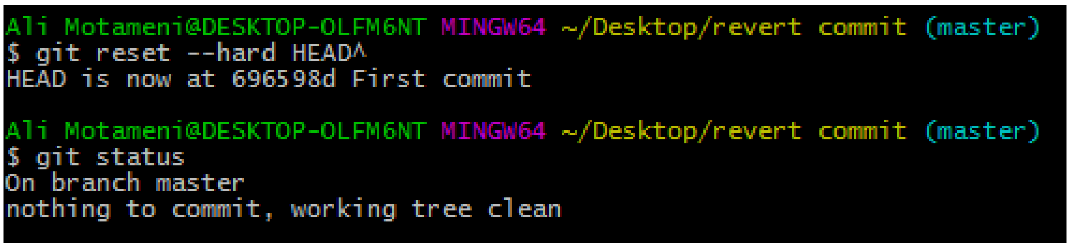
1. Now if we want to just undo commit without any other changes, we can use

   `git reset --soft HEAD^`


```
Ali Motameni@DESKTOP-OLFM6NT MINGW64 ~/Desktop/revert commit (master)
$ git reset --soft HEAD^

Ali Motameni@DESKTOP-OLFM6NT MINGW64 ~/Desktop/revert commit (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   code.txt
```

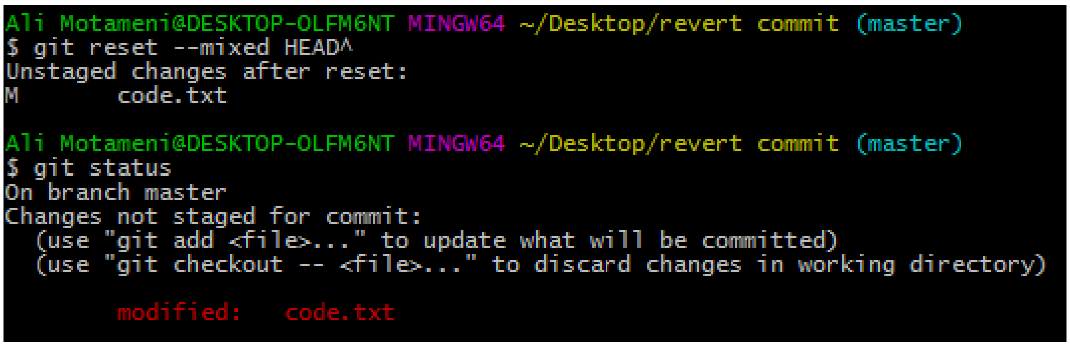
2. If we want to undo commit and its changes (***THIS IS DANGEROUS, because your change will lost***), we can use

   `git reset --hard HEAD^`


```
Ali Motameni@DESKTOP-OLFM6NT MINGW64 ~/Desktop/revert commit (master)
$ git reset --hard HEAD^
HEAD is now at 696598d First commit

Ali Motameni@DESKTOP-OLFM6NT MINGW64 ~/Desktop/revert commit (master)
$ git status
On branch master
nothing to commit, working tree clean
```

3. And if we want to undo commit and remove changes from stage, we can use

   `git reset --mixed HEAD^` or in a short form `git reset HEAD^`


```
Ali Motameni@DESKTOP-OLFM6NT MINGW64 ~/Desktop/revert commit (master)
$ git reset --mixed HEAD^
Unstaged changes after reset:
M       code.txt

Ali Motameni@DESKTOP-OLFM6NT MINGW64 ~/Desktop/revert commit (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   code.txt
```

Share Follow

edited Aug 5, 2019 at 19:24

community wiki
4 revs, 2 users 99%
Ali Motameni

90

Usually, you want to **undo** a commit because you made a mistake and you want to fix it - essentially what the OP did when he asked the question. Really, you actually want to **redo** a commit.

Most of the answers here focus on the command line. While the command line is the best way to use Git when you're comfortable with it, its probably a bit alien to those coming from other version control systems to Git.

Here's how to do it using a GUI. If you have Git installed, you already have everything you need to follow these instructions.

NOTE: I will assume here that you realised the commit was wrong before you pushed it. If you don't know what pushing means, then you probably haven't pushed. Carry on with the instructions. If you have pushed the faulty commit, the least risky way is just to follow up the faulty commit with a new commit that fixes things, the way you would do it in a version control system that does not allow you to rewrite history.

That said, here's how to fix your most recent fault commit using a GUI:

1. Navigate to your repository on the command line and start the GUI with `git gui`
2. Choose "Amend last commit". You will see your last commit message, the files you staged and the files you didn't.
3. Now change things to how you want them to look and click Commit.

Share Follow

edited Jul 23, 2021 at 20:52

community wiki

3 revs, 3 users 71%
Carl

86

If you want to revert the last commit but still want to keep the changes locally that were made in the commit, use this command:

```
git reset HEAD~1 --mixed
```

Share Follow

answered Aug 18, 2021 at 12:35

community wiki
Gerard de Visser

82

You can use:

```
git reset HEAD@{1}
```

This command will delete your wrong commit without a Git log.

Share Follow

edited Mar 25, 2017 at 8:23

community wiki
3 revs, 2 users 81%
Min Han

77

Undo the Last Commit

There are tons of situations where you really want to undo that last commit into your code. E.g. because you'd like to restructure it extensively - or even discard it altogether!

In these cases, the "reset" command is your best friend:

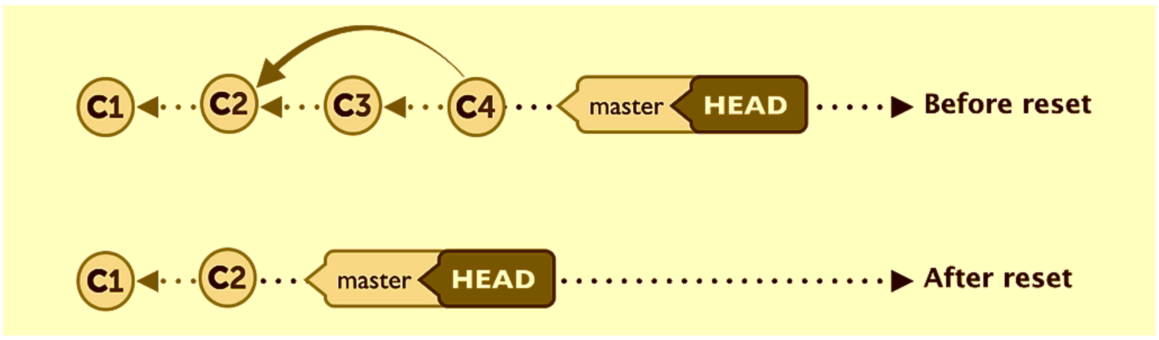
```
$ git reset --soft HEAD~1
```

The above command (reset) will rewind your current HEAD branch to the specified revision. In our example above, we'd like to return to the one before the current revision - effectively making our last commit undone.

Note the `--soft` flag: this makes sure that the changes in undone revisions are preserved. After running the command, you'll find the changes as uncommitted local modifications in your working copy.

If you don't want to keep these changes, simply use the `--hard` flag. Be sure to only do this when you're sure you don't need these changes any more.

```
$ git reset --hard HEAD~1
```



Share Follow

edited Jan 28, 2018 at 21:34

community wiki
3 revs, 2 users 86%
Mohit

"Working copy"? Is this a Git concept? Isn't it an SVN concept? – Peter Mortensen Jan 28, 2018 at 21:36

@PeterMortensen yes working copy, its a git concept though – Mohit May 4, 2018 at 19:46

75

Just undo the last commit:

```
git reset --soft HEAD~
```

Or undo the time before last time commit:

```
git reset --soft HEAD~2
```

Or undo any previous commit:

```
git reset --soft <commitID>
```

(you can get the commitID using `git reflog`)

When you undo a previous commit, remember to clean the workplace with

```
git clean
```

More details can be found in the docs: git-reset

Share Follow

edited May 18, 2015 at 16:55

community wiki
4 revs, 3 users 70%
steven

69

Before answering let's add some background, explaining what is this `HEAD`.

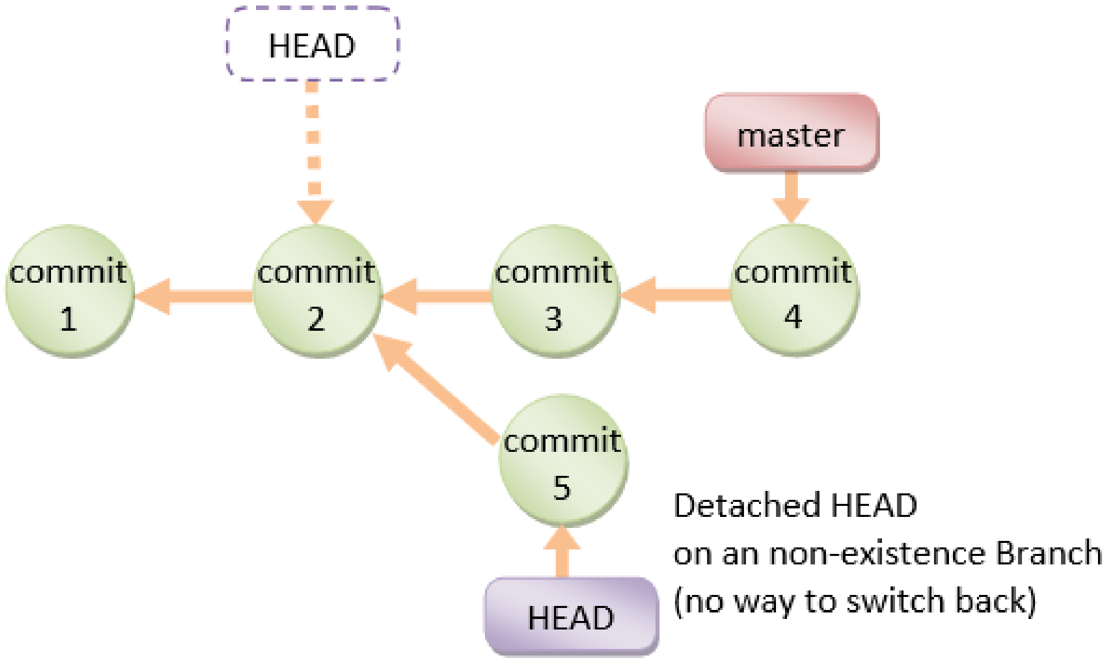
`First of all what is HEAD?`

`HEAD` is simply a reference to the current commit (latest) on the current branch.
There can only be a single `HEAD` at any given time. (excluding `git worktree`)

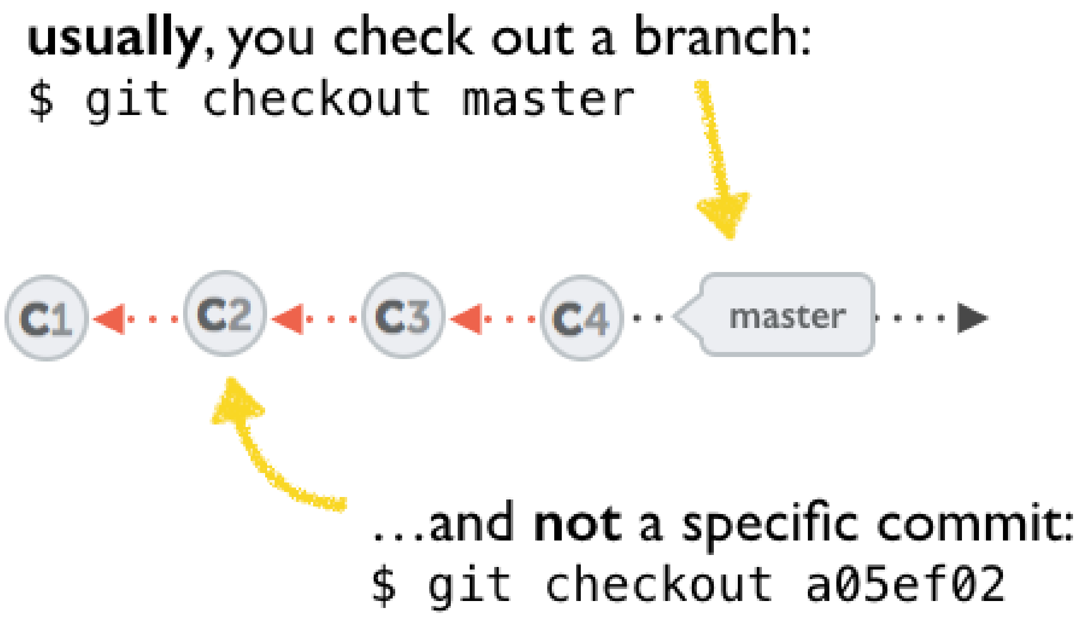
The content of `HEAD` is stored inside `.git/HEAD` and it contains the 40 bytes SHA-1 of the current commit.

`detached HEAD`

If you are not on the latest commit - meaning that `HEAD` is pointing to a prior commit in history its called ***`detached HEAD`***.



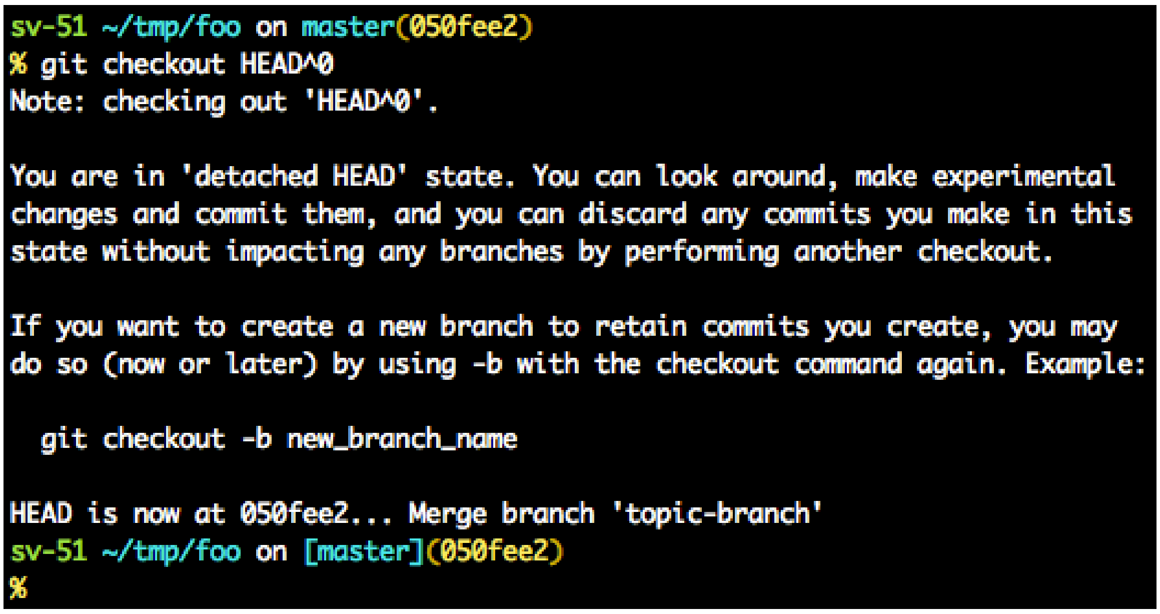
On the command line, it will look like this- SHA-1 instead of the branch name since the `HEAD` is not pointing to the tip of the current branch


usually, you check out a branch:

```
$ git checkout master
```

...and **not** a specific commit:

```
$ git checkout a05ef02
```



```
sv-51 ~/tmp/foo on master(050fee2)
% git checkout HEAD^0
Note: checking out 'HEAD^0'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b new_branch_name

HEAD is now at 050fee2... Merge branch 'topic-branch'
sv-51 ~/tmp/foo on [master](050fee2)
%
```

A few options on how to recover from a detached HEAD:

`git checkout`

```
git checkout <commit_id>
git checkout -b <new branch> <commit_id>
git checkout HEAD~X // x is the number of commits
t go back
```

This will checkout new branch pointing to the desired commit.
This command will checkout to a given commit.
At this point, you can create a branch and start to work from this point on.

```
# Checkout a given commit.
# Doing so will result in a `detached HEAD` which
mean that the `HEAD`
# is not pointing to the latest so you will need
to checkout branch
# in order to be able to update the code.
git checkout <commit-id>
```

```
# create a new branch forked to the given commit
git checkout -b <branch name>
```

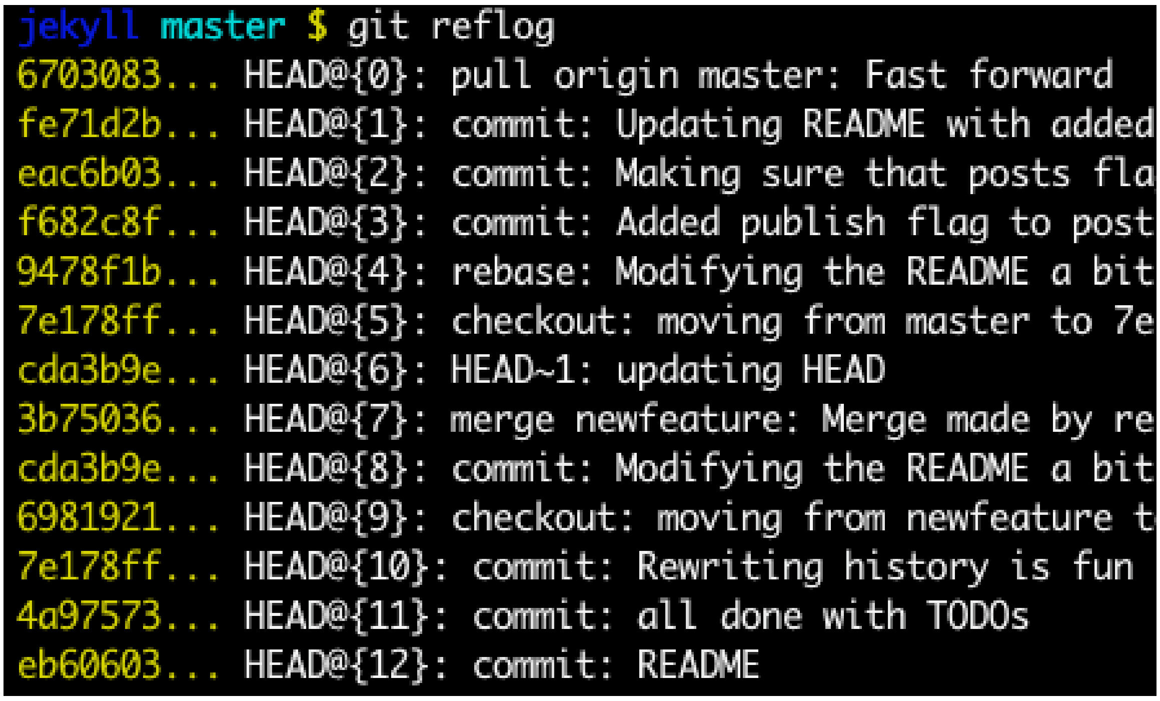
`git reflog`

You can always use the `reflog` as well.
`git reflog` will display any change which updated the `HEAD` and checking out the desired reflog entry will set the `HEAD` back to this commit.

Every time the HEAD is modified there will be a new entry in the `reflog`

```
git reflog
git checkout HEAD@{...}
```

This will get you back to your desired commit



```
jekyll master $ git reflog
6703083... HEAD@{0}: pull origin master: Fast forward
fe71d2b... HEAD@{1}: commit: Updating README with added
eac6b03... HEAD@{2}: commit: Making sure that posts fla
f682c8f... HEAD@{3}: commit: Added publish flag to post
9478f1b... HEAD@{4}: rebase: Modifying the README a bit
7e178ff... HEAD@{5}: checkout: moving from master to 7e
cda3b9e... HEAD@{6}: HEAD~1: updating HEAD
3b75036... HEAD@{7}: merge newfeature: Merge made by re
cda3b9e... HEAD@{8}: commit: Modifying the README a bit
6981921... HEAD@{9}: checkout: moving from newfeature t
7e178ff... HEAD@{10}: commit: Rewriting history is fun
4a97573... HEAD@{11}: commit: all done with TODOs
eb60603... HEAD@{12}: commit: README
```

`git reset --hard <commit_id>`

"Move" your HEAD back to the desired commit.

```
# This will destroy any local modifications.
# Don't do it if you have uncommitted work you want to
git reset --hard 0d1d7fc32

# Alternatively, if there's work to keep:
git stash
git reset --hard 0d1d7fc32
git stash pop
# This saves the modifications, then reapplies that pa
# You could get merge conflicts if you've modified thi
# changed since the commit you reset to.
```

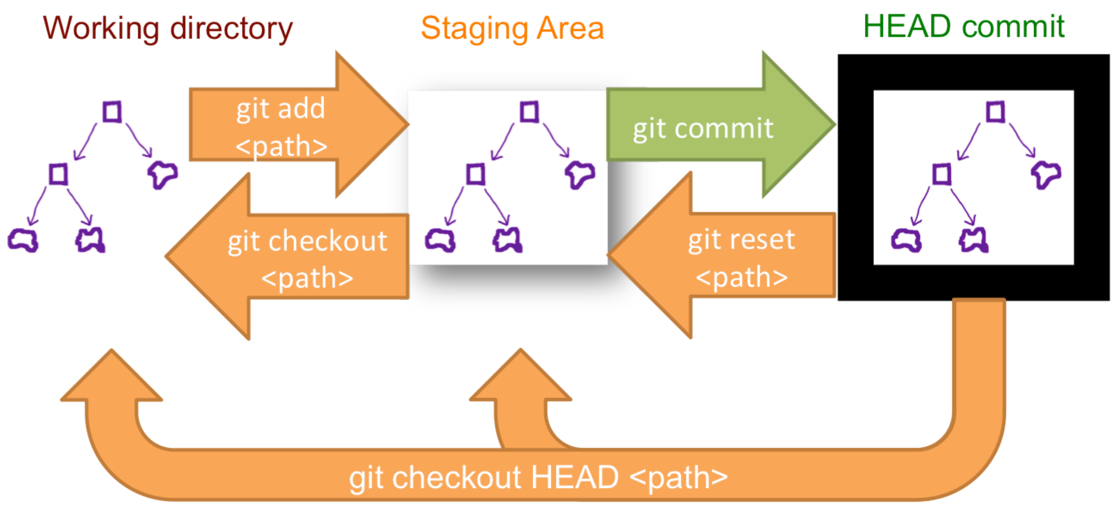
- Note: ([Since Git 2.7](#))
  you can also use the `git rebase --no-autostash` as well.

`git revert <sha-1>`

"Undo" the given commit or commit range.
The reset command will "undo" any changes made in the given commit.
A new commit with the undo patch will be committed while the original commit will remain in the history as well.

```
# add new commit with the undo of the original one.
# the <sha-1> can be any commit(s) or commit range
git revert <sha-1>
```

This schema illustrates which command does what.
As you can see there `reset && checkout` modify the `HEAD`.



Share Follow

edited Oct 14, 2019 at 14:42

community wiki
3 revs, 2 users 96%
CodeWizard

68

In my case I committed and pushed to the wrong branch, so what I wanted was to have all my changes back so I can commit them to a new correct branch, so I did this:

On the same branch that you committed and pushed, if you type "git status" you won't see anything new because you committed and pushed, now type:

```
git reset --soft HEAD~1
```

This will get all your changes(files) back in the stage area, now to get them back in the working directory(unstage) you just type:

```
git reset FILE
```

Where "File" is the file that you want to commit again. Now, this FILE should be in the working directory(unstaged) with all the changes that you did. Now you can change to whatever branch that you want and commit the changes in that branch. Of course, the initial branch that you committed is still there with all changes, but in my case that was ok, if it is not for you- you can look for ways to revert that commit in that branch.

Share Follow

edited Jul 23, 2021 at 21:09

community wiki
4 revs, 3 users 67%
FraK

66

Undo the last commit:

```
git reset --soft HEAD^ or git reset --soft HEAD~
```

This will undo the last commit.

Here `--soft` means reset into staging.

`HEAD~ or HEAD^` means to move to commit before HEAD.

Replace the last commit to new commit:

```
git commit --amend -m "message"
```

It will replace the last commit with the new commit.

Share Follow

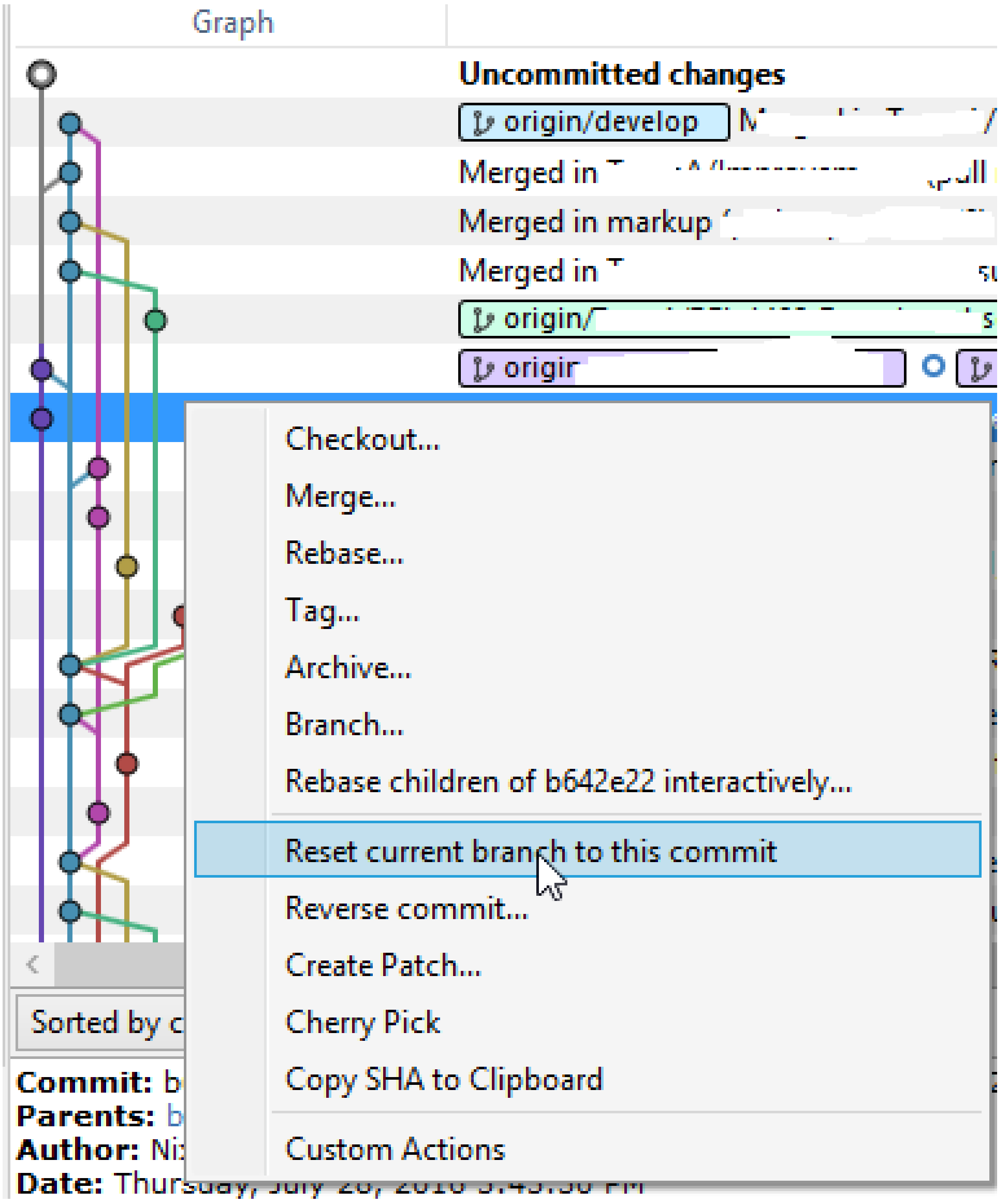
edited Mar 13, 2019 at 5:18

community wiki
2 revs, 2 users 94%
Ankit Patidar

63

If you are working with **[SourceTree](#)**, this will help you.

Right click on the commit then **select** "*Reset (current branch)/master to this commit*" and last **select** *"Soft" reset*.



Share Follow

edited Mar 25, 2017 at 9:22

community wiki
2 revs, 2 users 78%
Alexandr

To undo your local commit you use `git reset <commit>`. Also [that tutorial]() is very helpful to show you how it works.

61

Alternatively, you can use `git revert <commit>`: reverting should be used when you want to add another commit that rolls back the changes (but keeps them in the project history).

Share Follow

edited Nov 10, 2018 at 9:26

community wiki
5 revs, 4 users 53%
mfathy00

Be **extra** careful when reverting merge commits. You may lose your commits. Read about what Linus says about that: kernel.org/pub/software/scm/git/docs/howto/…
– Eugen Konkov Dec 15, 2016 at 16:25

Note that for git reset <commit>, use the commit sha for the PREVIOUS commit that you want to reset to, not the commit that you want to unstage. – Rich G Jul 7, 2022 at 15:25

59

Suppose you made a wrong commit locally and pushed it to a remote repository. You can undo the mess with these two commands:

First, we need to correct our local repository by going back to the commit that we desire:

```
git reset --hard <previous good commit id where
you want the local repository  to go>
```

Now we forcefully push this good commit on the remote repository by using this command:

```
git push --force-with-lease
```

The 'with-lease' version of the force option it will prevent accidental deletion of new commits you do not know about (i.e. coming from another source since your last pull).

Share Follow

edited Dec 13, 2019 at 16:40

community wiki
5 revs, 5 users 55%
KawaiKx

1 this worked for me the best, since I had already pushed the bad commit up to github – AeroHil May 6, 2019 at 20:36

54

VISUAL STUDIO USERS (2015, etc.)

If you cannot synchronise in Visual Studio as you are not allowed to push to a branch like "development" then as much as I tried, in Visual Studio NEITHER the **REVERT** NOR the **RESET** (hard or soft) would work.

Per the answer with TONS OF VOTES:

Use this at the command prompt of root of your project to nuke anything that will attempt to get pushed:

```
git reset --hard HEAD~1
```

Backup or zip your files just in case you don't wish to lose any work, etc...

Share Follow

edited Mar 25, 2017 at 8:22

community wiki
3 revs, 2 users 81%
Tom Stickel

54

Everybody comments in such a complicated manner.

If you want to remove the last commit from your branch, the simplest way to do it is:

```
git reset --hard HEAD~1
```

Now to actually push that change to get rid of your last commit, you have to

```
git push --force
```

And that's it. This will remove your last commit.

Share Follow

edited Jan 18, 2021 at 20:44

community wiki
2 revs, 2 users 80%
Stipe

6 But keep in mind that --hard will completely discard all the changes that were made in the last commit as well as the content of the index. – CloudJR Mar 14, 2020 at 11:58

Yes, in the case we need to just fix the commit, we shoud not use `--hard`, but leave it with the default `--soft` so we can remove and add stuff before the ``git push -f` – Maf Mar 11, 2021 at 16:29

53

A Typical Git Cycle

In speaking of Git-related commands in the previous answers, I would like to share my typical Git cycles with all readers which may helpful. Here is how I work with Git,

1. Cloning the first time from the remote server

   `git clone $project`

2. Pulling from remote (when I don't have a pending local commit to push)

   `git pull`

3. Adding a new local file1 into $to_be_committed_list (just imagine $to_be_committed_list means `staged` area)

`git add $file1`

4. Removing mistakenly added file2 from $to_be_committed_list (assume that file2 is added like step 3, which I didn't want)

   `git reset $file2`

5. Committing file1 which is in $to_be_committed_list

   `git commit -m "commit message description"`

6. Syncing local commit with remote repository before pushing

   `git pull --rebase`

7. Resolving when conflict occurs prerequisite configure mergetool

   `git mergetool #resolve merging here, also can manually merge`

8. Adding conflict-resolved files, let's say `file1`:

   `git add $file1`

9. Continuing my previous rebase command

   `git rebase --continue`

10. Pushing ready and already synced last local commit

    `git push origin head:refs/for/$branch # branch = master, dev, etc.`

Share Follow

edited Jun 20, 2020 at 9:12

community wiki
2 revs, 2 users 91%
Sazzad Hissain Khan

What if I am working on a fork, so basically I have 2 remotes actual repo e.g. incubator-mxnet and my forked repo ChaiBapchya/incubator-mxnet So in such a case, how can I solve merge conflicts from local to my forked repo branch – Chaitanya Bapat Oct 30, 2018 at 19:18

50

In these cases, the "reset" command is your best friend:

```
git reset --soft HEAD~1
```

Reset will rewind your current HEAD branch to the specified revision. In our example above, we'd like to return to the one before the current revision - effectively making our last commit undone.

Note the --soft flag: this makes sure that the changes in undone revisions are preserved. After running the command, you'll find the changes as uncommitted local modifications in your working copy.

If you don't want to keep these changes, simply use the --hard flag. Be sure to only do this when you're sure you don't need these changes anymore.

```
git reset --hard HEAD~1
```

Share Follow

edited Jun 17, 2020 at 9:54

community wiki
2 revs, 2 users 91%
Ankit Tiwari

48

In order to get rid of (all the changes in) last commit, last 2 commits and last n commits:

```
git reset --hard HEAD~1
git reset --hard HEAD~2
...
git reset --hard HEAD~n
```

And, to get rid of anything after a specific commit:

```
git reset --hard <commit sha>
```

e.g.,

```
git reset --hard 0d12345
```

Be careful with the hard option: it deletes the local changes in your repo as well and reverts to the previous mentioned commit. You should only run this if you are sure you messed up in your last commit(s) and would like to go back in time.

As a side-note, about 7 letters of the commit hash is enough, but in bigger projects, you may need up to 12 letters for it to be unique. You can also use the entire commit SHA if you prefer.

The above commands work in GitHub for Windows as well.

Share Follow

edited Jul 23, 2021 at 21:15

community wiki
3 revs, 2 users 84%
Alisa

47

Remove a wrong commit that is already pushed to Github

```
git push origin +(previous good commit id):(branch
name)
```

Please specify the last good commit id you would like to reset back in Github.

For example. If latest commit id is wrong then specify the previous commit id in above git command with the branch name.

You can get previous commit id using `git log`

Share Follow

edited Mar 13, 2019 at 5:22

community wiki
2 revs, 2 users 93%
V V

43

You need to do the easy and fast

```
    git commit --amend
```

if it's a private branch or

```
    git commit -m 'Replace .class files with .java
files'
```

if it's a shared or public branch.

Share Follow

edited Mar 25, 2017 at 8:14

community wiki
2 revs, 2 users 89%
Yahs Hef

43

I got the commit ID from `bitbucket` and then did:

```
git checkout commitID .
```

Example:

```
git checkout 7991072 .
```

And it reverted it back up to that working copy of that commit.

Share Follow

edited Mar 25, 2017 at 8:21

community wiki
4 revs, 4 users 48%
ioopl

Note: checking out '5456cea9'. You are in 'detached HEAD' state. You can look around, make experimental changes and commit them, and you can discard any commits you make in this state without impacting any branches by performing another checkout. If you want to create a new branch to retain commits you create, you may do so (now or later) by using -b with the checkout command again. Example: git checkout -b <new-branch-name> HEAD is now at 5456cea... Need to delete Exclusions.xslt from Documentation folder. - Delete What should i do after this – cSharma May 22, 2019 at 14:02

37

Do as the following steps.

Step 1

Hit `git log`

From the list of log, find the last commit hash code and then enter:

Step 2

```
git reset <hash code>
```

Share Follow

edited Jul 23, 2021 at 21:57

community wiki
4 revs, 3 users 55%
Paras Korat

After this, how can I fix the wrong commit to exclude wrong files? – Maf Mar 11, 2021 at 16:23

34

Use this command

```
git checkout -b old-state 0d1d7fc32
```

Share Follow

answered Jul 6, 2015 at 8:30

community wiki
Jishnu Sukumaran

34

In order to remove some files from a Git commit, use the “git reset” command with the “–soft” option and specify the commit before HEAD.

```
$ git reset --soft HEAD~1
```

When running this command, you will be presented with the files from the most recent commit (HEAD) and you will be able to commit them.

Now that your files are in the staging area, you can remove them (or unstage them) using the “git reset” command again.

```
$ git reset HEAD <file>
```

Note: this time, you are resetting from HEAD as you simply want to exclude files from your staging area

If you are simply not interested in this file any more, you can use the “git rm” command in order to delete the file from the index (also called the staging area).

```
$ git rm --cached <file>
```

When you are done with the modifications, you can simply commit your changes again with the “–amend” option.

```
$ git commit --amend
```

To verify that the files were correctly removed from the repository, you can run the “git ls-files” command and check that the file does not appear in the file (if it was a new one of course)

```
$ git ls-files

 <file1>
 <file2>
```

Remove a File From a Commit using Git Restore

Since Git 2.23, there is a new way to remove files from commit, but you will have to make sure that you are using a Git version greater or equal than 2.23.

```
$ git --version
```

Git version 2.24.1

Note: Git 2.23 was released in August 2019 and you may not have this version already available on your computer.

To install newer versions of Git, you can check this tutorial. To remove files from commits, use the “git restore” command, specify the source using the “–source” option and the file to be removed from the repository.

For example, in order to remove the file named “myfile” from the HEAD, you would write the following command

```
$ git restore --source=HEAD^ --staged  -- <file>
```

As an example, let’s pretend that you edited a file in your most recent commit on your “master” branch.

The file is correctly committed but you want to remove it from your Git repository.

To remove your file from the Git repository, you want first to restore it.

```
$ git restore --source=HEAD^ --staged  -- newfile

$ git status
```

On branch 'master'

Your branch is ahead of 'origin/master' by 1 commit. (use "git push" to publish your local commits)

Changes to be committed:

```
 (use "git restore --staged <file>..." to unstage)
    modified:   newfile
```

Changes not staged for commit:

```
 (use "git add <file>..." to update what will be
committed)
 (use "git restore <file>..." to discard changes
in working directory)
      modified:   newfile
```

As you can see, your file is back to the staging area.

From there, you have two choices, you can choose to edit your file in order to re-commit it again, or to simply delete it from your Git repository.

Remove a File from a Git Repository

In this section, we are going to describe the steps in order to remove the file from your Git repository.

First, you need to unstage your file as you won’t be able to remove it if it is staged.

To unstage a file, use the “git reset” command and specify the HEAD as source.

```
$ git reset HEAD newfile
```

When your file is correctly unstaged, use the “git rm” command with the “–cached” option in order to remove this file from the Git index (this won’t delete the file on disk)

```
$ git rm --cached newfile
```

rm 'newfile'

Now if you check the repository status, you will be able to see that Git staged a deletion commit.

```
$ git status
```

On branch 'master'

Your branch is ahead of 'origin/master' by 1 commit. (use "git push" to publish your local commits)

Changes to be committed:

```
  (use "git restore --staged <file>..." to unstage)
     deleted:    newfile
```

Now that your file is staged, simply use the “git commit” with the “–amend” option in order to amend the most recent commit from your repository.

```
`$ git commit --amend

 [master 90f8bb1] Commit from HEAD
  Date: Fri Dec 20 03:29:50 2019 -0500
  1 file changed, 2 deletions(-)
  delete mode 100644 newfile
```

`As you can see, this won’t create a new commit but it will essentially modify the most recent commit in order to include your changes.

Remove a Specific File from a Git Commit

In some cases, you don’t want all the files to be staged again: you only one to modify one very specific file of your repository.

In order to remove a specific file from a Git commit, use the “git reset” command with the “–soft” option, specify the commit before HEAD and the file that you want to remove.

```
$ git reset HEAD^ -- <file>
```

When you are done with the modifications, your file will be back in the staging area.

First, you can choose to remove the file from the staging area by using the “git reset” command and specify that you want to reset from the HEAD.

```
$ git reset HEAD <file>
```

Note: it does not mean that you will lose the changes on this file, just that the file will be removed from the staging area.

If you want to completely remove the file from the index, you will have to use the “git rm” command with the “–cached” option.

```
$ git reset HEAD <file>
```

In order to make sure that your file was correctly removed from the staging area, use the “git ls-files” command to list files that belong to the index.

```
$ git ls-files
```

When you are completely done with your modifications, you can amend the commit you removed the files from by using the “git commit” command with the “–amend” option.

```
$ git commit --amend
```

Share Follow

edited Jun 23, 2021 at 22:33

community wiki
3 revs, 2 users 89%
MontresorXPL

What's the difference between using `git rm file` and git checkout file`? – Maf Mar 11, 2021 at 16:37

33

```
git reset --soft HEAD~1
```

Reset will rewind your current HEAD branch to the specified revision.

Note the `--soft` flag: this makes sure that the changes in undone revisions are preserved. After running the command, you'll find the changes as uncommitted local modifications in your working copy.

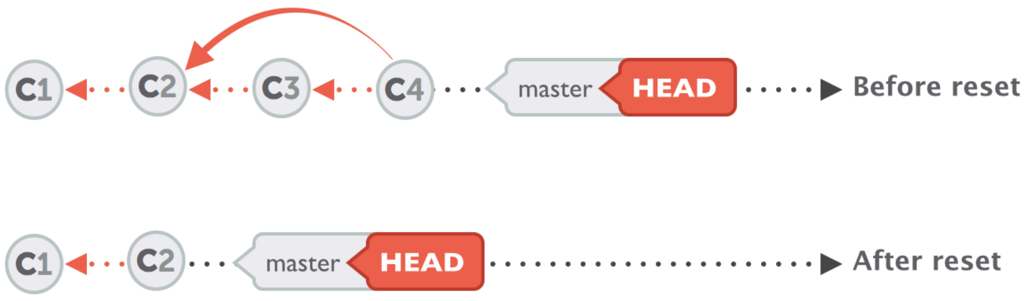
If you don't want to keep these changes, simply use the `--hard` flag. Be sure to only do this when you're sure you don't need these changes anymore.

```
git reset --hard HEAD~1
```

> Undoing Multiple Commits

```
git reset --hard 0ad5a7a6
```

Keep in mind, however, that using the reset command undoes all commits that came after the one you returned to:



Share Follow

edited Jul 10, 2019 at 20:25

community wiki
2 revs, 2 users 80%
Abu Bakr

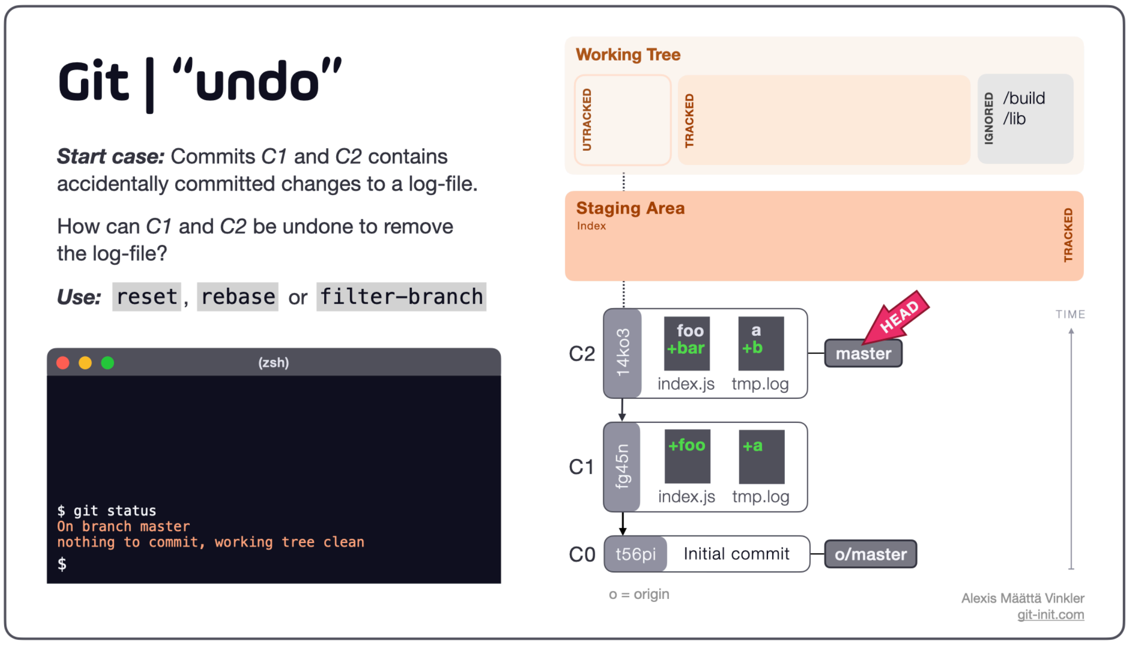
33

Undoing a series of local commits

> OP: How do I undo the most recent local commits in Git? I accidentally committed the wrong files [as part of several commits].

Start case

There are several ways to "undo" as series of commits, depending on the outcome you're after. Considering the start case below, `reset`, `rebase` and `filter-branch` can all be used to rewrite your history.



Git | "undo"

Start case: Commits *C1* and *C2* contains accidentally committed changes to a log-file.

How can *C1* and *C2* be undone to remove the log-file?

Use: `reset`, `rebase` or `filter-branch`

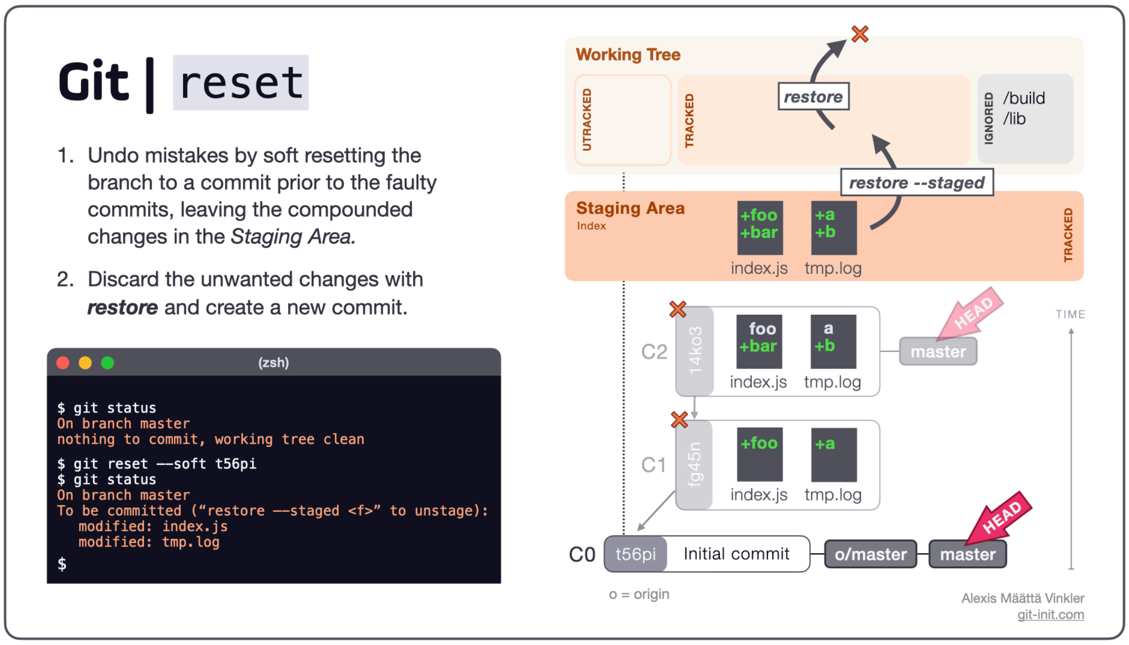
```
$ git status
On branch master
nothing to commit, working tree clean
$
```

How can C1 and C2 be undone to remove the `tmp.log` file from each commit?

In the examples below, absolute commit references are used, but it works the same way if you're more used to relative references (i.e. `HEAD~2` or `HEAD@{n}`).

Alternative 1: `reset`

```
$ git reset --soft t56pi
```


Git | reset

1. Undo mistakes by soft resetting the branch to a commit prior to the faulty commits, leaving the compounded changes in the *Staging Area.*
2. Discard the unwanted changes with ***restore*** and create a new commit.

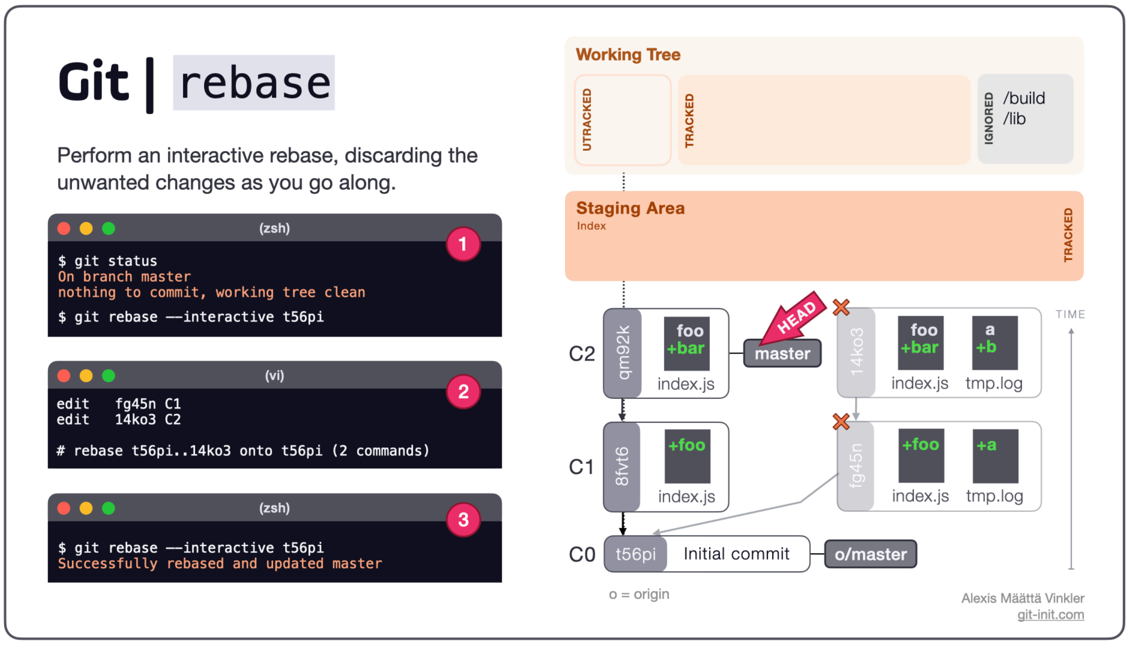
```
$ git status
On branch master
nothing to commit, working tree clean
$ git reset --soft t56pi
$ git status
On branch master
To be committed ("restore --staged <f>" to unstage):
   modified: index.js
   modified: tmp.log
$
```

With `reset`, a branch can be reset to a previous state, and any compounded changes be reverted to the *Staging Area*, from where any unwanted changes can then be discarded.

Note: As `reset` clusters all previous changes into the *Staging Area*, individual commit meta-data is lost. If this is not OK with you, chances are you're probably better off with `rebase` or `filter-branch` instead.

Alternative 2: `rebase`

```
$ git rebase --interactive t56pi
```


Git | rebase

Perform an interactive rebase, discarding the unwanted changes as you go along.

1 (zsh)
```
$ git status
On branch master
nothing to commit, working tree clean
$ git rebase —interactive t56pi
```

2 (vi)
```
edit   fg45n C1
edit   14ko3 C2

# rebase t56pi..14ko3 onto t56pi (2 commands)
```

3 (zsh)
```
$ git rebase —interactive t56pi
Successfully rebased and updated master
```

Using an interactive `rebase` each offending commit in the branch can be rewritten, allowing you to modify and discard unwanted changes. In the infographic above, the source tree on the right illustrates the state post `rebase`.

Step-by-step

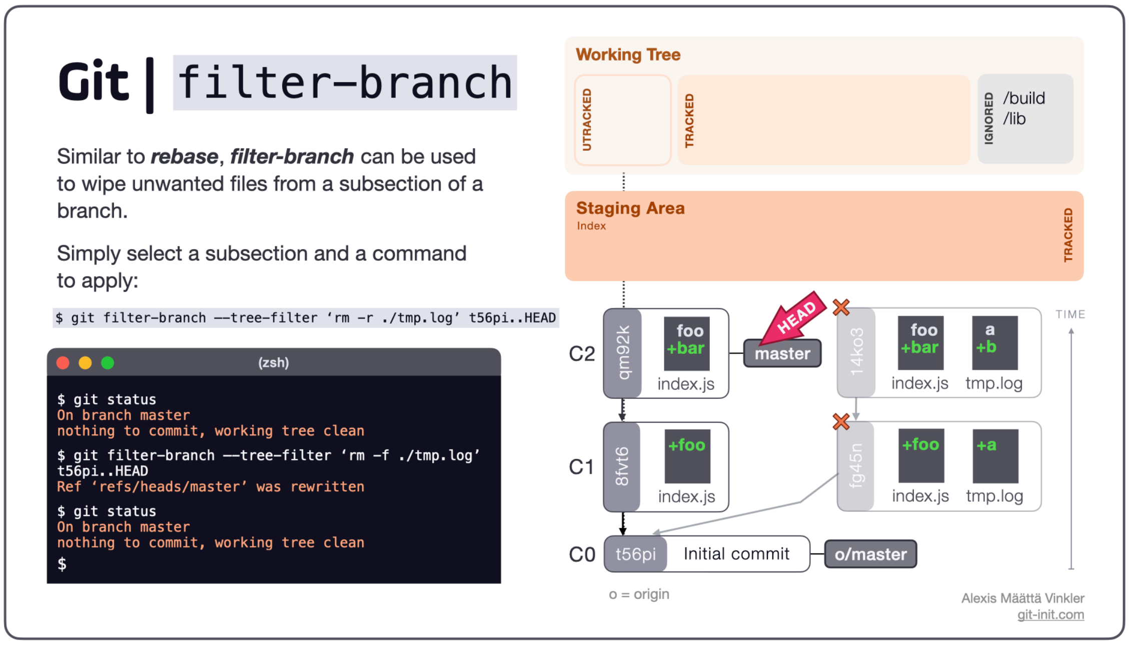
1. Select from which commit the rebase should be based (e.g. `t56pi`)
2. Select which commits you'd like to change by replacing `pick` with `edit`. Save and close.
3. Git will now stop on each selected commit allowing you to reset `HEAD`, remove the unwanted files, and create brand new commits.

Note: With `rebase` much of the commit meta data is kept, in contrast to the `reset` alternative above. This is most likely a preferred option, if you want to keep much of your history but only remove the unwanted files.

Alternative 3: `filter-branch`

```
$ git filter-branch --tree-filter 'rm -r
./tmp.log' t56pi..HEAD
```

Above command would filter out the file `./tmp.log` from all commits in the desired range `t56pi..HEAD` (assuming our initial start case from above). See below illustration for clarity.



Git | filter–branch

Similar to ***rebase***, ***filter-branch*** can be used to wipe unwanted files from a subsection of a branch.

Simply select a subsection and a command to apply:

```
$ git filter–branch ––tree–filter ‘rm –r ./tmp.log’ t56pi..HEAD
```

```
(zsh)
$ git status
On branch master
nothing to commit, working tree clean
$ git filter–branch ––tree–filter ‘rm –f ./tmp.log’
t56pi..HEAD
Ref ‘refs/heads/master’ was rewritten
$ git status
On branch master
nothing to commit, working tree clean
$
```

Similar to `rebase`, `filter-branch` can be used to wipe unwanted files from a subsection of a branch. Instead of manually editing each commit through the rebase process, `filter-branch` can automatically preformed the desired action on each commit.

Note: Just like `rebase`, `filter-branch` would preserve the rest of the commit meta-data, by only discarding the desired file. Notice how *C1* and *C2* have been rewritten, and the log-file discarded from each commit.

Conclusion

Just like anything related to software development, there are multiple ways to achieve the same (or similar) outcome for a give problem. You just need to pick the one most suitable for your particular case.

Finally - a friendly advice

Do note that all three alternatives above rewrites the history completely. Unless you know exactly what you're doing and have good communication within your team - only rewrite commits that have not yet been published remotely!

Source: All examples above are borrowed from this blog.

Share Follow

answered Oct 5, 2021 at 13:26

community wiki
Alexis Määttä Vinkler

32

You can undo your Git commits in two ways: First is you can use `git revert`, if you want to keep your commit history:

```
git revert HEAD~3
git revert <hashcode of commit>
```

Second is you can use `git reset`, which would delete all your commit history and bring your head to commit where you want it.

```
git reset <hashcode of commit>
git reset HEAD~3
```

You can also use the `--hard` keyword if any of it starts behaving otherwise. But, I would only recommend it until it's extremely necessary.

edited Oct 19, 2018 at 18:10

community wiki
2 revs, 2 users 79%
Shwetank

32

You can always do a `git checkout <SHA code>` of the previous version and then commit again with the new code.

edited Nov 10, 2018 at 9:27

community wiki
3 revs, 3 users 40%
shreshta bm

31

Here is site: `Oh shit, git!`.

Here are many recipes how to undo things in Git. Some of them:

> Oh shit, I need to change the message on my last commit!

```
git commit --amend
# follow prompts to change the commit message
```

> Oh shit, I accidentally committed something to master that should have been on a brand new branch!

```
# Create a new branch from the current state of
master
git branch some-new-branch-name
# Remove the commit from the master branch
git reset HEAD~ --hard
git checkout some-new-branch-name
# Your commit lives in this branch now :)
```

Share Follow

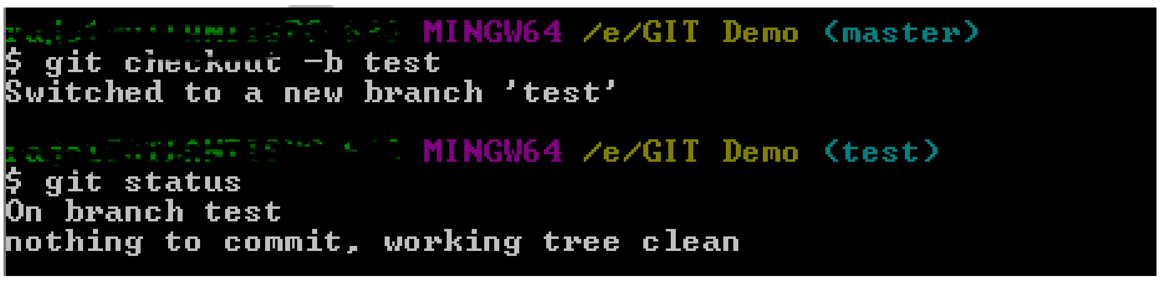
edited Oct 19, 2018 at 18:08

community wiki
2 revs, 2 users 78%
Eugen Konkov

30

You can undo your commits from the local repository. Please follow the below scenario.

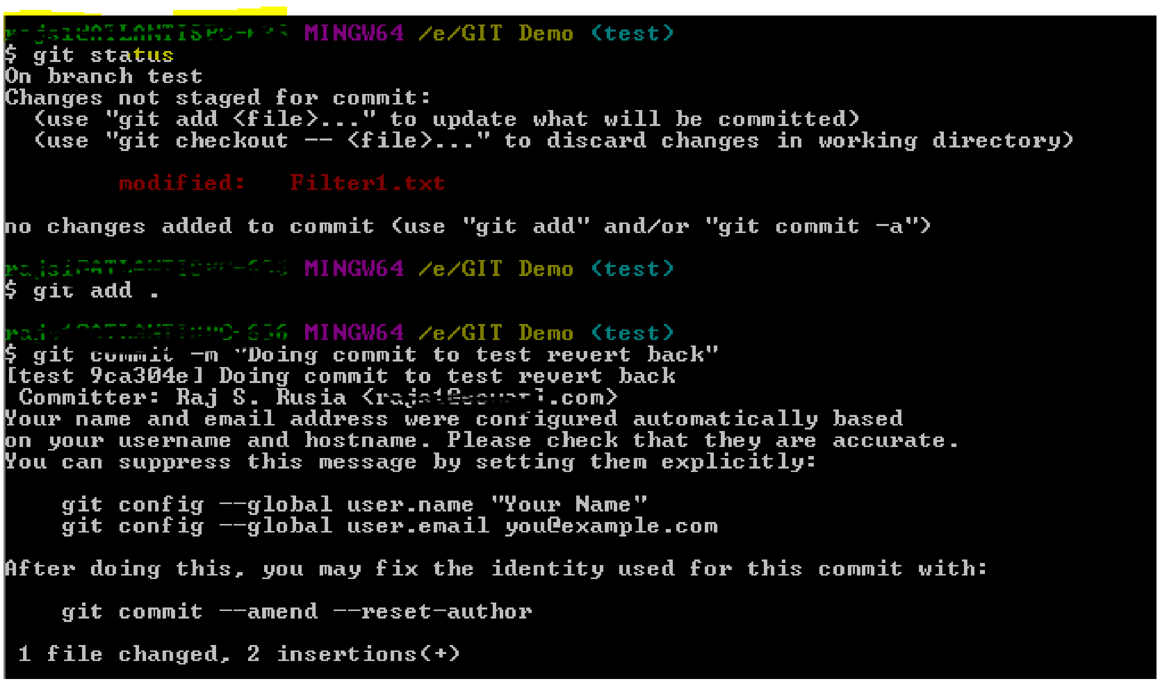
In the below image I check out the 'test' branch (using Git command `git checkout -b test`) as a local and check status (using Git command `git status`) of local branch that there is nothing to commit.



```
[redacted] MINGW64 /e/GIT Demo (master)
$ git checkout -b test
Switched to a new branch 'test'

[redacted] MINGW64 /e/GIT Demo (test)
$ git status
On branch test
nothing to commit, working tree clean
```

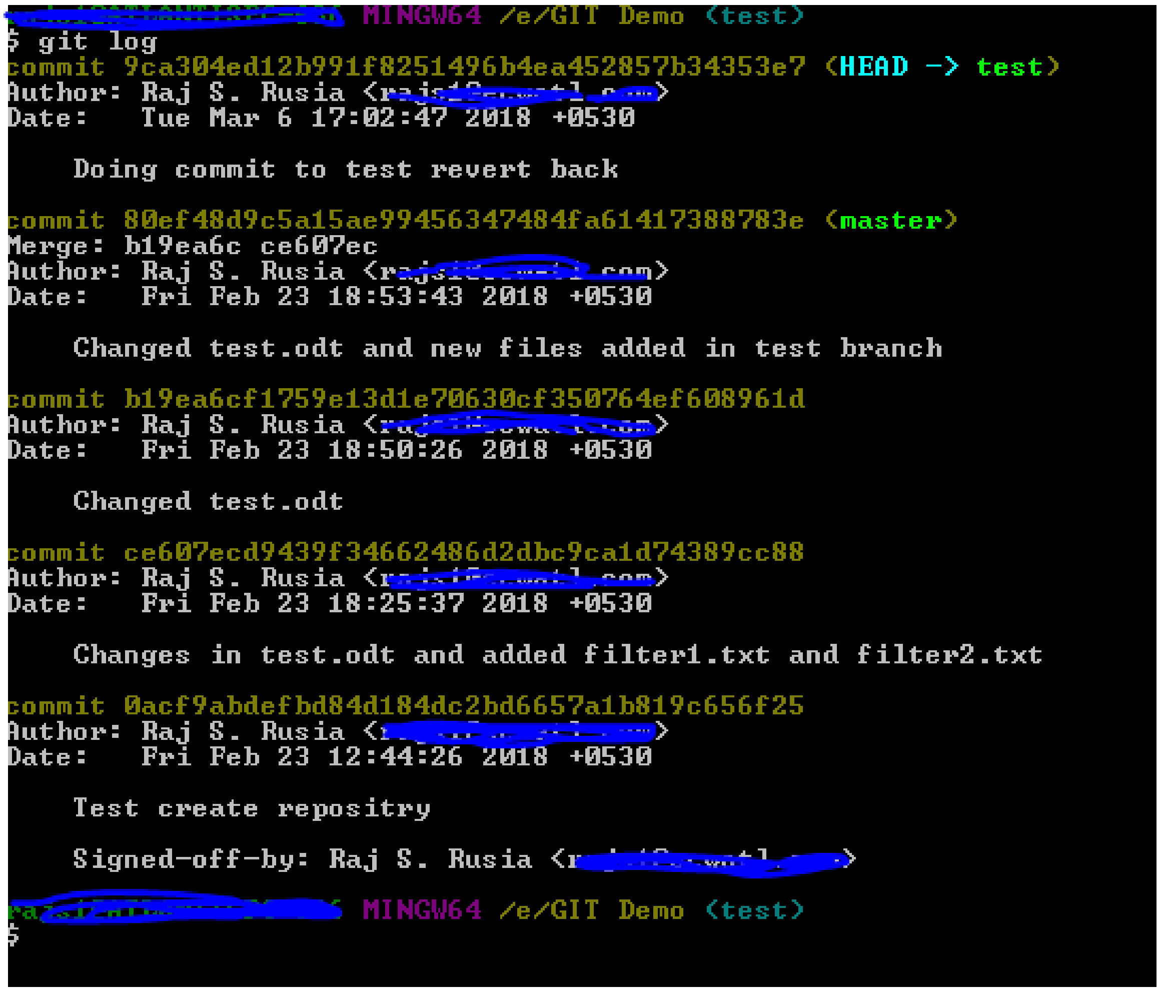
In the next image image you can see here I made a few changes in **Filter1.txt** and added that file to the staging area and then committed my changes with some message (using Git command `git commit -m "Doing commit to test revert back"`).

"-m is for commit message"



```
[redacted] MINGW64 /e/GIT Demo (test)
$ git status
On branch test
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   Filter1.txt

no changes added to commit (use "git add" and/or "git commit -a")

[redacted] MINGW64 /e/GIT Demo (test)
$ git add .

[redacted] MINGW64 /e/GIT Demo (test)
$ git commit -m "Doing commit to test revert back"
[test 9ca304e] Doing commit to test revert back
 Committer: Raj S. Rusia <[redacted].com>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly:

    git config --global user.name "Your Name"
    git config --global user.email you@example.com

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 1 file changed, 2 insertions(+)
```

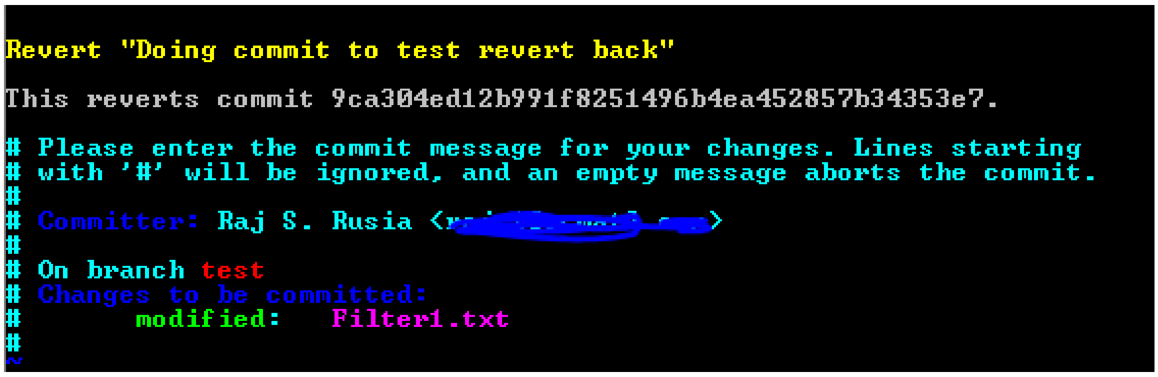
In the next image you can see your commits log whatever you have made commits (using Git command `git log`).



```
[redacted] MINGW64 /e/GIT Demo (test)
$ git log
commit 9ca304ed12b991f8251496b4ea452857b34353e7 (HEAD -> test)
Author: Raj S. Rusia <[redacted]>
Date:   Tue Mar 6 17:02:47 2018 +0530

    Doing commit to test revert back

commit 80ef48d9c5a15ae99456347484fa61417388783e (master)
Merge: b19ea6c ce607ec
Author: Raj S. Rusia <[redacted]>
Date:   Fri Feb 23 18:53:43 2018 +0530

    Changed test.odt and new files added in test branch

commit b19ea6cf1759e13d1e70630cf350764ef608961d
Author: Raj S. Rusia <[redacted]>
Date:   Fri Feb 23 18:50:26 2018 +0530

    Changed test.odt

commit ce607ecd9439f34662486d2dbc9ca1d74389cc88
Author: Raj S. Rusia <[redacted]>
Date:   Fri Feb 23 18:25:37 2018 +0530

    Changes in test.odt and added filter1.txt and filter2.txt

commit 0acf9abdefbd84d184dc2bd6657a1b819c656f25
Author: Raj S. Rusia <[redacted]>
Date:   Fri Feb 23 12:44:26 2018 +0530

    Test create repositry

    Signed-off-by: Raj S. Rusia <[redacted]>

[redacted] MINGW64 /e/GIT Demo (test)
$
```

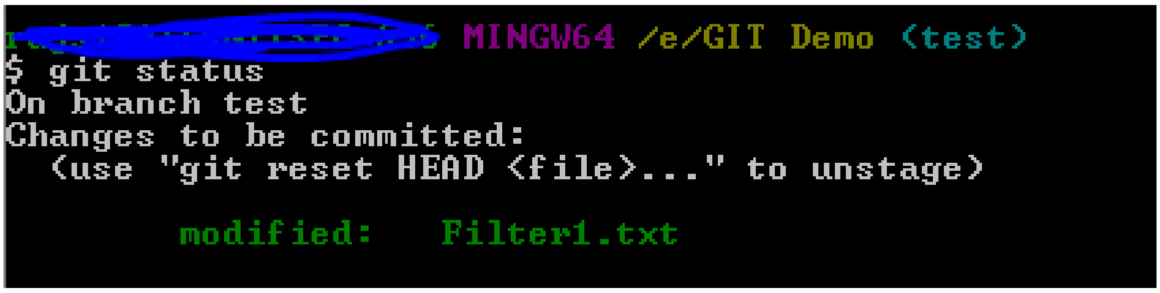
So in the above image you can see the commit id with each commit and with your commit message now whatever commit you want to revert back or undo copy that commit id and hit the below Git command, `git revert {"paste your commit id"}`. Example:

```
git revert
9ca304ed12b991f8251496b4ea452857b34353e7
```



```
Revert "Doing commit to test revert back"

This reverts commit 9ca304ed12b991f8251496b4ea452857b34353e7.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Committer: Raj S. Rusia <[redacted]>
#
# On branch test
# Changes to be committed:
#       modified:   Filter1.txt
#
~
```

I have reverted back my last commit. Now if you check your Git status, you can see the modified file which is **Filter1.txt** and yet to commit.



```
[redacted] MINGW64 /e/GIT Demo (test)
$ git status
On branch test
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   Filter1.txt
```

Share Follow

edited Oct 19, 2018 at 18:07

community wiki
5 revs, 4 users 67%
Raj S. Rusia

30

The simplest way to undo the last commit is

```
git reset HEAD^
```

This will bring the project state before you have made the commit.

Share Follow

answered Dec 10, 2018 at 7:05

community wiki
Arun Karnawat